User Manual

32

# V850E2/Fx4-G

## User's Manual: Hardware

32-bit Microcontroller

**V850E2/FF4-G**

µPD70F4177
µPD70F4178

**V850E2/FG4-G**

µPD70F4179
µPD70F4180



Renesas Electronics
www.renesas.com

R01UH0366EJ0100, Rev. 1.00
February 28, 2013

## Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics.

The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

| | |
|---|---|
| “Standard”: | Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots. |
| “High Quality”: | Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support. |
| “Specific”: | Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life. |

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

| | |
|---|---|
| (Note 1) | “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries. |
| (Note 2) | “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics. |

# General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   – The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at power-on

   The state of the product is undefined at the moment when power is supplied.

   – The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

     In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

     In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited.

   – The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   – When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between products

   Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.

   – The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

# Table of Contents

# How to use this manual

## Purpose and target readers

This manual is designed to provide the user with an understanding of the hardware functions of the microcontroller. It is intended for users designing application systems incorporating the microcontroller. A basic knowledge of electric circuits, logical circuits, and microcontrollers is necessary in order to use this manual.

### Special notations

Following special notations are used throughout this document:

**Note** Additional remark or tip

**Caution** Item deserving extra attention

### Electrical specifications

This manual does not present any electrical specifications.
Refer to the Data Sheet for detailed definitions of all electrical properties.
For information about the Data Sheet document, refer to the section *"Related Documents"* in the chapter *"Introduction"*.

### Additional documents

Following types of documents are available for the V850E2/Fx4-G microcontrollers. Make sure to refer to the latest versions of these documents. The newest versions of the documents listed may be obtained from the Renesas Electronics Web site.

| Document Type | Description | Document |
|---|---|---|
| Data sheet | Hardware overview and electrical characteristics | Refer to the section *"Related Documents"* in the chapter *"Introduction"* |
| User manual: Hardware | Hardware specifications (pin assignments, memory maps, functional modules specifications and operation description) Note: Refer to the application notes for details on using functional modules. | |
| User manual: 32-bit Microprocessor Core Architecture | Description of CPU, its instruction set and processor protection functions | |
| Application note | Information on using peripheral functions and application examples, sample programs and information on writing programs in assembly language and C | Available from Renesas Electronics Web site |
| Renesas technical update | Product specifications, updates on documents, etc. | |

# Content of this manual

In the following brief hints are given where to find certain information about the V850E2/Fx4-G microcontrollers.

**Product overview**

Refer to the chapter “Introduction” for an overview of the features of all target microcontrollers and their block diagrams.
Order codes for all devices and a list of related documents is given here as well.

**CPU core functions**

The functions of the CPU core (e.g. instruction set, processor protection functions, etc.) are not subject to this manual. Refer to the separate CPU core manual, shown in the section “Related Documents” in the chapter “Introduction”.

**CPU Subsystem functions**

The functions of the CPU Subsystem (including address map, operation modes, etc.) are described in the chapter “CPU System Function”.
The section “Write protected Registers” in this chapter describes how to deal with registers, that feature special write protection facilities.
If the microcontroller has separate bus systems beside the CPU Subsystem to connect certain functional modules, refer to the chapter “Bus Architecture”.

**Port functions**

The chapter “Port Functions” describes all input/output port related functions, such as port sharing, I/O buffer control, port filters.
The features and electrical properties of the I/O buffers are not subject to this manual, but are described in the Data Sheet.

**Interrupt functions**

Refer to the chapter “Interrupt Controller”.
Note that the function of each interrupt source is not described here, but in the related chapter of the module, that generates the interrupt.

**DMA/DTS functions**

Refer to the chapter “DMA/DTS Controller” or “DMA Controller”, if the target microcontroller does not feature DTS functions.
Note that the function of each DMA/DTS trigger source is not described here, but in the related chapter of the module, that generates the trigger signal.

**Flash memory**

For microcontrollers with on-chip flash memory refer to the chapter “Flash Memory” for information about the flash memories structure and features, programming facilities, etc.

**Stand-by functions**

How to set the microcontroller in stand-by modes and wake it up again is described in the chapter “Stand-by Controller (STBC)”.

**Code protection and security**

Facilities to protect program code in on-chip flash memory (if available) from illegal read-out via external flash programming equipment or debuggers is described in the chapter “Code Protection and Security”.

**Clock supply** The chapter “Clock Controller” describes the generation and operation of all clocks, provide to the entire microcontroller.

**Resets** The sources that can generate reset signals to all or dedicated internal modules and how to control them is described in the chapter “Reset Controller”.

**Functional modules** The description of most functional modules, like timers, serial interfaces, etc. is provided in separate chapters. These chapters have a certain structure of information presentation. Refer to the section *“Functional modules descriptions”*.

**Debugging** The main features on the On-Chip Debug Unit of the microcontroller is described in the chapter “On-chip Debug Unit (OCD)”.
Note that the description of the external debugger tool is not subject to this manual.

**Power supply** The chapter “Power Supply” provides information which modules of the microcontrollers are supplied by which external power supply pins.
Note that the specification of the external power supply is not subject to this manual. Refer to the Data Sheet for detailed definitions of the power supply.

**Boundary scan** If the target microcontroller supports boundary scan testing, refer to the chapter “Boundary Scan” for information about available Boundary Scan features.

## Notation of numbers and symbols

**Symbols** Symbols and notation are used as follows:

- Weight in data notation: Left is high order column, right is low order column
- Active low notation: $\overline{xxx}$ (pin or signal name is over-scored) or /xxx (slash before signal name)
- Memory map address: High order at high stage and low order at low stage

**Numeric notation**

- Binary: xxxx or $xxx_B$
- Decimal: xxxx
- Hexadecimal: $xxxx_H$ or 0x xxxx

**Numeric prefixes** represent different factors, depending on the measure:

| Prefix | Powers of 2 | Powers of 10 |
|---|---|---|
| k (kilo) | – | $10^3 = 1000$ |
| K (Kilo) | $2^{10} = 1024$ | – |
| M (Mega) | $2^{20} = 1024^2 = 1,048,576$ | $10^6 = 1000^2 = 1,000,000$ |
| G (Giga) | $2^{30} = 1024^3 = 1,073,741,824$ | $10^9 = 1000^3 = 1,000,000,000$ |
| m (milli) | – | $10^{-3} = 0.001$ |
| µ (micro) | – | $10^{-6} = 0.001^2 = 0.000,001$ |
| p (piko) | – | $10^{-9} = 0.001^3 = 0.000,000,001$ |
| | For example used for<br>• address and memory spaces in bytes: KB, MB, GB | For example used for<br>• frequencies: kHz, MHz, GHz<br>• times: ms, µs<br>• resistance: kΩ, MΩ<br>• capacitance: µF, pF |

**Register contents** X, x = don't care

## Diagrams

Block diagrams do not necessarily show the exact wiring in hardware but the functional structure.

Timing diagrams are for functional explanation purposes only, without any relevance to the real hardware implementation.

## Trademarks

All trademarks are the property of their respective owners.

# Functional modules descriptions

Most of the chapters provide a technical description of a certain functional module.

These chapters are split into two parts:

- The first section describes all properties of the functional module specific to the target product of the user manual, such as instances, register base addresses, input/output signal names, etc.
- The subsequent sections describe the features of the functional module as a generic description. The generic description is common to all user manuals of products, that feature this module.

## Functional modules abbreviation convention

Each functional module has a unique abbreviation, for instance

TAUA for the Timer Array Unit A

This shortcut is used in names for various purposes:

- The module registers and their bits names are preceded by this shortcut, for instance

  TAUAnTS for the TAUAn channel start trigger register

  The index “n” denotes the instance number of the module, refer to the next section and the key words *“Instances”* and *“Instances index n”*.
- The base address of the module registers include the by this shortcut, for instance

  <TAUAn_base> for the base address of the TAUAn registers
- The input/output signals of the module are preceded by this shortcut, for instance

  TAUAnTTIN0 for the TAUAn channel 0 input signal
- The names of the module interrupts includes the module shortcut, for instance

  INTTAUAnI0 for the TAUAn channel 0 interrupt

## Product specific features

The product specific section is always structured by a set of identical key words.

For the naming of signals product specific section serves also as some kind of interface between the generic module description and all other parts of the document.
This means that the names of signals, used in the generic module description, may be translated to other names, that are used in the other document chapters.
This name translation is given in form of tables, as the following as an example:

| Module signals | Function | Connected to |
|---|---|---|
| **Module shortcut:** | | |
| Name used in generic module description | Brief functional description | Name used in remaining document |

The following lists the key words for product specific definitions. As examples, definitions of different modules are used.

**Cautions**

1. The following product specific definitions are only used as examples and do not define any properties of the target product of this document.
2. Consequently the functional modules, used for examples purposes, may not be available with the target product of this document.

**Instances** The devices of the target product may contain different numbers of the functional module, so called instances. The “Instances” paragraph specifies the number of instances for all devices of the target product.

**Table 0-1 Example: Instances of TAUAn**

| Timer Array Unit A | Device_1 | Device_2 | Device_3 |
|---|---|---|---|
| Instance | 2 | 4 | 2 |
| Name | TAUA0 to TAUA1 | TAUA0 to TAUA3 | TAUA0 to TAUA1 |

**Instances index n**

Throughout the following generic module description, an instance of a module is identified by the index "n", for instance

TAUAnTS for the TAUAn channel start trigger register

“n” counts from 0 to the number of instances minus one.

**Other indices**

In case other indices, except “n” for instances, are used throughout the generic module description, they are specified here.

**Channel index m**

Timer Array Unit A has 16 channels. Throughout this chapter, the individual channels are identified by the index “m” (m = 0 to 15), thus a certain channel is denoted as CHm.
The even numbered channels (m = 0, 2, 4, 6, 8, 10, 12, 14) are denoted as CHm_even.
The odd numbered channels (m = 1, 3, 5, 7 , 9, 11, 13, 15) are denoted as CHm_odd.

**Register addresses**

All module register addresses in the generic description are given as address offsets to a base address, that is individual to a certain module instance n. For each module instance n the individual base address is given here.

**Table 0-2 Example: Register base addresses <TAUAn_base>**

| TAUAn instance | <TAUAn_base> address |
|---|---|
| TAUA0 | FF80 $8000_H$ |
| TAUA1 | FF80 $9000_H$ |
| TAUA2 | FF80 $A000_H$ |
| TAUA3 | FF80 $B000_H$ |
| TAUA4 | FF80 $C000_H$ |

**Clock supply** The clock signals of each instance n of the module and their connection to other functional modules of the device are given here.
A figure shows the modules clock supply options.

**Table 0-3 Example: TAUAn clock supply**

| TAUAn instance | TAUAn clock | Connected to |
|---|---|---|
| TAUA0 | PCLK | Clock Generator CKSCLK_006 |
| TAUA1 | PCLK | Clock Generator CKSCLK_104 |
| TAUA2 | PCLK | Clock Generator CKSCLK_111 |
| TAUA3 | PCLK | Clock Generator CKSCLK_106 |
| TAUA4 | PCLK | Clock Generator CKSCLK_105 |

**Interrupts and DMA/DTS** The interrupt signals of each instance n of the module and their connections to other functional modules of the device are given here.

**Table 0-4 Example: TAUAn interrupt and DMA/DTS requests**

| TAUAn signals | Function | Connected to |
|---|---|---|
| **TAUA0:** | | |
| INTTAUA0I0 to INTTAUA0I7 | Channel 0 to 7 interrupt | Interrupt Controller INTTAUA0I0 to INTTAUA0I7 |
| INTTAUA0I8 | Channel 8 interrupt | Interrupt Controller INTTAUA0I8<br>DMA Controller trigger 15<br>DTS Controller trigger 9 |
| INTTAUA0I9 to 15 | Channels 9 to 14 interrupt | not connected |
| **TAUA1:** | | |
| ... | ... | ... |

**Internal signals** Signals of some modules are connected to other device's modules. Such connections are defined here.

**Table 0-5 Example: VIn internal signals**

| VIn signal | Function | Connected to |
|---|---|---|
| VI0EN | VIO0 enable | VI0 I/F control |
| ES_VSYNC | VSYNC signal output | MVO0 EVSYNC |
| ES_HSYNC | HSYNC signal output | MVO0 EHSYNC |

**H/W reset** The signals, that reset each instance n of the module, are listed here.

**Table 0-6 Example: TAUAn reset signals**

| TAUAn | Reset signal |
|---|---|
| TAUA0 | • Reset Controller SYSRES<br>• Reset upon Isolated-Area-0 wake-up from DEEPSTOP mode |
| TAUA1 to TAUA4 | • Reset Controller SYSRES<br>• Reset upon Isolated-Area-1 wake-up from DEEPSTOP mode |

**I/O signals** The input/output signals of each instance n of the module and their connections to ports and other functional modules of the device are given here.

**Table 0-7 Example: TAUAn I/O signals**

| TAUA signal | Function | Connected to |
|---|---|---|
| **TAUA0:** | | |
| TAUA0TTIN0 | Channel 0 input | Port TAUA0I0[a] or FCN0 TSOUT or port URTE0RX[b] |
| TAUA0TTIN1 | Channel 1 input | Port TAUA0I1[a] or FCN1 TSOUT or port URTE1RX[b] |
| TAUA0TTIN2 to TAUA0TTIN15 | Channel 2 to 15 input | Port TAUA0I2[a] to TAUA0I115[a] |
| TAUA0TTOUT0 to TAUA0TTOUT15 | Channel 0 to 15 output | Port TAUA0O0 to TAUA0O15 |
| **TAUA1:** | | |
| ... | ... | ... |

a) These input signals are passed through a noise filter, refer to the section *"Port Filters"* in the chapter *"Port Functions"*.

b) Refer to *15.2 "TAUA Input Selection" on page 662* for details.

**Special definitions** If the functional module needs any particular definitions for its operation, which are product dependent, these are defined here.

## Further information

For further information see *http://www.renesas.com*.

# Chapter 1 Introduction

## 1.1 V850E2/Fx4-G Product Line Overview

### (1) V850E2/FF4-G product line overview

Table 1-1 V850E2/FF4-G product series overview (1/2)

| Series name: | | | FF4-G-256K | FF4-G-512K |
|---|---|---|---|---|
| Part number: | | | μPD70F4177 | μPD70F4178 |
| Internal memory | Instruction flash | | 256 KB | 512 KB |
| | Data flash | | 32 KB | |
| | CPU RAM | | 32 KB | 64 KB |
| | Back-up RAM | | 4 KB | |
| CPU | CPU core | | V850E2 | |
| | CPU frequency | | 64 MHz max.[a] | |
| | System Protection Functions (SPF) | MPU | provided | |
| | | SRP | provided | |
| DMA | | | 8 channels | |
| Operating clock | Main Oscillator (MainOsc) | | 4, 5, 6, 10, 12, 16, 20 MHz | |
| | Low Speed Internal Oscillator (LS IntOsc) | | 240 kHz typ. | |
| | High Speed Internal Oscillator (HS IntOsc) | | 8 MHz typ. | |
| | PLL0 | | 64 MHz max. | |
| I/O ports | | | 59 | |
| A/D converter (ADAA0) | | | 1 × 10 channels, 10 bit | |
| Timers | Timer Array Unit B (TAUB), 16 bit | | 1 unit × 16 channels | |
| | Timer Array Unit J (TAUJ), 32 bit | | 1 unit × 4 channels | |
| | Window Watchdog (WDTA) | | 2 channels | |
| | Operating System Timer (OSTM) | | 1 channel | |
| Serial interfaces | CAN I/F (FCN) | | 3 channel (32 messages buffer) | 6 channel (32 messages buffer) |
| | UART I/F (URTE) with LIN Master Controller (LM) | | 2 channels | |
| | Synchronous I/F (CSIG) | | 2 channels | |
| | $I^2C$ I/F (IICB) | | 1 channel | |
| External interrupts | Maskable (INTPn) | External | 9 | |
| | | Internal | 61 | 70 |
| | Non-maskable (NMI) | External | 1 | |
| | | Internal | 2 | |
| Other functions | Power-On-Clear (POC) | | provided | |
| | Clock Monitors (CLMA) | | provided for MainOsc, HS IntOsc, PLL0 supervision | |
| | Key Return (KR) | | 8 channels | |
| | On-Chip debug (OCD) | | provided | |

**Table 1-1 V850E2/FF4-G product series overview (2/2)**

| Series name: | | FF4-G-256K | FF4-G-512K |
|---|---|---|---|
| **Part number:** | | **µPD70F4177** | **µPD70F4178** |
| Voltage supply | Internal supply | $V_{POC}$ to 5.5 V[a] | |
| | I/O supply | $V_{POC}$ to 5.5 V[a] | |
| Operating Temperature | | -40° C to +125° C[a] | |
| Package | | 80-pin LQFP | |

a) Refer to the Data Sheet.

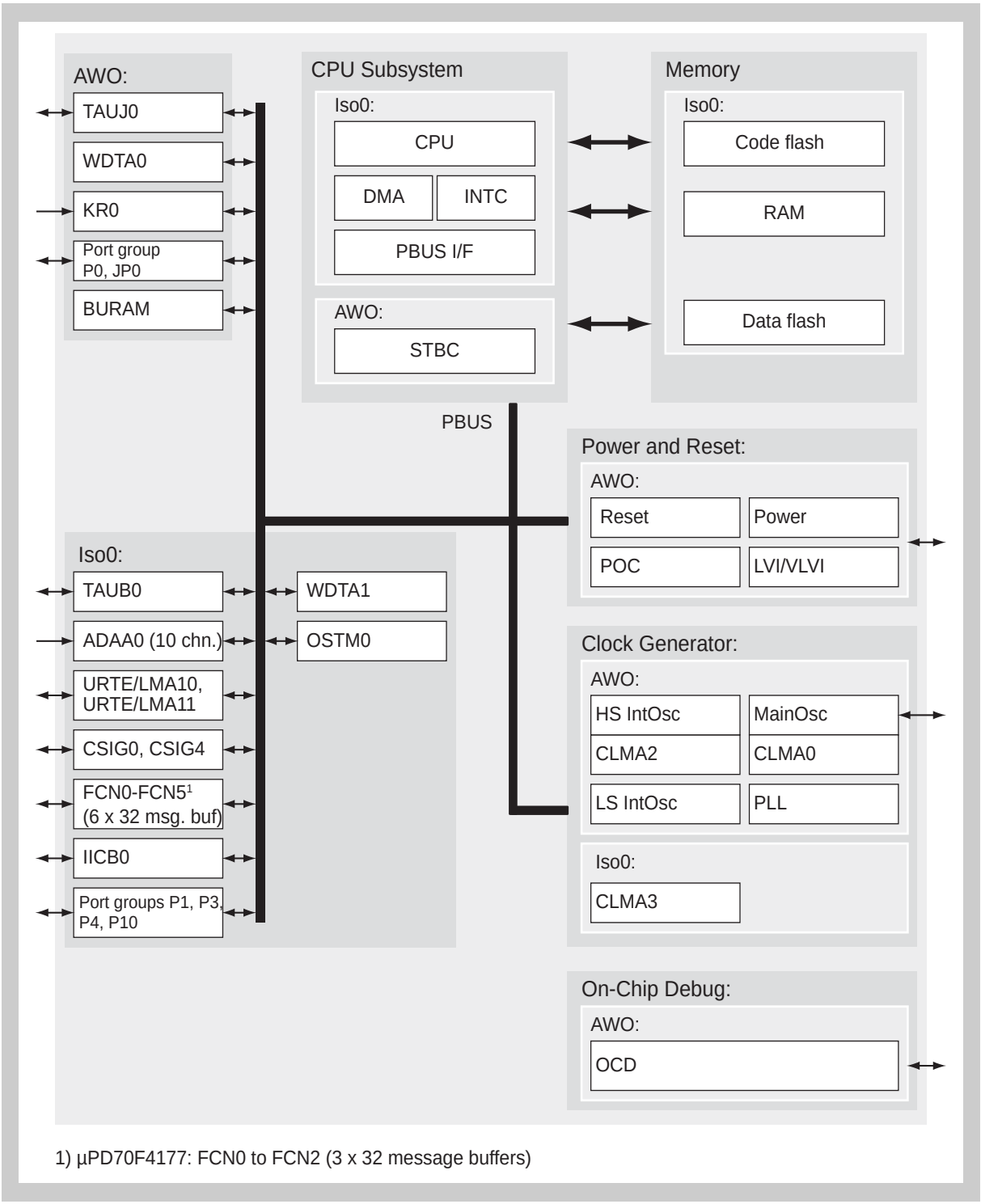



**Figure 1-1 V850E2/FF4-G block diagram**

### (2) V850E2/FG4-G product line overview

**Table 1-2 V850E2/FG4-G product series overview**

| Series name: | | | FG4-G-256K | FG4-G-512K |
|---|---|---|---|---|
| Part number: | | | μPD70F4179 | μPD70F4180 |
| Internal memory | Instruction flash | | 256 KB | 512 KB |
| | Data flash | | 32 KB | |
| | CPU RAM | | 32 KB | 64 KB |
| | Back-up RAM | | 4 KB | |
| CPU | CPU core | | V850E2 | |
| | CPU frequency | | 64 MHz max.[a] | |
| | System Protection Functions (SPF) | MPU | provided | |
| | | SRP | provided | |
| DMA | | | 8 channels | |
| Operating clock | Main Oscillator (MainOsc) | | 4, 5, 6, 10, 12, 16, 20 MHz | |
| | Low Speed Internal Oscillator (LS IntOsc) | | 240 kHz typ. | |
| | High Speed Internal Oscillator (HS IntOsc) | | 8 MHz typ. | |
| | PLL0 | | 64 MHz max. | |
| I/O ports | | | 77 | |
| A/D converter (ADAA0) | | | 1 × 16 channels, 10 bit | |
| Timers | Timer Array Unit B (TAUB), 16 bit | | 1 unit × 16 channels | |
| | Timer Array Unit J (TAUJ), 32 bit | | 1 unit × 4 channels | |
| | Window Watchdog (WDTA) | | 2 channels | |
| | Operating System Timer (OSTM) | | 1 channel | |
| Serial interfaces | CAN I/F (FCN) | | 3 channel (32 messages buffer) | 6 channel (32 messages buffer) |
| | UART I/F (URTE) with LIN Master Controller (LM) | | 3 channels | |
| | Synchronous I/F (CSIG) | | 3 channels | |
| | $I^2C$ I/F (IICB) | | 1 channel | |
| External interrupts | Maskable (INTPn) | External | 13 | |
| | | Internal | 67 | 76 |
| | Non-maskable (NMI) | External | 1 | |
| | | Internal | 2 | |
| Other functions | Power-On-Clear (POC) | | provided | |
| | Clock Monitors (CLMA) | | provided for MainOsc, HS IntOsc, PLL0 supervision | |
| | Key Return (KR) | | 8 channels | |
| | On-Chip debug (OCD) | | provided | |
| Voltage supply | Internal supply | | $V_{POC}$ to 5.5 V[a] | |
| | I/O supply | | $V_{POC}$ to 5.5 V[a] | |
| Operating Temperature | | | -40° C to +125° C[a] | |
| Package | | | 100-pin LQFP | |

a) Refer to the Data Sheet.

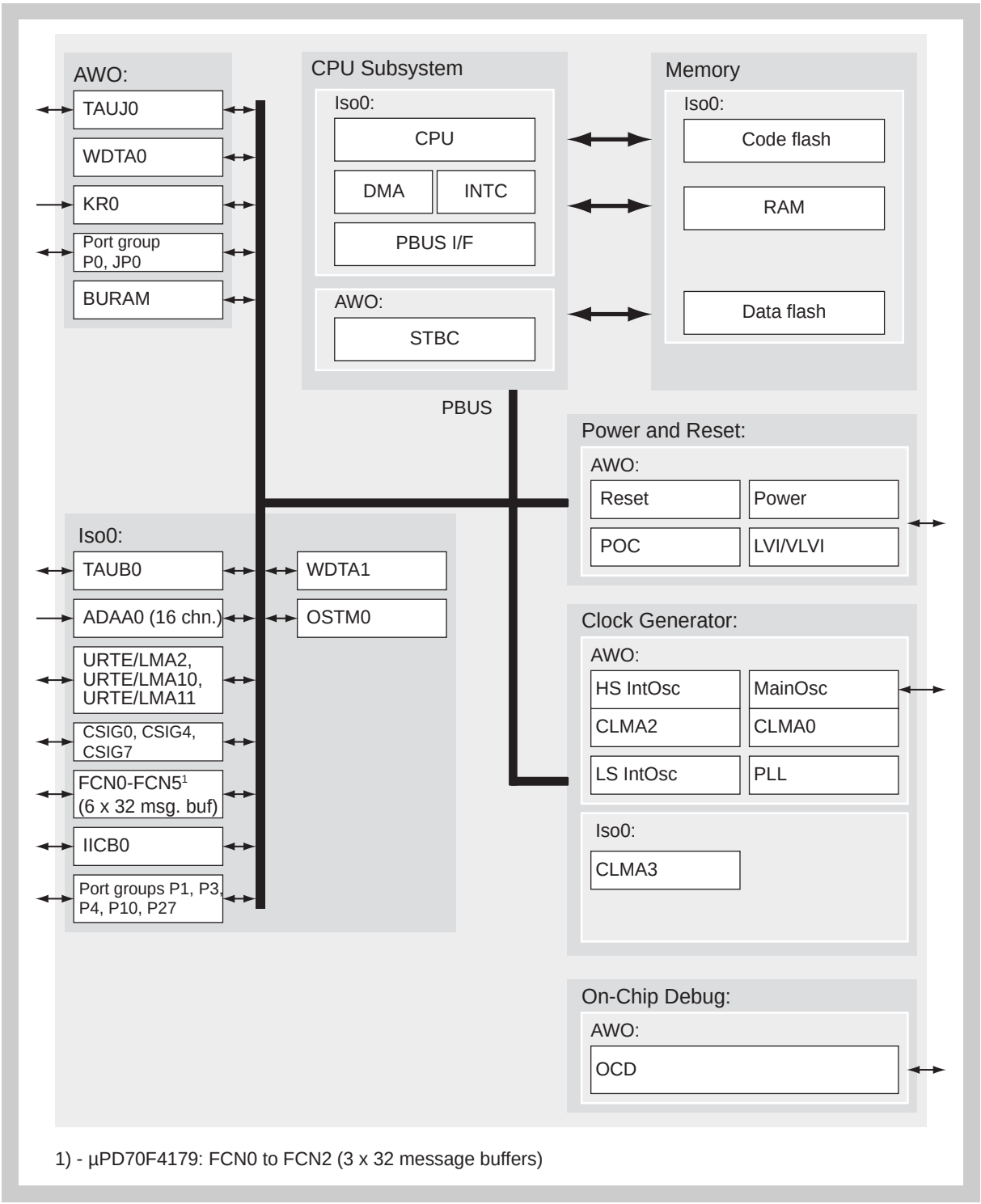



**Figure 1-2 V850E2/FG4-G block diagram**

## 1.2 Related Documents

**Table 1-3 Related documents**

| Document number[a] | Title |
|---|---|
| R01US0037EJxxxx | User's Manual:<br>V850E2 32-bit Microcontroller Core Architecture |
| R01DS0167EJxxxx | FF4-G Data Sheet |
| R01DS0166EJxxxx | FG4-G Data Sheet |
| R20UT0008EJxxxx | User's Manual:<br>PG-FP5 Flash Memory Programmer |
| U17638EJxVxUM00 | User's Manual:<br>QB-V850MINI, QB-V850MINIL On-Chip Debug Emulator |
| R01US0046EDxxxx | User's Manual:<br>Flash Self-Programming Library FSL - T06 |
| <tbd> | User's Manual:<br>Data Flash Access Library FDL - T06 |

a) "x" denotes the current document revision numbers.

## 1.3 Ordering Information

Table 1-4 V850E2/Fx4-G ordering information

| Series name | Device name | Renesas order code | Remarks |
|---|---|---|---|
| FF4-G-256K | µPD70F4177 | uPD70F4177GK(Ax)-GAK-AX | |
| FF4-G-512K | µPD70F4178 | uPD70F4178GK(Ax)-GAK-AX | |
| FG4-G-256K | µPD70F4179 | uPD70F4179GC(Ax)-UEU-AX | |
| FG4-G-512K | µPD70F4180 | uPD70F4180GC(Ax)-UEU-AX | |

# Chapter 2 Port Functions

This chapter contains a generic description of the Port control functions.

The first section describes all properties specific to the V850E2/Fx4-G, such as port groups, register base addresses, etc.

The second section describes the features of the port control functions that apply to all ports.

The third section summarizes the individual functions of all pins of V850E2/Fx4-G microcontrollers.

Finally the function of analog and digital filters, which are implemented at some pins, are described.

## 2.1 V850E2/Fx4-G Port Features

**Port groups**

The V850E2/Fx4-G microcontrollers have following number of port groups:

**Table 2-1 Port groups of V850E2/Fx4-G**

| Port groups | V850E2/FF4-G | V850E2/FG4-G |
|---|---|---|
| Number | 6 | 7 |
| Names | P0, P1, P3, P4, P10, JP0 | P0, P1, P3, P4, P10, P27, JP0 |

**Port groups index n**

Throughout this chapter, the individual port groups are identified by the index “n”, for example, PMCn for the port mode control register of Pn.

**Register addresses**

All port and JTAG port control register addresses are given as address offsets from the individual base addresses <PORTn_base> and <JPORT0_base>. The base addresses <PORTn_base> and <JPORT0_base> are specified in the following table:

**Table 2-2 Port base addresses <PORTn_base> and <JPORT0_base>**

| <PORTn_base> address | <JPORT0_base> address |
|---|---|
| FF40 $0000_H$ | FF44 $0000_H$ |

## 2.2 Overview

The microcontroller has various pins for input/output functions, known as ports. The ports are organized in port groups.

The microcontroller also has several control registers to allocate other than general purpose input/output functions to the pins.

For a description of the terms pin, port, or port group, see the following section *“Terms”*.

**Features summary**

- Configuration possible for individual pins.
- The following features can be selected for most of the pins:
  - One out of four input buffer characteristics
  - Output current limit
  - Open drain emulation
  - Pull-up or pull-down resistor connection
- The following registers are offered for most of the ports:
  - Direct register for reading the pin values
  - Port register
  - Port set/reset register
  - Register for output inversion

### 2.2.1 Terms

In this chapter, the following terms are used:

- **Pin**

  Denotes the physical pin. Every pin is denoted by a unique pin number. The pin numbers depend on the package and are given in the Data Sheet.

  Most of the pins can be used in several modes. Thus the pin name depends on the selected mode.

- **Port group**

  Denotes a group of ports. The ports of a port group have a common set of port mode control registers.

- **Port mode / Port**

  A pin in port mode works as a general purpose input/output pin. It is then called “port”.

  The corresponding name is Pn_m. For example, P0_7 denotes port 7 of port group 0. It is referenced as “port P0_7”.

- **Alternative mode**

  In alternative mode, a pin can be used for various non-general purpose input/output functions, for example as the input/output pin of on-chip peripherals.

  The corresponding pin name depends on the selected function. For example, pin INTP0 denotes the pin for one of the external interrupt inputs.

  Note that two different names can refer to the same physical pin, for example P0_0 and INTP0. The different names indicate the function in which the pin is being operated.

**JTAG ports** The JTAG port group JP0 is used for connecting the debugger for on-chip debugging purposes. Therefore it present a special port group, as the ports of JP0 are not available for application purposes during a debug session. During normal operation, i.e. without debugger, the JP0 ports can be used in the same way as all others.

The JTAG port group JP0 control registers and their control bits have the same names as the other port groups, registers and bits, but are identified by a “J” prefix.

**Note** Throughout this chapter the description of all ports and their registers apply also to the JTAG ports, unless otherwise noted.

### 2.2.2 Pin function configuration

The pins can operate in three different general modes:

- Port mode (PMCn.PMCn_m = 0)
  In port mode the pin operates as a general purpose I/O port. PMn.PMn_m selects input or output.
- S/W I/O control alternative mode (PMCn.PMCn_m = 1, PIPCn.PIPCn_m = 0)
  In S/W I/O control alternative mode the pin is operated by an alternative function. The selection between input or output is done by S/W via the PMn.PMn_m control bits.
- Direct I/O control alternative mode (PMCn.PMCn_m = 1, PIPCn.PIPCn_m = 1)
  In direct I/O control alternative mode the pin is operated by an alternative function. In contrast to the S/W I/O control alternative mode the input/output control is also handled by the alternative function, thus the S/W doesn't have to care about.

An overview of the register settings is given in the tables below.

**Table 2-3 Pin function configuration (overview)**

| Mode | Control bits | | | I/O |
|---|---|---|---|---|
| | PMCn_m | PMn_m | PIPCn_m | |
| Port | 0 | 0 | X | O |
| | | 1[a] | | I |
| S/W I/O control alternative | 1 | 0 | 0 | O |
| | | 1 | 0 | I |
| Direct I/O control alternative | | X | 1 | controlled by alternative function |

a) The input buffer must be enabled (PIBCn.PIBCn_m = 1)

If a pin is operated in an alternative mode (PMCn.PMCn_m = 1), one out of up to four different alternative functions can be selected by the PFCn and PFCEn registers.

Selection of one of the alternative input and output functions:

- S/W I/O control alternative functions (PIPCn.PIPCn_m = 0):
  - outputs (PMn_m = 0): ALT-OUT1 to ALT-OUT4
  - inputs (PMn_m = 1): ALT-IN1 to ALT-IN4
- Direct I/O control alternative functions (PIPCn.PIPCn_m = 1):
  - input/out of ALT-OUT1 to ALT-OUT4 and ALT-IN1 to ALT-IN4 is directly controlled by the alternative function

**Table 2-4 Alternative mode selection overview (PMCn.PMCn_m = 1)**

| Function | Registers | | | | I/O |
|---|---|---|---|---|---|
| | PIPC[a] | PM[a] | PFCE | PFC | |
| Alternative output mode 1 (ALT-OUT1) | 0 | 0 | 0 | 0 | O |
| Alternative input mode 1 (ALT-IN1) | | 1 | | | I |
| Alternative output mode 2 (ALT-OUT2) | | 0 | 0 | 1 | O |
| Alternative input mode 2 (ALT-IN2) | | 1 | | | I |
| Alternative output mode 3 (ALT-OUT3) | | 0 | 1 | 0 | O |
| Alternative input mode 3 (ALT-IN3) | | 1 | | | I |
| Alternative output mode 4 (ALT-OUT4) | | 0 | 1 | 1 | O |
| Alternative input mode 4 (ALT-IN4) | | 1 | | | I |

a) If PIPCn.PIPCn_m = 1, the I/O direction is directly controlled by the alternative function and PM is ignored.

**Caution** In case a certain alternative input function is available via multiple ports, only one port must be configured to use this alternative input function. All other ports must be configured to use other signals.

**PMn/PMCn register write**

The port mode register PMn and port mode control register PMCn can be manipulated in two different ways:

- Direct PMn/PMCn write
  New value can be written directly to the PMn/PMCn register.
- Indirect PMn/PMCn bit set/reset
  An indirect way to set or reset a PMn/PMCn bit is possible by using following registers:
  - Port mode set reset register PMSRn
    If the bit PMSRn.PMSRn(m+16) = 1, the value of bit PMSRn.PMSRn_m determines the value of PMn.PMn_m.
    Thus PMn_m can be set/reset without a direct write to PMn.
  - Port mode control set reset register PMCSRn
    If the bit PMCSRn.PMCSRn(m+16) = 1, the value of bit PMCSRn.PMCSRn_m determines the value of PMCn.PMn_m.
    Thus PMn_m can be set/reset without a direct write to PMCn.

  The indirect PMn/PMCn set/reset operation provides access to single bits of the PMn/PMCn register, while leaving all other register bits untouched.

Both ways to manipulate a PMn/PMCn bit can be used concurrently.

**Note** It is recommended to use the indirect PMn/PMCn bit set/reset method for changing a single bit or concurrently several bits of the PMn/PMCn register, since all other bits are not modified and can be independently treated by other S/W modules, for instance in interrupt service routines.

### 2.2.3 Pin data input/output

In the following the registers are described, used for data input and output.

Depending on the pin mode, the source of the data to be output and the data read via the PPRn register differs.

**Output data**

In *port mode* (PMCn.PMCn_m = 0) the data of Pn.Pn_m is output to pin Pn_m.

In *alternative mode* (PMCn.PMCn_m = 1) the pin Pn_m output is determined by the alternative function.

**Input data**

A read operation of the PPRn register returns either the value of the Pn_m pin, the associated bit of the port register Pn.Pn_m or the data output by an alternative function.

The source of the data read via PPRn depends on the pin mode and the setting of several control bits.

The table below summarizes the different PPRn read modes.

**Note**

PBDCn_m is not included in the table, as it can be set to 1 for reading the Pn_m pin level in all modes.

**Table 2-5 PPRn_m read values**

| PMCn_m | PMn_m | PIBCn_m | PIPCn_m | PODCn_m | Mode | PPRn_m read value |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | X | X | Port input, input buffer disabled | Pn.Pn_m register |
| | | 1 | | X | Port input, input buffer enabled | Pn_m pin |
| | 0 | X | | 0 | Port push-pull output | Pn.Pn_m register[a] |
| | | | | 1 | Port open-drain output | |
| 1 | 1 | X | 0 | X | S/W I/O control alternative input | Pn_m pin |
| | 0 | | | 0 | S/W I/O control alternative push-pull output | Alternative function output[a] |
| | | | | 1 | S/W I/O control alternative open-drain output | |
| | X | | 1 | 0 | Direct I/O control alternative input/ push-pull output | If alternative functions sets port in<br>• input: PPRn_m = Pn_m pin<br>• output: PPRn_m = alternative function output[a] |
| | | | | 1 | Direct I/O control alternative input/ open-drain output | |

a) If PBDCn_m = 1, Pn_m pin level is read via PPRn_m.

The control registers in the table above have following effects:

- PBDCn.PBDCn_m (see table footnote)
  This bit forces to read the Pn_m pin level via PPRn_m, thus enabling a bi-directional mode, where the level of pin Pn_m can also be read back if the port is operated in an output mode.
- PMCn.PMCn_m
  This bit selects port mode (PMCn_m = 0) or alternative mode (PMCn_m = 1).

- PMn.PMn_m
  This bit selects input (PMn_m = 1) or output (PMn_m = 0) in port mode (PMCn_m = 0) and S/W I/O control alternative function mode (PMCn_m = 1, PIPCn_m = 0).
- PIBCn.PIBCn_m
  This bit disables (PIBCn_m = 0) or enables (PIBCn_m = 1) the input buffer in input port mode (PMCn_m = 0 and PMn_m = 1). If the input buffer is disabled, PPRn_m reads the Pn.Pn_m bit, otherwise the Pn_m pin level is returned.
- PIPCn.PIPCn_m
  This bit selects between the S/W and direct I/O control alternative mode.
- PODCn.PODCn_m
  This bit selects between push-pull (PODCn_m = 0) and open-drain (PODCn_m = 1) output.

**Pn register write**

The data to be output via port Pn_m in port mode (PMCn.PMCn_m = 0) is held in the port register Pn. The Pn data can be manipulated in two different ways:

- Direct Pn write
  New data can be written directly to the Pn register.
- Indirect Pn bit set/reset/not
  An indirect way to set (Pn_m = 1), reset (Pn_m = 0), or invert ($\overline{Pn\_m} \rightarrow Pn\_m$) a Pn bit is possible using two registers:
  - Port set reset register PSRn
    If the bit PSRn.PSRn(m+16) = 1, the value of bit PSRn.PSRn_m determines the value of Pn.Pn_m.
    Thus Pn_m can be set/reset without a direct write to Pn.
  - Port NOT register PNOTn
    Setting PNOTn.PNOTn_m = 1 inverts the bit Pn.Pn_m without a direct write to Pn_m.

  The indirect Pn set/reset/not operation provides access to single bits of the Pn register, while leaving all other Pn bits untouched.

Both ways to manipulate a Pn bit can be used concurrently.

**Note** It is recommended to use the indirect Pn bit set/reset/not method for changing a single bit or concurrently several bits of the Pn register, since all other bits are not modified and can independently be treated by other S/W modules, for instance in interrupt service routines.

**Caution** If a port Pn_m

- provides an alternative output ALT_OUTx and input function ALT_INx
- and is used in alternative output mode ALT_OUTx (PMCn.PMCn_m = 1, PMn.PMn_m = 0)
- and the bi-directional mode is enabled (PBDCn.PBDCn_m = 1) for reading the Pn_m level via PPRn.PPRn_m,

the Pn_m output, i.e. of ALT_OUTx, is internally fed back to the alternative input function ALT_INx.

### 2.2.4 Port control logic diagram

The following diagram shows the logical circuitry of the port control functions.

**Note** The diagram is only a logical reference and does not show the real circuitry.

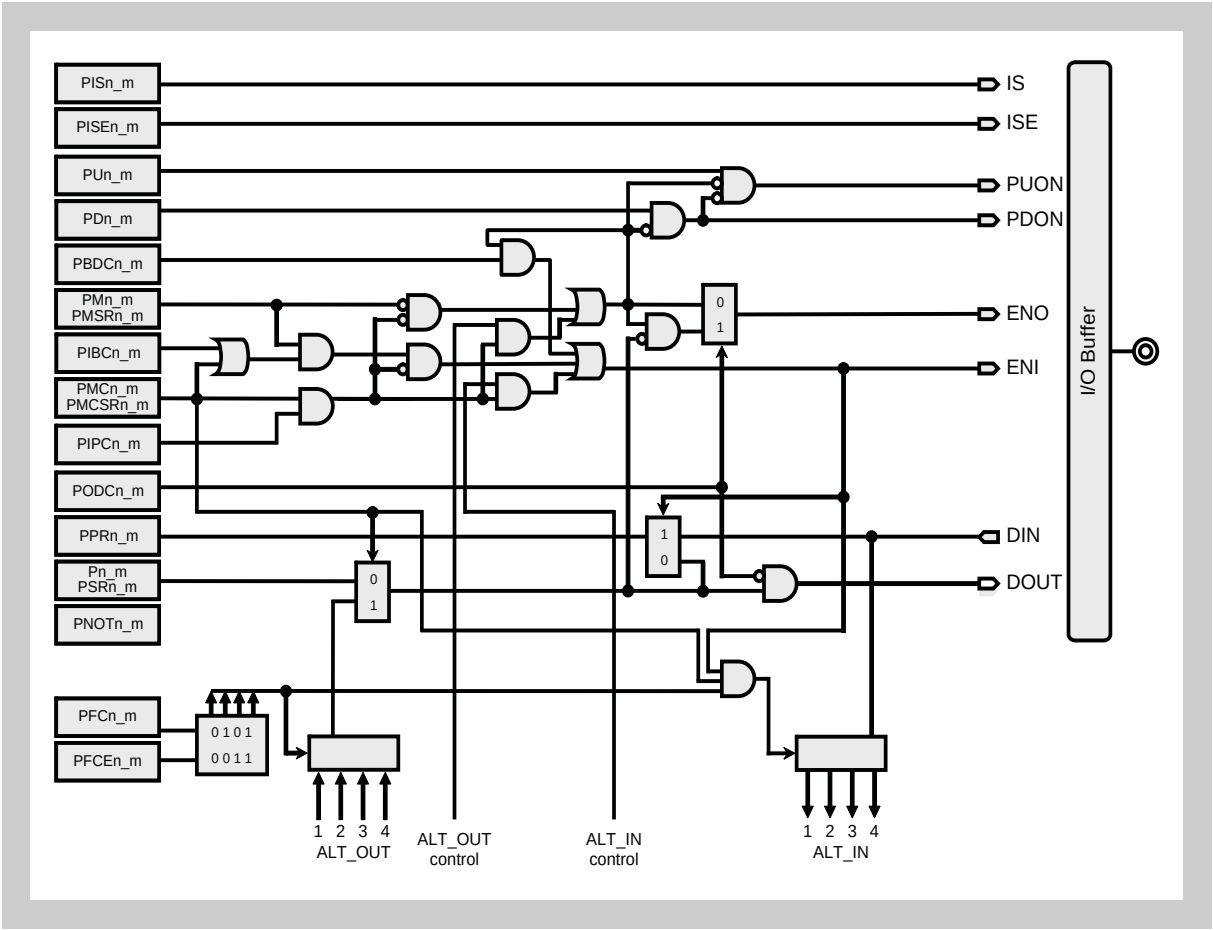



**Figure 2-1 Port control logic diagram**

The signals to the I/O buffer in the diagram above have the following general function:

| Buffer control signal | General function |
|---|---|
| IS, ISE | input buffer selection |
| PUON/PDON | pull-up/-down register control |
| ENO/ENI | output/input buffer enable |
| DIN/DOUT | port data in/out |

## 2.3 Port Group Configuration Registers

This section starts with an overview of all configuration registers and then presents all registers in detail. The configuration registers are grouped as follows:

- *"Pin function configuration registers"*
- *"Pin data input/output"*
- *"Configuration of electrical characteristics registers"*

### 2.3.1 Writing to protected registers

Write protected registers are protected from inadvertent write access due to erroneous program execution, etc.

Following port registers feature this special write protection:

- Port open drain control registers PODCn, JPODC0

Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

### 2.3.2 Port control registers overview

The following registers are used for the configuration of the individual pins of the port groups:

**Note** Some of the registers, listed in the table below, are not available for all port groups n. Refer to the section *"V850E2/Fx4-G port functions"* below for information which registers are available for the individual port groups.

**Table 2-6 Registers for port group configuration (1/2)**

| Register name | Shortcut | Address |
|---|---|---|
| **Port function configuration:** | | |
| Port mode control register | PMCn | <PORTn_base> + $0400_H$ + n x 4 |
| | JPMC0 | <JPORT0_base> + $0040_H$ |
| Port mode control set reset register | PMCSRn | <PORTn_base> + $0900_H$ + n x 4 |
| | JPMCSR0 | <JPORT0_base> + $0090_H$ |
| Port IP control register | PIPCn | <PORTn_base> + $4200_H$ + n x 4 |
| Port mode register | PMn | <PORTn_base> + $0300_H$ + n x 4 |
| | JPM0 | <JPORT0_base> + $0030_H$ |
| Port mode set reset register | PMSRn | <PORTn_base> + $0800_H$ + n x 4 |
| | JPMSR0 | <JPORT0_base> + $0080_H$ |
| Port input buffer control register | PIBCn | <PORTn_base> + $4000_H$ + n x 4 |
| | JPIBC0 | <JPORT0_base> + $0400_H$ |
| Port function control register | PFCn | <PORTn_base> + $0500_H$ + n x 4 |
| | JPFC0 | <JPORT0_base> + $0050_H$ |
| Port function control expansion register | PFCEn | <PORTn_base> + $0600_H$ + n x 4 |

**Table 2-6 Registers for port group configuration (2/2)**

| Register name | Shortcut | Address |
|---|---|---|
| **Data input/output:** | | |
| Port bi-direction control register | PBDCn | <PORTn_base> + $4100_H$ + n x 4 |
| | JPBDC0 | <JPORT0_base> + $0410_H$ |
| Port pin read register | PPRn | <PORTn_base> + $0200_H$ + n x 4 |
| | JPPR0 | <JPORT0_base> + $0020_H$ |
| Port register | Pn | <PORTn_base> + $0000_H$ + n x 4 |
| | JP0 | <JPORT0_base> + $0000_H$ |
| Port NOT register | PNOT0 | <PORTn_base> + $0700_H$ + n x 4 |
| | JPNOT0 | <JPORT0_base> + $0070_H$ |
| Port set reset register | PSRn | <PORTn_base> + $0100_H$ + n x 4 |
| | JPSR0 | <JPORT0_base> + $0010_H$ |
| **Configuration of electrical characteristics:** | | |
| Pull-up option register | PUn | <PORTn_base> + $4300_H$ + n x 4 |
| | JPU0 | <JPORT0_base> + $0430_H$ |
| Pull-down option register | PDn | <PORTn_base> + $4400_H$ + n x 4 |
| | JPD0 | <JPORT0_base> + $0440_H$ |
| Port open drain control register | PODCn | <PORTn_base> + $4500_H$ + n x 4 |
| | JPODC0 | <JPORT0_base> + $0450_H$ |
| Port input buffer selection register | PISn | <PORTn_base> + $4700_H$ + n x 4 |
| | JPIS0 | <JPORT0_base> + $0470_H$ |
| Port input buffer selection expansion register | PISEn | <PORTn_base> + $4800_H$ + n x 4 |
| | JPISE0 | <JPORT0_base> + $0480_H$ |

**<PORTn_base>** The base address <PORTn_base> of the port contorl registers is defined in the first section of this chapter under the key word “Register addresses”.

**JTAG port registers** The following register descriptions do not explicitely reference the JTAG port registers. However all description apply also to the respective JTAG port registers, but the base address of the JTAG port registers is different:

**<JPORT0_base>** The base addresses <JPORT0_base> of the JTAG port control registers is defined in the first section of this chapter under the key word “Register addresses”.

**Initial register values** The initial values after reset release depend on the port, and are not described in the following register descriptions, but are given in the section *“V850E2/Fx4-G Port Groups Configuration”*.

### 2.3.3 Port function configuration registers

#### (1) PMCn/JPMC0 - Port mode control register

This register specifies whether the individual pins of port group n are in port mode or in alternative mode.

**Access** PMCn: This register can be read/written in 16-bit units.
JPMC0: This register can be read/written in 8-bit units.

**Address** PMCn: <PORTn_base> + $0400_H$ + n x 4
JPMC0: <JPORT0_base> + $0040_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PMCn_15 | PMCn_14 | PMCn_13 | PMCn_12 | PMCn_11 | PMCn_10 | PMCn_9 | PMCn_8 | PMCn_7 | PMCn_6 | PMCn_5 | PMCn_4 | PMCn_3 | PMCn_2 | PMCn_1 | PMCn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPMC0 are named JPMC0_[7:0].

**Table 2-7 PMCn/JPMC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PMCn_[15:0] | Specifies the operation mode of the corresponding pin:<br>0: Port mode<br>1: Alternative mode |

**Cautions**

1. Setting PMCn.PMCn_m = 1 to use a port in alternative mode does not hand over I/O control to the alternative function. If the alternative function requires direct I/O control, PIPCn.PIPCn_m must also be set to 1.
2. Setting PMCn.PMCn_m = 1 to use a port in alternative mode may also require to configure a port filter, if this port is used as a signal input. The input signal may be passed through a noise filter, that may need to be configured. Refer to the section *"Port Filters"* in this chapter.

### (2) PMCSRn/JPMCSR0 - Port mode control set reset register

This register provides an alternative method to write data to the PMCn register.

The register's upper 16 bit PMCSRn_[31:16] specify which PMCn.PMCn_m bit will be modified by the corresponding bit of the lower 16 bit PMCSRn_[15:0].

**Access** These registers can be read/written in 32-bit units.
Bits 31 to 16 are always read as $0000_H$.
Reading bits 15 to 0 returns the value of register PMCn/JPMC0.

**Address** PMCSRn: <PORTn_base> + $0900_H$ + n x 4
JPMCSR0: <JPORT0_base> + $0090_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PMCSRn_31 | PMCSRn_30 | PMCSRn_29 | PMCSRn_28 | PMCSRn_27 | PMCSRn_26 | PMCSRn_25 | PMCSRn_24 | PMCSRn_23 | PMCSRn_22 | PMCSRn_21 | PMCSRn_20 | PMCSRn_19 | PMCSRn_18 | PMCSRn_17 | PMCSRn_16 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PMCSRn_15 | PMCSRn_14 | PMCSRn_13 | PMCSRn_12 | PMCSRn_11 | PMCSRn_10 | PMCSRn_9 | PMCSRn_8 | PMCSRn_7 | PMCSRn_6 | PMCSRn_5 | PMCSRn_4 | PMCSRn_3 | PMCSRn_2 | PMCSRn_1 | PMCSRn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPMCSR0 are named JPMCSR0_[31:0].

**Table 2-8 PMCSRn/JPMCSR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 16 | PMCSRn_[31:16] | PMCSRn_m specifies whether the value of the corresponding lower bit PMCSRn_m value is written to PMCn_m:<br>0: PMCn_m is independent of PMCSRn_m<br>1: PMCn_m is PMCSRn_m<br>Example:<br>If PMCSRn.PMCSRn_31 = 1, the value of bit PMCSRn.PMCSRn_15 is written to bit PMCn.PMCn_15 and output. |
| 15 to 0 | PMCSRn_[15:0] | Specifies the PMCn_m value if the corresponding upper bit PMCSRn_(m+16) is 1:<br>0: PMCn_m = 0<br>1: PMCn_m = 1 |

### (3) PIPCn - Port IP control register

This register specifies whether the I/O direction of pin Pn_m is controlled by the port mode register PMn.PMn_m or by an alternative function.

If pin Pn_m is operated in alternative mode (PMCn.PMCn_m = 1) and the alternative function requires to directly control the I/O direction of Pn_m, PIPCn.PIPCn_m must be set to 1 as well. This hands over I/O control to the alternative function and overrules the PMn.PMn_m setting.

**Access** PIPCn: This register can be read/written in 16-bit units.

**Address** PIPCn: <PORTn_base> + $4200_H$ + n x 4

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PIPCn_15 | PIPCn_14 | PIPCn_13 | PIPCn_12 | PIPCn_11 | PIPCn_10 | PIPCn_9 | PIPCn_8 | PIPCn_7 | PIPCn_6 | PIPCn_5 | PIPCn_4 | PIPCn_3 | PIPCn_2 | PIPCn_1 | PIPCn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 2-9 PIPCn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PIPCn_[15:0] | Specifies the I/O control mode:<br>0: I/O mode is selected by PMn.PMn_m (S/W I/O control)<br>1: I/O mode is selected by peripheral function (direct I/O control) |

### (4) PMn/JPM0 - Port mode register

The PMn register specifies whether the individual pins of the port group n are in input mode or in output mode.

**Access** PMn: This register can be read/written in 16-bit units.
JPM0: This register can be read/written in 8-bit units.

**Address** PMn: <PORTn_base> + $0300_H$ + n x 4
JPM0: <JPORT0_base> + $0030_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PMn_15 | PMn_14 | PMn_13 | PMn_12 | PMn_11 | PMn_10 | PMn_9 | PMn_8 | PMn_7 | PMn_6 | PMn_5 | PMn_4 | PMn_3 | PMn_2 | PMn_1 | PMn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPM0 are named JPM0_[7:0].

**Table 2-10 PMn/JPM0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PMn_[15:0] | Specifies input/output mode of the corresponding pin:<br>0: Output mode (output enabled)<br>1: Input mode (output disabled) |

**Notes**
1. To use a port in input port mode (PMCn.PMCn_m = 0 and PMn.PMn_m = 1), the input buffer must be enabled (PIBCn.PIBCn_m = 1).
2. By default, PMn_m specifies the I/O direction in port mode (PMCn.PMCn_m = 0) and alternative mode (PMCn.PMCn_m=1), since PIPCn.PIPCn_m = 0 after reset.

**(5) PMSRn/JPMSR0 - Port mode set reset register**

This register provides an alternative method to write data to the PMn register.

The register's upper 16 bit PMSRn_[31:16] specify which PMn.PMn_m bit will be modified by the corresponding bit of the lower 16 bit PMSRn_[15:0].

**Access** These registers can be read/written in 32-bit units.
Bits 31 to 16 are always read as $0000_H$.
Reading bits 15 to 0 returns the value of register PMn/JPM0.

**Address** PMSRn: <PORTn_base> + $0800_H$ + n x 4
JPMSR0: <JPORT0_base> + $0080_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PMSRn_31 | PMSRn_30 | PMSRn_29 | PMSRn_28 | PMSRn_27 | PMSRn_26 | PMSRn_25 | PMSRn_24 | PMSRn_23 | PMSRn_22 | PMSRn_21 | PMSRn_20 | PMSRn_19 | PMSRn_18 | PMSRn_17 | PMSRn_16 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PMSRn_15 | PMSRn_14 | PMSRn_13 | PMSRn_12 | PMSRn_11 | PMSRn_10 | PMSRn_9 | PMSRn_8 | PMSRn_7 | PMSRn_6 | PMSRn_5 | PMSRn_4 | PMSRn_3 | PMSRn_2 | PMSRn_1 | PMSRn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPMSR0 are named JPMSR0_[31:0].

**Table 2-11 PMSRn/JPMSR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 16 | PMSR n_[31:16] | PMSRn_m specifies whether the value of the corresponding lower bit PMSRn_m value is written to PMn_m:<br>0: PMn_m is independent of PMSRn_m<br>1: PMn_m is PMSRn_m<br>Example:<br>If PMSRn.PMSRn_31 = 1, the value of bit PMSRn.PMSRn_15 is written to bit PMn.PMn_15 and output. |
| 15 to 0 | PMSR n_[15:0] | Specifies the PMn_m value if the corresponding upper bit PMSRn_(m+16) is 1:<br>0: PMn_m = 0<br>1: PMn_m = 1 |

### (6) PIBCn/JPIBC0 - Port input buffer control register

In input port mode (PMCn.PMCn_m = 0 and PMn.PMn_m = 1) this register enables/disables the port pin's input buffer.

**Access** PIBCn: This register can be read/written in 16-bit units.
JPIBC0: This register can be read/written in 8-bit units.

**Address** PIBCn: <PORTn_base> + $4000_H$ + n x 4
JPIBC0: <JPORT0_base> + $0400_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PIBCn_15 | PIBCn_14 | PIBCn_13 | PIBCn_12 | PIBCn_11 | PIBCn_10 | PIBCn_9 | PIBCn_8 | PIBCn_7 | PIBCn_6 | PIBCn_5 | PIBCn_4 | PIBCn_3 | PIBCn_2 | PIBCn_1 | PIBCn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPIBC0 are named JPIBC0_[7:0].

**Table 2-12 PIBCn/JPIBC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PIBCn_[15:0] | Enables/disables the input buffer:<br>0: Input buffer disabled<br>1: Input buffer enabled |

**Note** When the input buffer is disabled, it does not consume current even when the pin level is Hi-Z state. Thus the pin does not need to be fixed to a high or low level externally.

### (7) PFCn/JPFC0 - Port function control register

This register, together with register PFCEn, specifies an alternative function of the pins.

Some alternative functions require direct I/O control of pin Pn_m. For such alternative functions PIPCn.PIPCn_m must be set to 1 as well.
For other alternative functions, input/output must be specified by PMn.PMn_m.

**Access** PFCn: This register can be read/written in 16-bit units.
JPFC0: This register can be read/written in 8-bit units.

**Address** PFCn: <PORTn_base> + $0500_H$ + n x 4
JPFC0: <JPORT0_base> + $0050_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PFCn_15 | PFCn_14 | PFCn_13 | PFCn_12 | PFCn_11 | PFCn_10 | PFCn_9 | PFCn_8 | PFCn_7 | PFCn_6 | PFCn_5 | PFCn_4 | PFCn_3 | PFCn_2 | PFCn_1 | PFCn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPFC0 are named JPFC0_[7:0].

**Table 2-13 PFCn/JPFC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PFCn_[15:0] | Specifies the alternative function of a pin.<br>See *Table 2-4 "Alternative mode selection overview (PMCn.PMCn_m = 1)" on page 40* for details. |

### (8) PFCEn - Port function control expansion register

This register, together with register PFCn, specifies an alternative function of the pins.

Some alternative functions require direct I/O control of pin Pn_m. For such alternative functions PIPCn.PIPCn_m must be set to 1 as well.
For other alternative functions, input/output must be specified by PMn.PMn_m.

**Access** PFCEn: This register can be read/written in 16-bit units.

**Address** PFCEn: <PORTn_base> + $0600_H$ + n x 4

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PFCEn_15 | PFCEn_14 | PFCEn_13 | PFCEn_12 | PFCEn_11 | PFCEn_10 | PFCEn_9 | PFCEn_8 | PFCEn_7 | PFCEn_6 | PFCEn_5 | PFCEn_4 | PFCEn_3 | PFCEn_2 | PFCEn_1 | PFCEn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 2-14 PFCEn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PFCEn_[15:0] | Specifies the alternative function of a pin.<br>See *Table 2-4 "Alternative mode selection overview (PMCn.PMCn_m = 1)" on page 40* for details. |

### 2.3.4 Data input/output registers

#### (1) PBDCn/JPBDC0 - Port bi-direction control register

This register enables the input buffer of Pn_m, if its output buffer is enabled as well.
Thus the concerned port Pn_m is operated in bi-directional mode and the Pn_m pin level is read via PPRn.PPRn_m.

**Note** If Pn_m is not configured as output, the input buffer can not be activated via the PBDCn/JPBDCn register.

**Access** PBDCn: This register can be read/written in 16-bit units.
JPBDC0: This register can be read/written in 8-bit units.

**Address** PBDCn: <PORTn_base> + $4100_H$ + n x 4
JPBDC0: <JPORT0_base> + $0410_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PBDCn_15 | PBDCn_14 | PBDCn_13 | PBDCn_12 | PBDCn_11 | PBDCn_10 | PBDCn_9 | PBDCn_8 | PBDCn_7 | PBDCn_6 | PBDCn_5 | PBDCn_4 | PBDCn_3 | PBDCn_2 | PBDCn_1 | PBDCn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPBDC0 are named JPBDC0_[7:0].

**Table 2-15 PBDCn/JPBDC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PBDCn_[15:0] | Enables/disables bi-directional mode of the corresponding pin:<br>0: Bi-directional mode disabled<br>1: Bi-directional mode enabled |

### (2) PPRn/JPPR0 - Port pin read register

This register reflects the actual level of pin Pn_m, the value of the Pn.Pn_m bit or the level of an alternative output function. The value which is read depends on various control settings as described in *Table 2-5 "PPRn_m read values" on page 41* .

**Access** PPRn: This register can be read/written in 16-bit units.
JPPR0: This register can be read/written in 8-bit units.

**Address** PPRn: <PORTn_base> + $0200_H$ + n x 4
JPPR0: <JPORT0_base> + $0020_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PPRn_15 | PPRn_14 | PPRn_13 | PPRn_12 | PPRn_11 | PPRn_10 | PPRn_9 | PPRn_8 | PPRn_7 | PPRn_6 | PPRn_5 | PPRn_4 | PPRn_3 | PPRn_2 | PPRn_1 | PPRn_0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Note** The control bits of the JTAG port register JPPR0 are named JPPR0_[7:0].

**Table 2-16 PPRn/JPPR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PPRn_[15:0] | Pin Pn_m, Pn.Pn_m value or alternative function output. |

### (3) Pn/JP0 - Port register

This register holds the data Pn.Pn_m to be output via the related port Pn_m in output port mode (PMCn.PMCn_m = 0 and PMn.PMn_m = 0).

**Access** Pn: This register can be read/written in 16-bit units.
JP0: This register can be read/written in 8-bit units.

**Address** Pn: <PORTn_base> + $0000_H$ + n x 4
JP0: <JPORT0_base> + $0000_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pn_15 | Pn_14 | Pn_13 | Pn_12 | Pn_11 | Pn_10 | Pn_9 | Pn_8 | Pn_7 | Pn_6 | Pn_5 | Pn_4 | Pn_3 | Pn_2 | Pn_1 | Pn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JP0 are named JP0_[7:0].

**Table 2-17 Pn/JP0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | Pn_[15:0] | Sets the output level of pin m (m = 0 to 15):<br>0: Outputs low level<br>1: Outputs high level |

**Note** The bits of this register can be manipulated by different means, refer to *2.2.3 "Pin data input/output" on page 41* under the keyword "Pn register write".

### (4) PNOTn/JPNOT0 - Port NOT register

This register allows to invert a bit Pn_m of the port register Pn without directly writing to Pn.

**Access** PNOTn: This register can be read/written in 16-bit units.
JPNOT0: This register can be read/written in 8-bit units.
These registers are always read as $0000_H$.

**Address** PNOTn: <PORTn_base> + $0700_H$ + n x 4
JPNOT0: <JPORT0_base> + $0070_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PNOTn_15 | PNOTn_14 | PNOTn_13 | PNOTn_12 | PNOTn_11 | PNOTn_10 | PNOTn_9 | PNOTn_8 | PNOTn_7 | PNOTn_6 | PNOTn_5 | PNOTn_4 | PNOTn_3 | PNOTn_2 | PNOTn_1 | PNOTn_0 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Note** The control bits of the JTAG port register JPNOT0 are named JPNOT0_[7:0].

**Table 2-18 PNOTn/JPNOT0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PNOT n_[15:0] | Specifies if Pn.Pn_m is inverted:<br>0: Pn.Pn_m is not inverted ($\overline{Pn\_m} \rightarrow$ Pn_m)<br>1: Pn.Pn_m is inverted (Pn_m $\rightarrow$ Pn_m) |

### (5) PSRn/JPSR0 - Port set reset register

This register provides an alternative method to write data to the Pn register.

The register's upper 16 bit PSRn_[31:16] specify which Pn.Mn_m bit will be modified by the corresponding bit of the lower 16 bit PSRn_[15:0].

**Access** These registers can be read/written in 32-bit units.
Bits 31 to 16 are always read as $0000_H$.
Reading bits 15 to 0 returns the value of register Pn/JP0.

**Address** PSRn: <PORTn_base> + $0100_H$ + n x 4
JPSR0: <JPORT0_base> + $0010_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PSRn_31 | PSRn_30 | PSRn_29 | PSRn_28 | PSRn_27 | PSRn_26 | PSRn_25 | PSRn_24 | PSRn_23 | PSRn_22 | PSRn_21 | PSRn_20 | PSRn_19 | PSRn_18 | PSRn_17 | PSRn_16 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PSRn_15 | PSRn_14 | PSRn_13 | PSRn_12 | PSRn_11 | PSRn_10 | PSRn_9 | PSRn_8 | PSRn_7 | PSRn_6 | PSRn_5 | PSRn_4 | PSRn_3 | PSRn_2 | PSRn_1 | PSRn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPSR0 are named JPSR0_[31:0].

**Table 2-19 PSRn/JPSR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 16 | PSRn_[31:16] | PSRn_m specifies whether the value of the corresponding lower bit PSRn_m value is written to Pn_m:<br>0: Pn_m is independent of PSRn_m<br>1: Pn_m is PSRn_m<br>Example:<br>If PSRn.PSRn31 = 1, the value of bit PSRn.PSRn_15 is written to bit Pn.Pn_15 and output.<br>Reading of PSRn_[31:0] returns always $0000_H$. |
| 15 to 0 | PSRn_[15:0] | Specifies the Pn_m value if the corresponding upper bit PSRn_(m+16) is 1:<br>0: Pn_m = 0<br>1: Pn_m = 1<br>Reading of PSRn_[15:0] returns the value of the Pn register. |

### 2.3.5 Configuration of electrical characteristics registers

#### (1) PUn/JPU0 - Pull-up option register

This register specifies whether a pull-up resistor is connected to an input pin.

**Access** PUn: This register can be read/written in 16-bit units.
JPU0: This register can be read/written in 8-bit units.

**Address** PUn: <PORTn_base> + $4300_H$ + n x 4
JPU0: <JPORT0_base> + $0430_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PUn_15 | PUn_14 | PUn_13 | PUn_12 | PUn_11 | PUn_10 | PUn_9 | PUn_8 | PUn_7 | PUn_6 | PUn_5 | PUn_4 | PUn_3 | PUn_2 | PUn_1 | PUn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPU0 are named JPU0_[7:0].

**Table 2-20 PUn/JPU0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PUn_[15:0] | Specifies whether a pull-up resistor is connected to the corresponding pin:<br>0: No pull-up resistor connected<br>1: Pull-up resistor connected |

**Notes**
1. If a pin is configured that both a pull-up resistor (PUn.PUn_m = 1) and a pull-down resistor (PDn.PDn_m = 1) are connected, the pull-down resistor is automatically selected and the pull-up resistor is not connected.
2. The pull-up resistor has no effect when the pin is operated in output mode.

### (2) PDn/JPD0 - Pull-down option register

This register specifies whether a pull-down resistor is connected to an input pin.

**Access** PDn: This register can be read/written in 16-bit units.
JPD0: This register can be read/written in 8-bit units.

**Address** PDn: <PORTn_base> + $4400_H$ + n x 4
JPD0: <JPORT0_base> + $0440_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PD n_15 | PD n_14 | PD n_13 | PD n_12 | PD n_11 | PD n_10 | PD n_9 | PD n_8 | PD n_7 | PD n_6 | PD n_5 | PD n_4 | PD n_3 | PD n_2 | PD n_1 | PD n_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPD0 are named JPD0_[7:0].

**Table 2-21 PDn/JPD0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PD n_[15:0] | Specifies whether a pull-down resistor is connected to the corresponding pin:<br>0: No pull-down resistor connected<br>1: Pull-down resistor connected |

**Notes**
1. If a pin is configured that both a pull-up resistor (PUn.PUn_m = 1) and a pull-down resistor (PDn.PDn_m = 1) are connected, the pull-down resistor is automatically selected and the pull-up resistor is not connected.
2. The pull-down resistor has no effect when the pin is operated in output mode.

### (3) PODCn/JPODC0 - Port open drain control register

This register selects push-pull or open-drain as output buffer function.

**Protection** Writing to this register is protected by a special sequence of instructions. Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** These registers can be read/written in 32-bit units.
The bits 31 to 16 must always be written with "0" and "0" is returned when read.

**Address** PODCn: <PORTn_base> + $4500_H$ + n x 4
JPODC0: <JPORT0_base> + $0450_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PODCn_15 | PODCn_14 | PODCn_13 | PODCn_12 | PODCn_11 | PODCn_10 | PODCn_9 | PODCn_8 | PODCn_7 | PODCn_6 | PODCn_5 | PODCn_4 | PODCn_3 | PODCn_2 | PODCn_1 | PODCn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPODC0 are named JPODC0_[31:0].

**Table 2-22 PODCn/JPODC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PODCn_[15:0] | Specifies the output buffer function:<br>0: Push-pull<br>1: Open-drain |

### (4) PISn/JPIS0 - Port input buffer selection register

This register specifies the input buffer characteristics.

A port can have up to four different input buffer characteristics.
The type of input characteristic is selected by the

- port input buffer selection register PISn
- port input buffer selection expansion register PISEn

**Table 2-23 Port input buffer characteristic selection**

| PISEn_m | PISn_m | Input buffer characteristic |
|---|---|---|
| 0 | 0 | Setting prohibited |
| | 1 | Schmitt2 (type 2) |
| 1 | 0 | Schmitt1 (type 3) |
| | 1 | Schmitt4 (type 4) |

Refer to the Data Sheet for electrical characteristics of the different types and which types are available for each port.

**Access** PISn: This register can be read/written in 16-bit units.
JPIS0: This register can be read/written in 8-bit units.

**Address** PISn: <PORTn_base> + $4700_H$ + n x 4
JPIS0: <JPORT0_base> + $0470_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PISn_15 | PISn_14 | PISn_13 | PISn_12 | PISn_11 | PISn_10 | PISn_9 | PISn_8 | PISn_7 | PISn_6 | PISn_5 | PISn_4 | PISn_3 | PISn_2 | PISn_1 | PISn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPIS0 are named JPIS0_[7:0].

**Table 2-24 PISn/JPIS0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PISn_[15:0] | Specifies the input buffer characteristic of port m (m = 0 to 15) together with the bits PISEn[15:0]. |

### (5) PISEn/JPISE0 - Port input buffer selection expansion register

This register specifies the input buffer characteristics together with the port input selection register PISn.

If a port has up to five input buffer characteristics, the port input selection advanced register PISAn is also valid.

**Access** PISEn: This register can be read/written in 16-bit units.
JPISE0: This register can be read/written in 8-bit units.

**Address** PISEn: <PORTn_base> + $4800_H$ + n x 4
JPISE0: <JPORT0_base> + $0480_H$

**Initial Value** Refer to the section *"V850E2/Fx4-G Port Groups Configuration"*.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PISEn_15 | PISEn_14 | PISEn_13 | PISEn_12 | PISEn_11 | PISEn_10 | PISEn_9 | PISEn_8 | PISEn_7 | PISEn_6 | PISEn_5 | PISEn_4 | PISEn_3 | PISEn_2 | PISEn_1 | PISEn_0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Note** The control bits of the JTAG port register JPISE0 are named JPISE0_[7:0].

**Table 2-25 PISEn/JPISE0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | PISEn_[15:0] | Specifies the input buffer characteristic of port m (m = 0 to 15) together with the bits PISn[15:0].<br>Refer to the PISn register description for how to select the input buffer characteristic. |

## 2.4 V850E2/Fx4-G Port Groups Configuration

This section provides

- an overview of the port register protection clusters, refer to the section *“Port registers protection clusters”*
- general information for all ports , refer to the section *“Common port functions”*
- details of all port groups and their associated control registers for each device, refer to the sections
  - *“V850E2/FF4-G port functions”*
  - *“V850E2/FG4-G port functions”*
- a list of input/output signals with port functionality, refer to the section *“Non-port input/putput signals”*
- an alphabetic pin functions list and the ports, the functions can be assigned to, refer to the section *“Alphabetic pin function list”*
- a description of the port status during and after reset and in stand-by modes, refer to the section *“Port functions during/after reset and in stand-by modes”*
- recommendations concerning unused pins, refer to the section *“Recommended connection of unused pins”*.

### 2.4.1 Port register protection clusters

Several registers of certain port groups n are bundled in port protection clusters:

**Table 2-26 Port protection clusters**

| Port protection cluster | Port groups |
|---|---|
| 1 | JP0 |
| 2 | P0 |
| 3 | P1, P3, P4, P10, P27 |

For further information concerning port register protection refer to the section *“Write protected Registers”* in the chapter *“CPU System Functions”* for a detailed description how to write to write protected registers.

### 2.4.2 Common port functions

This section provides information about special port functions, common to all devices.

#### (1) Initialization of port control registers

The port control registers are initialized by the following reset signals:

**Table 2-27 Port control registers reset signals**

| Port group | Power domain | Reset signal |
|---|---|---|
| JP0, P0 | Always-On-Area | • Reset Controller SYSRES |
| P1, P3, P4, P10, P27 | Isolated-Area-0 | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

#### (2) JP_0 to JP_5: Debug interface

If the debug reset $\overline{\text{DCUTRST}}$ is at high level at reset release, the port of the JP0 port group are used for the debugger interface:

- JP0_0: DCUTDI input
- JP0_1: DCUTDO output
- JP0_2: DCUTCK input
- JP0_3: DCUTMS
- JP0_4: $\overline{\text{DCUTRST}}$
- JP0_5: $\overline{\text{DCUTRDY}}$

Consequently all port and alternative modes on these pins can not be used while the debugger is connected.

Refer to the chapter *"On-chip Debug Unit (OCD)"* and the section *"Operation Modes"* of chapter *"CPU System Functions"* for details.

**Note** In order to connect the debugger via the JP0 pins the flash mask option OPBT0.OPBT0[31] has to be set to 1.

#### (3) JP0_0, JP0_1, JP0_2: Flash programmer

These ports are used for connecting a flash programmer.

Refer to the chapter *"Flash Memory"* and the section *"Operation Modes"* of chapter *"CPU System Functions"* for details.

#### (4) Mode pins

Following ports are used as mode setting signals in combination with the FLMD0 pin:

- P0_1: FLMD1

Refer to the section *"Operation Modes"* of chapter *"CPU System Functions"* for details.

### (5) Permanent inputs

Permanent input means, that the input to a port is unconditionally connected to another module. Thus settings of the port control registers do not impact this connection.

Following ports are permanently connected to dedicated function modules:

**Table 2-28 Permanent input functions**

| Port | Permanent input to | Port | Permanent input to |
|---|---|---|---|
| P10_0 | ADAA0I0 | P10_8 | ADAA0I8 |
| P10_1 | ADAA0I1 | P10_9 | ADAA0I9 |
| P10_2 | ADAA0I2 | P10_10 | ADAA0I10 |
| P10_3 | ADAA0I3 | P10_11 | ADAA0I11 |
| P10_4 | ADAA0I4 | P10_12 | ADAA0I12 |
| P10_5 | ADAA0I5 | P10_13 | ADAA0I13 |
| P10_6 | ADAA0I6 | P10_14 | ADAA0I14 |
| P10_7 | ADAA0I7 | P10_15 | ADAA0I15 |

**Note** If the ports of the port group P10 shall be used in input port mode, set ADAA0CTL1.ADAA0GPS = 1.

### (6) Direct I/O control (PIPC)

Some modules take over the input and output control of the used ports automatically.
These ports have to be set in alternative mode by setting PMCn.PMCn_m, PFCn.PFCn_m and PFCEn.PFCEn_m accordingly and I/O control has to be handed over to the module by setting

PIPCn.PIPCn_m = 1.

The setting of PMn.PMn_m has no more effect for these ports.

The following table lists all alternative modes, where PIPCn.PIPCn_m has to be set to 1.
Note that not all functions in the table below are available for all devices.

**Table 2-29 Alternative modes with PIPCn.PIPCn_m = 1**

| Port | Function | Alternative mode |
|---|---|---|
| Clocked Serial Interfaces G (CSIG): | | |
| P0_14 | CSIG0SO | ALT_OUT4 |
| P0_15 | CSIG0SC | ALT_IN4/ALT_OUT4 |
| P3_5 | CSIG0SC | ALT_IN4/ALT_OUT4 |
| P3_6 | CSIG0SO | ALT_OUT4 |
| P4_4 | CSIG0SO | ALT_OUT2 |
| P4_5 | CSIG0SC | ALT_IN2/ALT_OUT2 |
| P0_1 | CSIG4SO | ALT_OUT2 |
| P0_3 | CSIG4SC | ALT_IN2/ALT_OUT2 |
| P4_7 | CSIG4SO | ALT_OUT2 |
| P4_8 | CSIG4SC | ALT_IN2/ALT_OUT2 |
| P1_3 | CSIG7SO | ALT_OUT4 |
| P1_4 | CSIG7SC | ALT_IN4/ALT_OUT4 |

### 2.4.3 V850E2/FF4-G port functions

This section summarizes all port functions of the V850E2/FF4-G devices and its port control registers.

#### (1) V850E2/FF4-G general I/O functions

The table below shows all alternative functions, that can be applied to the V850E2/FF4-G ports.
It also gives the settings of the control bits PMCn_m, PFCn_m, PFCEn_m and PMn_m to the respective port into the different modes.

**Table 2-30 V850E2/FF4-G general I/O functions (1/3)**

| Port mode | Alternative mode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PMCn_m = 0 | PMCn_m = 1 | | | | | | | |
| | PFCEn_m = 0, PFCn_m = 0 | | PFCEn_m = 0, PFCn_m = 1 | | PFCEn_m = 1, PFCn_m = 0 | | PFCEn_m = 1, PFCn_m = 1 | |
| | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 |
| | ALT_IN1 | ALT_OUT1 | ALT_IN2 | ALT_OUT2 | ALT_IN3 | ALT_OUT3 | ALT_IN4 | ALT_OUT4 |
| **Port group 0 (Always-On-Area, E0VDD/E0VSS power supply):** | | | | | | | | |
| P0_0 | DPIN0 | | CSIG4SSI | | ADCA0 TRG0 | | INTP0 | |
| P0_1 | DPIN1 | | CSIG4 DCS[C] | CSIG4SO | | | INTP1 | TAUB0O1 |
| P0_2 | DPIN2 | | CSIG4SI | | ADCA0 TRG2 | | INTP2 | TAUB0O2 |
| P0_3 | DPIN3 | | CSIG4SC[a] | | ADCA0 TRG1 | | INTP3 | |
| P0_4 | DPIN4 | FCN0TX | | | | | INTP11 | |
| P0_5 | FCN0RX/ DPIN5 | | | | | | INTP12 | |
| P0_6 | FCN1RX/ DPIN6 | URTE11TX | KR0I1 | | NMI | | | |
| P0_7 | URTE11RX/ DPIN7 | FCN1TX | KR0I2 | | INTP4 | | | |
| P0_8 | DPIN8 | URTE10TX | KR0I3 | FCN3TX[b] | | TAUB0O5 | IICB0SDA[a] | |
| P0_9 | URTE10RX/ DPIN9 | FCN2TX | KR0I4 | | INTP6 | TAUB0O6 | IICB0SCL[a] | |
| P0_10 | DPIN10 | URTE11TX | FCN3RX[b] | | | | | |
| P0_11 | DPIN11 | | FCN2RX | | | | | |
| P0_12 | TAUJ0I0/ DPIN12 | TAUJ0O0 | KR0I0 | | INTP8 | FCN4TX[b] | CSIG0SSI | |
| P0_13 | TAUJ0I1/ DPIN13 | TAUJ0O1 | KR0I5 | | | FCN5TX[b] | CSIG0SI | |
| P0_14 | TAUJ0I2 | TAUJ0O2/ DPO | KR0I6 | | FCN5RX[b] | | CSIG0DCS[c] | CSIG0SO |
| P0_15 | TAUJ0I3 | TAUJ0O3/APO | KR0I7 | | FCN4RX[b] | | CSIG0SC[a] | |

**Table 2-30 V850E2/FF4-G general I/O functions (2/3)**

| Port mode | Alternative mode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PMCn_m = 0 | PMCn_m = 1 | | | | | | | |
| | PFCEn_m = 0, PFCn_m = 0 | | PFCEn_m = 0, PFCn_m = 1 | | PFCEn_m = 1, PFCn_m = 0 | | PFCEn_m = 1, PFCn_m = 1 | |
| | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 |
| | ALT_IN1 | ALT_OUT1 | ALT_IN2 | ALT_OUT2 | ALT_IN3 | ALT_OUT3 | ALT_IN4 | ALT_OUT4 |
| **Port group 1 (Isolated-Area-0, E1VDD/E1VSS power supply):** | | | | | | | | |
| P1_1 | TAUB0I1 | TAUB0O1 | | | | | FCN1RX | FCN0TX |
| P1_2 | TAUB0I2 | TAUB0O2 | | | | | | FCN1TX |
| P1_3 | TAUB0I3 | TAUB0O3 | FCN3RX[b] | | | | | |
| P1_4 | TAUB0I4 | TAUB0O4 | | FCN3TX[b] | | | | |
| P1_5 | TAUB0I5 | TAUB0O5 | FCN4RX[b] | | | | | |
| P1_6 | TAUB0I6 | TAUB0O6 | | FCN4TX[b] | | | | |
| P1_7 | TAUB0I7 | TAUB0O7 | | | | | FCN0RX | |
| P1_8 | TAUB0I8 | TAUB0O8 | | | | | FCN2RX | |
| P1_9 | TAUB0I9 | TAUB0O9 | INTP3 | | | | | FCN2TX |
| P1_10 | TAUB0I10 | TAUB0O10 | FCN5RX[b] | | | | INTP4 | |
| P1_11 | TAUB0I11 | TAUB0O11 | | FCN5TX[b] | | | | |
| **Port group 3 (Isolated-Area-0, E1VDD/E1VSS power supply):** | | | | | | | | |
| P3_4 | | | TAUB0I4 | TAUB0O4 | KR0I5 | | CSIG0RYI | CSIG0RYO |
| P3_5 | | | TAUB0I5 | TAUB0O5 | KR0I4 | | CSIG0SC[a] | |
| P3_6 | | | TAUB0I6 | TAUB0O6 | | | CSIG0DCS[c] | CSIG0SO |
| P3_7 | | | TAUB0I7 | TAUB0O7 | | | CSIG0SI | |
| **Port group 4 (Isolated-Area-0, E1VDD/E1VSS power supply):** | | | | | | | | |
| P4_0 | | | TAUB0I13 | TAUB0O13 | | | FCN0RX | |
| P4_1 | | | TAUB0I14 | TAUB0O14 | | | | FCN0TX |
| P4_2 | | | TAUB0I15 | TAUB0O15 | | | FCN1RX | |
| P4_3 | | | CSIG0SI | URTE10TX | | | | FCN1TX |
| P4_4 | INTP2 | | URTE 10RX | CSIG0SO | | | FCN2RX | |
| P4_5 | | | CSIG0SC[a] | | KR0I3 | | | FCN2TX |
| P4_6 | | | CSIG4SI | URTE11TX | KR0I2 | | FCN3RX[b] | |
| P4_7 | INTP4 | | URTE11RX | CSIG4SO | KR0I1 | | | FCN3TX[b] |
| P4_8 | | | CSIG4SC[a] | | KR0I0 | | FCN4RX[b] | |
| P4_9 | | | | CSIG0RYO | | | | FCN4TX[b] |
| P4_10 | | | CSIG4RYI | | | | FCN5RX[b] | |
| P4_11 | | | | | | | | FCN5TX[b] |

**Table 2-30 V850E2/FF4-G general I/O functions (3/3)**

| Port mode | Alternative mode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PMCn_m = 0 | PMCn_m = 1 | | | | | | | |
| | PFCEn_m = 0, PFCn_m = 0 | | PFCEn_m = 0, PFCn_m = 1 | | PFCEn_m = 1, PFCn_m = 0 | | PFCEn_m = 1, PFCn_m = 1 | |
| | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 |
| | ALT_IN1 | ALT_OUT1 | ALT_IN2 | ALT_OUT2 | ALT_IN3 | ALT_OUT3 | ALT_IN4 | ALT_OUT4 |
| **Port group 10 (Isolated-Area-0, A0VDD/A0VSS power supply):** | | | | | | | | |
| P10_3 | | | | | | | | |
| P10_4 | | | | | | | | |
| P10_5 | | | | | | | | |
| P10_6 | | | | | | | | |
| P10_7 | | | | | | | | |
| P10_8 | | | | | | | | |
| P10_9 | ADCA0 TRG0 | | | | | | | |
| P10_10 | ADCA0 TRG1 | | | | | | | |
| P10_11 | ADCA0 TRG2 | | | | | | | |
| P10_12 | | | | | | | | |
| **Port group JP0 (Always-On-Area, E0VDD/E0VSS power supply):** | | | | | | | | |
| JP0_0 | INTP0 | | TAUJ0I0 | TAUJ0O0 | | | | |
| JP0_1 | INTP1 | | TAUJ0I1 | TAUJ0O1 | | | | |
| JP0_2 | INTP2 | | TAUJ0I2 | TAUJ0O2 | | | | |
| JP0_3 | INTP3 | | TAUJ0I3 | TAUJ0O3 | | | | |
| JP0_4 | | | | | | | | |
| JP0_5 | NMI | | | | | | | |

a) To use this alternative function, set the PIPCn.PIPCnm bit to 1. Input or output is not affected by the PMn.PMnm bit setting.
b) This alternative function is not available in µPD70F4177.
c) To use this alternative function, set the CSIGnCTL1.CSIGnDCS bit to 1 (Data Consistency Check enable) and also set the PIPCn.PIPCnm or PBDCn.PBDCnm bit to 1. Data consistency check is not affected by the PMn.PMnm bit setting.

### (2) V850E2/FF4-G port control registers

The following table summarizes all V850E2/FF4-G port control registers, their addresses and intitial values.

**Tables legend**

A: Register address

I: Initial value

B: Available bits
- 1: available, x: not available
- right: bit 0, left: bit 15

**Table 2-31 V850E2/FF4-G port (groups 0, 1, 3, 4) control registers (1/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 0 | 1 | 3 | 4 |
| Pn | A: | FF40 0000$_H$ | FF40 0004$_H$ | FF40 000C$_H$ | FF40 0010$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PSRn | A: | FF40 0100$_H$ | FF40 0104$_H$ | FF40 010C$_H$ | FF40 0110$_H$ |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | 0000 0000$_H$ | 0000 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PNOTn | A: | FF40 0700$_H$ | FF40 0704$_H$ | FF40 070C$_H$ | FF40 0710$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PPRn | A: | FF40 0200$_H$ | FF40 0204$_H$ | FF40 020C$_H$ | FF40 0210$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PMn | A: | FF40 0300$_H$ | FF40 0304$_H$ | FF40 030C$_H$ | FF40 0310$_H$ |
| | I: | FFFF$_H$ | FFFF$_H$ | FFFF$_H$ | FFFF$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PMCn | A: | FF40 0400$_H$ | FF40 0404$_H$ | FF40 040C$_H$ | FF40 0410$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PFCn | A: | FF40 0500$_H$ | FF40 0504$_H$ | FF40 050C$_H$ | FF40 0510$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111<br>1111 1x11 1111 1111 | xxxx 1111 1111 111x<br>xxxx x111 1xxx x11x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111<br>xxxx x111 1111 1111 |
| PFCEn | A: | FF40 0600$_H$ | FF40 0604$_H$ | FF40 060C$_H$ | FF40 0610$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1x11 1111 1111 | xxxx 1111 1xxx x11x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111<br>xxxx xxx1 1111 1111 |
| PMSRn | A: | FF40 0800$_H$ | FF40 0804$_H$ | FF40 080C$_H$ | FF40 0810$_H$ |
| | I: | 0000 FFFF$_H$ | 0000 FFFF$_H$ | 0000 FFFF$_H$ | 0000 FFFF$_H$ |
| | B: | 1111 1x11 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PMCSRn | A: | FF40 0900$_H$ | FF40 0904$_H$ | FF40 090C$_H$ | FF40 0910$_H$ |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | 0000 0000$_H$ | 0000 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |

**Table 2-31 V850E2/FF4-G port (groups 0, 1, 3, 4) control registers (2/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 0 | 1 | 3 | 4 |
| PIBCn | A: | FF40 4000$_H$ | FF40 4004$_H$ | FF40 400C$_H$ | FF40 4010$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PBDCn | A: | FF40 4100$_H$ | FF40 4104$_H$ | FF40 410C$_H$ | FF40 4110$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PIPCn | A: | FF40 4200$_H$ | FF40 4204$_H$ | FF40 420C$_H$ | FF40 4210$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PUn | A: | FF40 4300$_H$ | FF40 4304$_H$ | FF40 430C$_H$ | FF40 4310$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PDn | A: | FF40 4400$_H$ | FF40 4404$_H$ | FF40 440C$_H$ | FF40 4410$_H$ |
| | I: | 0000$_H$ | 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PODCn | A: | FF40 4500$_H$ | FF40 4504$_H$ | FF40 450C$_H$ | FF40 4510$_H$ |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | 0000$_H$ | 0000$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PISn | A: | FF40 4700$_H$ | FF40 4704$_H$ | FF40 470C$_H$ | FF40 4710$_H$ |
| | I: | FFFF$_H$ | FFFF$_H$ | FFFF$_H$ | FFFF$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PISEn | A: | FF40 4800$_H$ | FF40 4804$_H$ | FF40 480C$_H$ | FF40 4810$_H$ |
| | I: | FFFF$_H$ | FFFF$_H$ | FFFF$_H$ | FFFF$_H$ |
| | B: | 1111 1111 1111 1111 | xxxx 1111 1111 111x | xxxx xxxx 1111 xxxx | xxxx 1111 1111 1111 |
| PPCMDn | A: | FF40 4C00$_H$ | FF40 4C04$_H$ | FF40 4C0C$_H$ | FF40 4C10$_H$ |
| | I: | 00$_H$ | 00$_H$ | 00$_H$ | 00$_H$ |
| | B: | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 |
| PPROTSn | A: | FF40 4B00$_H$ | FF40 4B04$_H$ | FF40 4B0C$_H$ | FF40 4B10$_H$ |
| | I: | 00$_H$ | 00$_H$ | 00$_H$ | 00$_H$ |
| | B: | xxxx xxx1 | xxxx xxx1 | xxxx xxx1 | xxxx xxx1 |

**Table 2-32 V850E2/FF4-G port (groups 10, JP0) control registers (1/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 10 | JP0[a] | | |
| Pn | A: | FF40 0028$_H$ | FF44 0000$_H$ | | |
| | I: | 0000$_H$ | 00$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xx11 1111 | | |
| PSRn | A: | FF40 0128$_H$ | FF44 0010$_H$ | | |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xxxx xxxx xx11 1111 | | |
| PNOTn | A: | FF40 0728$_H$ | FF44 0070$_H$ | | |
| | I: | 0000$_H$ | 00$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xx11 1111 | | |
| PPRn | A: | FF40 0228$_H$ | FF44 0020$_H$ | | |
| | I: | FFFF$_H$ | 00$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xx11 1111 | | |
| PMn | A: | FF40 0328$_H$ | FF44 0030$_H$ | | |
| | I: | FFFF$_H$ | FF$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xx11 1111 | | |
| PMCn | A: | FF40 0428$_H$ | FF44 0040$_H$ | | |
| | I: | 0000$_H$ | 00$_H$ | | |
| | B: | xxxx 111x xxxx xxxx | xx1x 1111 | | |
| PFCn | A: | – | FF44 0050$_H$ | | |
| | I: | | 00$_H$ | | |
| | B: | | xxxx 1111 | | |
| PFCEn | A: | – | – | | |
| | I: | | | | |
| | B: | | | | |
| PMSRn | A: | FF40 0828$_H$ | FF44 0080$_H$ | | |
| | I: | 0000 FFFF$_H$ | 0000 00FF$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xxxx xxxx xx11 1111 | | |
| PMCSRn | A: | FF40 0928$_H$ | FF44 0090$_H$ | | |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | | |
| | B: | xxxx 111x xxxx xxxx | xxxx xxxx xx1x 1111 | | |
| PIBCn | A: | FF40 4028$_H$ | FF44 0400$_H$ | | |
| | I: | 0000$_H$ | 00$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xx11 1111 | | |
| PBDCn | A: | FF40 4128$_H$ | FF44 0410$_H$ | | |
| | I: | 0000$_H$ | 00$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xx11 1111 | | |
| PIPCn | A: | – | – | | |
| | I: | | | | |
| | B: | | | | |
| PUn | A: | – | FF44 0430$_H$ | | |
| | I: | | 00$_H$ | | |
| | B: | | xx11 1111 | | |

**Table 2-32 V850E2/FF4-G port (groups 10, JP0) control registers (2/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 10 | JP0[a] | | |
| PDn | A: | – | FF44 0440$_H$ | | |
| | I: | | 00$_H$ | | |
| | B: | | xx11 1111 | | |
| PODCn | A: | FF40 4528$_H$ | FF44 0450$_H$ | | |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | | |
| | B: | xxx1 1111 1111 1xxx | xxxx xxxx xx11 1111 | | |
| PISn | A: | – | FF44 0470$_H$ | | |
| | I: | | FF$_H$ | | |
| | B: | | xx11 1111 | | |
| PISEn | A: | – | FF44 0480$_H$ | | |
| | I: | | FF$_H$ | | |
| | B: | | xx11 1111 | | |
| PPCMDn | A: | FF40 4C28$_H$ | FF44 04C0$_H$ | | |
| | I: | 00$_H$ | 00$_H$ | | |
| | B: | 1111 1111 | 1111 1111 | | |
| PPROTSn | A: | FF40 4B28$_H$ | FF44 04B0$_H$ | | |
| | I: | 00$_H$ | 00$_H$ | | |
| | B: | xxxx xxx1 | xxxx xxx1 | | |

a) The JTAG port group registers are identified by a “J” prefix, with n = 0. For example, JP0 stands for the register name of Pn.

### 2.4.4 V850E2/FG4-G port functions

This section summarizes all port functions of the V850E2/FG4-G devices and its port control registers.

#### (1) V850E2/FG4-G general I/O functions

The table below shows all alternative functions, that can be applied to the V850E2/FG4-G ports.
It also gives the settings of the control bits PMCn_m, PFCn_m, PFCEn_m and PMn_m to the respective port into the different modes.

**Table 2-33 V850E2/FG4-G general I/O functions (1/3)**

| Port mode | Alternative mode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PMCn_m = 0 | PMCn_m = 1 | | | | | | | |
| | PFCEn_m = 0, PFCn_m = 0 | | PFCEn_m = 0, PFCn_m = 1 | | PFCEn_m = 1, PFCn_m = 0 | | PFCEn_m = 1, PFCn_m = 1 | |
| | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 |
| | ALT_IN1 | ALT_OUT1 | ALT_IN2 | ALT_OUT2 | ALT_IN3 | ALT_OUT3 | ALT_IN4 | ALT_OUT4 |
| **Port group 0 (Always-On-Area, E0VDD/E0VSS power supply):** | | | | | | | | |
| P0_0 | DPIN0 | | CSIG4SSI | | ADCA0 TRG0 | | INTP0 | |
| P0_1 | DPIN1 | | CSIG4DCS[c] | CSIG4SO | URTE2RX | | INTP1 | TAUB0O1 |
| P0_2 | DPIN2 | | CSIG4SI | | ADCA0 TRG2 | URTE2TX | INTP2 | TAUB0O2 |
| P0_3 | DPIN3 | | CSIG4SC[a] | | ADCA0 TRG1 | | INTP3 | |
| P0_4 | DPIN4 | FCN0TX | | | | | INTP11 | |
| P0_5 | FCN0RX/ DPIN5 | | | | | | INTP12 | |
| P0_6 | FCN1RX/ DPIN6 | URTE11TX | KR0I1 | | NMI | | | |
| P0_7 | URTE11RX/ DPIN7 | FCN1TX | KR0I2 | | INTP4 | | | |
| P0_8 | DPIN8 | URTE10TX | KR0I3 | FCN3TX[b] | INTP5 | TAUB0O5 | IICB0SDA[a] | |
| P0_9 | URTE10RX/ DPIN9 | FCN2TX | KR0I4 | | INTP6 | TAUB0O6 | IICB0SCL[a] | |
| P0_10 | DPIN10 | URTE11TX | FCN3RX[b] | | INTP9 | | | |
| P0_11 | URTE11RX/ DPIN11 | | FCN2RX | | INTP10 | | | |
| P0_12 | TAUJ0I0/ DPIN12 | TAUJ0O0 | KR0I0 | | INTP8 | FCN4TX[b] | CSIG0SSI | |
| P0_13 | TAUJ0I1/ DPIN13 | TAUJ0O1 | KR0I5 | | INTP7 | FCN5TX[b] | CSIG0SI | |
| P0_14 | TAUJ0I2 | TAUJ0O2/ DPO | KR0I6 | | FCN5RX[b] | | CSIG0DCS[c] | CSIG0SO |
| P0_15 | TAUJ0I3 | TAUJ0O3/APO | KR0I7 | | FCN4RX[b] | | CSIG0SC[a] | |

**Table 2-33 V850E2/FG4-G general I/O functions (2/3)**

| Port mode | Alternative mode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PMCn_m = 0 | PMCn_m = 1 | | | | | | | |
| | PFCEn_m = 0, PFCn_m = 0 | | PFCEn_m = 0, PFCn_m = 1 | | PFCEn_m = 1, PFCn_m = 0 | | PFCEn_m = 1, PFCn_m = 1 | |
| | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 |
| | ALT_IN1 | ALT_OUT1 | ALT_IN2 | ALT_OUT2 | ALT_IN3 | ALT_OUT3 | ALT_IN4 | ALT_OUT4 |
| **Port group 1 (Isolated-Area-0, E1VDD/E1VSS power supply):** | | | | | | | | |
| P1_1 | TAUB0I1 | TAUB0O1 | | | | | FCN1RX | FCN0TX |
| P1_2 | TAUB0I2 | TAUB0O2 | | | | | CSIG7SI | FCN1TX |
| P1_3 | TAUB0I3 | TAUB0O3 | FCN3RX[b] | | | | CSIG7DCS[c] | CSIG7SO |
| P1_4 | TAUB0I4 | TAUB0O4 | | FCN3TX[b] | | | CSIG7SC[a] | |
| P1_5 | TAUB0I5 | TAUB0O5 | FCN4RX[b] | | | | CSIG7RYI | CSIG7RYO |
| P1_6 | TAUB0I6 | TAUB0O6 | | FCN4TX[b] | | | CSIG7SSI | |
| P1_7 | TAUB0I7 | TAUB0O7 | | | | | FCN0RX | |
| P1_8 | TAUB0I8 | TAUB0O8 | | | | | FCN2RX | |
| P1_9 | TAUB0I9 | TAUB0O9 | INTP3 | | | | | FCN2TX |
| P1_10 | TAUB0I10 | TAUB0O10 | FCN5RX[b] | | | | INTP4 | |
| P1_11 | TAUB0I11 | TAUB0O11 | | FCN5TX[b] | | | INTP5 | |
| P1_12 | TAUB0I12 | TAUB0O12 | | | | | INTP6 | |
| P1_13 | TAUB0I13 | TAUB0O13 | | | | | INTP7 | |
| P1_14 | TAUB0I14 | TAUB0O14 | | | | | INTP8 | |
| P1_15 | TAUB0I15 | TAUB0O15 | | | | | INTP9 | |
| **Port group 3 (Isolated-Area-0, E1VDD/E1VSS power supply):** | | | | | | | | |
| P3_2 | | | TAUB0I2 | TAUB0O2 | KR0I7 | | | |
| P3_3 | | | TAUB0I3 | TAUB0O3 | KR0I6 | | | |
| P3_4 | | | TAUB0I4 | TAUB0O4 | KR0I5 | | CSIG0RYI | CSIG0RYO |
| P3_5 | | | TAUB0I5 | TAUB0O5 | KR0I4 | | CSIG0SC[a] | |
| P3_6 | | | TAUB0I6 | TAUB0O6 | | | CSIG0DCS[c] | CSIG0SO |
| P3_7 | | | TAUB0I7 | TAUB0O7 | | | CSIG0SI | |
| **Port group 4 (Isolated-Area-0, E1VDD/E1VSS power supply):** | | | | | | | | |
| P4_0 | | | TAUB0I13 | TAUB0O13 | | | FCN0RX | |
| P4_1 | | | TAUB0I14 | TAUB0O14 | | | URTE2RX | FCN0TX |
| P4_2 | | | TAUB0I15 | TAUB0O15 | | | FCN1RX | URTE2TX |
| P4_3 | | | CSIG0SI | URTE10TX | | | | FCN1TX |
| P4_4 | INTP2 | | URTE10RX | CSIG0SO | | | FCN2RX | |
| P4_5 | | | CSIG0SC[a] | | KR0I3 | | | FCN2TX |
| P4_6 | | | CSIG4SI | URTE11TX | KR0I2 | | FCN3RX[b] | |
| P4_7 | INTP4 | | URTE11RX | CSIG4SO | KR0I1 | | | FCN3TX[b] |
| P4_8 | | | CSIG4SC[a] | | KR0I0 | | FCN4RX[b] | |
| P4_9 | | | | CSIG0RYO | | | | FCN4TX[b] |
| P4_10 | | | CSIG4RYI | | | | FCN5RX[b] | |
| P4_11 | | | | | | | | FCN5TX[b] |
| **Port group 10 (Isolated-Area-0, A0VDD/A0VSS power supply):** | | | | | | | | |
| P10_0 | | | | | | | | |
| P10_1 | | | | | | | | |
| P10_2 | | | | | | | | |
| P10_3 | | | | | | | | |
| P10_4 | | | | | | | | |

**Table 2-33 V850E2/FG4-G general I/O functions (3/3)**

| Port mode | Alternative mode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PMCn_m = 0 | PMCn_m = 1 | | | | | | | |
| | PFCEn_m = 0, PFCn_m = 0 | | PFCEn_m = 0, PFCn_m = 1 | | PFCEn_m = 1, PFCn_m = 0 | | PFCEn_m = 1, PFCn_m = 1 | |
| | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 | PMn_m = 1 | PMn_m = 0 |
| | ALT_IN1 | ALT_OUT1 | ALT_IN2 | ALT_OUT2 | ALT_IN3 | ALT_OUT3 | ALT_IN4 | ALT_OUT4 |
| P10_5 | | | | | | | | |
| P10_6 | | | | | | | | |
| P10_7 | | | | | | | | |
| P10_8 | | | | | | | | |
| P10_9 | ADCA0 TRG0 | | | | | | | |
| P10_10 | ADCA0 TRG1 | | | | | | | |
| P10_11 | ADCA0 TRG2 | | | | | | | |
| P10_12 | | | | | | | | |
| P10_13 | | | | | | | | |
| P10_14 | | | | | | | | |
| P10_15 | | | | | | | | |
| **Port group 27(Isolated-Area-0, E1VDD/ E1VSS power supply):** | | | | | | | | |
| P27_0 | INTP0 | | | | | | | |
| P27_1 | INTP1 | | | | | | | |
| P27_2 | INTP2 | | | | | | | |
| P27_3 | INTP3 | | | | | | | |
| P27_4 | INTP4 | | | | | | | |
| P27_5 | INTP5 | | | | | | | |
| **Port group JP0 (Always-On-Area, E0VDD/E0VSS power supply):** | | | | | | | | |
| JP0_0 | INTP0 | | TAUJ0I0 | TAUJ0O0 | | | | |
| JP0_1 | INTP1 | | TAUJ0I1 | TAUJ0O1 | | | | |
| JP0_2 | INTP2 | | TAUJ0I2 | TAUJ0O2 | | | | |
| JP0_3 | INTP3 | | TAUJ0I3 | TAUJ0O3 | | | | |
| JP0_4 | | | | | | | | |
| JP0_5 | NMI | | | | | | | |

a) To use this alternative function, set the PIPCn.PIPCnm bit to 1. Input or output is not affected by the PMn.PMnm bit setting.
b) This alternative function is not available in µPD70F4179.
c) To use this alternative function, set the CSIGnCTL1.CSIGnDCS bit to 1 (Data Consistency Check enable) and also set the PIPCn.PIPCnm or PBDCn.PBDCnm bit to 1. Data consistency check is not affected by the PMn.PMnm bit setting.

### (2) V850E2/FG4-G port control registers

The following table summarizes all V850E2/FG4-G port control registers, their addresses and intitial values.

**Tables legend**

A: Register address

I: Initial value

B: Available bits
- 1: available, x: not available
- right: bit 0, left: bit 15

**Table 2-34 V850E2/FG4-G port (groups 0, 1, 3, 4) control registers (1/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 0 | 1 | 3 | 4 |
| Pn | A: | FF40 $0000_H$ | FF40 $0004_H$ | FF40 $000C_H$ | FF40 $0010_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PSRn | A: | FF40 $0100_H$ | FF40 $0104_H$ | FF40 $010C_H$ | FF40 $0110_H$ |
| | I: | 0000 $0000_H$ | 0000 $0000_H$ | 0000 $0000_H$ | 0000 $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PNOTn | A: | FF40 $0700_H$ | FF40 $0704_H$ | FF40 $070C_H$ | FF40 $0710_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PPRn | A: | FF40 $0200_H$ | FF40 $0204_H$ | FF40 $020C_H$ | FF40 $0210_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PMn | A: | FF40 $0300_H$ | FF40 $0304_H$ | FF40 $030C_H$ | FF40 $0310_H$ |
| | I: | $FFFF_H$ | $FFFF_H$ | $FFFF_H$ | $FFFF_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PMCn | A: | FF40 $0400_H$ | FF40 $0404_H$ | FF40 $040C_H$ | FF40 $0410_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PFCn | A: | FF40 $0500_H$ | FF40 $0504_H$ | FF40 $050C_H$ | FF40 $0510_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PFCEn | A: | FF40 $0600_H$ | FF40 $0604_H$ | FF40 $060C_H$ | FF40 $0610_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111<br>xxxx 11x1 1111 1111 |
| PMSRn | A: | FF40 $0800_H$ | FF40 $0804_H$ | FF40 $080C_H$ | FF40 $0810_H$ |
| | I: | 0000 $FFFF_H$ | 0000 $FFFF_H$ | 0000 $FFFF_H$ | 0000 $FFFF_H$ |
| | B: | 1111 1111 1111 1111 | 11111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PMCSRn | A: | FF40 $0900_H$ | FF40 $0904_H$ | FF40 $090C_H$ | FF40 $0910_H$ |
| | I: | 0000 $0000_H$ | 0000 $0000_H$ | 0000 $0000_H$ | 0000 $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |

**Table 2-34 V850E2/FG4-G port (groups 0, 1, 3, 4) control registers (2/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 0 | 1 | 3 | 4 |
| PIBCn | A: | FF40 $4000_H$ | FF40 $4004_H$ | FF40 $400C_H$ | FF40 $4010_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PBDCn | A: | FF40 $4100_H$ | FF40 $4104_H$ | FF40 $410C_H$ | FF40 $4110_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PIPCn | A: | FF40 $4200_H$ | FF40 $4204_H$ | FF40 $420C_H$ | FF40 $4210_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PUn | A: | FF40 $4300_H$ | FF40 $4304_H$ | FF40 $430C_H$ | FF40 $4310_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PDn | A: | FF40 $4400_H$ | FF40 $4404_H$ | FF40 $440C_H$ | FF40 $4410_H$ |
| | I: | $0000_H$ | $0000_H$ | $0000_H$ | $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PODCn | A: | FF40 $4500_H$ | FF40 $4504_H$ | FF40 $450C_H$ | FF40 $4510_H$ |
| | I: | 0000 $0000_H$ | 0000 $0000_H$ | 0000 $0000_H$ | 0000 $0000_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PISn | A: | FF40 $4700_H$ | FF40 $4704_H$ | FF40 $470C_H$ | FF40 $4710_H$ |
| | I: | $FFFF_H$ | $FFFF_H$ | $FFFF_H$ | $FFFF_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PISEn | A: | FF40 $4800_H$ | FF40 $4804_H$ | FF40 $480C_H$ | FF40 $4810_H$ |
| | I: | $FFFF_H$ | $FFFF_H$ | $FFFF_H$ | $FFFF_H$ |
| | B: | 1111 1111 1111 1111 | 1111 1111 1111 111x | xxxx xxxx 1111 11xx | xxxx 1111 1111 1111 |
| PPCMDn | A: | FF40 $4C00_H$ | FF40 $4C04_H$ | FF40 $4C0C_H$ | FF40 $4C10_H$ |
| | I: | $00_H$ | $00_H$ | $00_H$ | $00_H$ |
| | B: | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 |
| PPROTSn | A: | FF40 4B00H | FF40 4B04H | FF40 4B0CH | FF40 4B10H |
| | I: | $00_H$ | $00_H$ | $00_H$ | $00_H$ |
| | B: | xxxx xxx1 | xxxx xxx1 | xxxx xxx1 | xxxx xxx1 |

**Table 2-35 V850E2/FG4-G port (groups 10, 27, JP0) control registers (1/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 10 | 27 | JP0[a] | |
| Pn | A: | FF40 0028$_H$ | FF40 006C$_H$ | FF44 0000$_H$ | |
| | I: | 0000$_H$ | 0000$_H$ | 00$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xx11 1111 | |
| PSRn | A: | FF40 0128$_H$ | FF40 016C$_H$ | FF44 0010$_H$ | |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | 0000 0000$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xxxx xxxx xx11 1111 | |
| PNOTn | A: | FF40 0728$_H$ | FF40 076C$_H$ | FF44 0070$_H$ | |
| | I: | 0000$_H$ | 0000$_H$ | 00$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xx11 1111 | |
| PPRn | A: | FF40 0228$_H$ | FF40 026C$_H$ | FF44 0020$_H$ | |
| | I: | 0000$_H$ | 0000$_H$ | 00$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xx11 1111 | |
| PMn | A: | FF40 0328$_H$ | FF40 036C$_H$ | FF44 0030$_H$ | |
| | I: | FFFF$_H$ | FFFF$_H$ | FF$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xx11 1111 | |
| PMCn | A: | FF40 0428$_H$ | FF40 046C$_H$ | FF44 0040$_H$ | |
| | I: | 0000$_H$ | 0000$_H$ | 00$_H$ | |
| | B: | xxxx 111x xxxx xxxx | xxxx xxxx xx11 1111 | xx1x 1111 | |
| PFCn | A: | – | – | FF44 0050$_H$ | |
| | I: | | | 00$_H$ | |
| | B: | | | xxxx 1111 | |
| PFCEn | A: | – | – | – | |
| | I: | | | | |
| | B: | | | | |
| PMSRn | A: | FF40 0828$_H$ | FF40 086C$_H$ | FF44 0080$_H$ | |
| | I: | 0000 FFFF$_H$ | 0000 FFFF$_H$ | 0000 00FF$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xxxx xxxx xx11 1111 | |
| PMCSRn | A: | FF40 0928$_H$ | FF40 096C$_H$ | FF44 0090$_H$ | |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | 0000 0000$_H$ | |
| | B: | xxxx 111x xxxx xxxx | xxxx xxxx xx11 1111 | xxxx xxxx xx1x 1111 | |
| PIBCn | A: | FF40 4028$_H$ | FF40 406C$_H$ | FF44 0400$_H$ | |
| | I: | 0000$_H$ | 0000$_H$ | 00$_H$ | |
| | B: | xxxx 1111 1111 1111 | xxxx xxxx xx11 1111 | xx11 1111 | |
| PBDCn | A: | FF40 4128$_H$ | FF40 416C$_H$ | FF44 0410$_H$ | |
| | I: | 0000$_H$ | 0000$_H$ | 00$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xx11 1111 | |
| PIPCn | A: | – | – | – | |
| | I: | | | | |
| | B: | | | | |
| PUn | A: | – | FF40 436C$_H$ | FF44 0430$_H$ | |
| | I: | | 0000$_H$ | 00$_H$ | |
| | B: | | xxxx xxxx xx11 1111 | xx11 1111 | |

**Table 2-35 V850E2/FG4-G port (groups 10, 27, JP0) control registers (2/2)**

| Register | | Port group n = | | | |
|---|---|---|---|---|---|
| | | 10 | 27 | JP0[a] | |
| PDn | A: | – | FF40 446C$_H$ | FF44 0440$_H$ | |
| | I: | | 0000$_H$ | 0000$_H$ | |
| | B: | | xxxx xxxx xx11 1111 | xx11 1111 | |
| PODCn | A: | FF40 4528$_H$ | FF40 456C$_H$ | FF44 0450$_H$ | |
| | I: | 0000 0000$_H$ | 0000 0000$_H$ | 0000 0000$_H$ | |
| | B: | 1111 1111 1111 1111 | xxxx xxxx xx11 1111 | xxxx xxxx xx11 1111 | |
| PISn | A: | – | FF40 476C$_H$ | FF44 0470$_H$ | |
| | I: | | FFFF$_H$ | FF$_H$ | |
| | B: | | xxxx xxxx xx11 1111 | xx11 1111 | |
| PISEn | A: | – | FF40 486C$_H$ | FF44 0480$_H$ | |
| | I: | | FFFF$_H$ | FF$_H$ | |
| | B: | | xxxx xxxx xx11 1111 | xx11 1111 | |
| PPCMDn | A: | FF40 4C28$_H$ | FF40 4C6C$_H$ | FF44 04C0$_H$ | |
| | I: | 00$_H$ | 00$_H$ | 00$_H$ | |
| | B: | 1111 1111 | 1111 1111 | 1111 1111 | |
| PPROTSn | A: | FF40 4B28$_H$ | FF40 4B6C$_H$ | FF44 04B0$_H$ | |
| | I: | 00$_H$ | 00$_H$ | 00$_H$ | |
| | B: | xxxx xxx1 | xxxx xxx1 | xxxx xxx1 | |

a) The JTAG port group registers are identified by a “J” prefix, with n = 0. For example, JP0 stands for the register name of Pn.

### 2.4.5 Non-port input/output signals

Following signals are input/output via pins without port functionality. Thus no port control registers are available for these pins:

**Table 2-36 General pin functions**

| | |
|---|---|
| Mode signals | • FLMD0<br>• $\overline{\text{RESET}}$ |
| Oscillator/clock connections | • X1, X2 |
| Power supply | • all power supply and voltage reference signals |

### 2.4.6 Alphabetic pin function list

The following table lists all pin signals in alphabetic order and the ports, they can be assigned to.
Pins which are not assigned to any ports, are also listed in the section *"Non-port input/output signals"* above.

**Note** The table shows all V850E2/Fx4-G signals and does not note the availability of a signal on a certain device.

**Table 2-37 Alphabetic pin function list (1/4)**

| Pin name | I/O | Pin function | Port |
|---|---|---|---|
| A0VDD | – | A/D Converter 0 voltage supply | – |
| A0VSS | – | A/D Converter 0 ground | – |
| ADAA0I0 | I | A/D Converter 0 input channel 0 | P10_0 |
| ADAA0I1 | I | A/D Converter 0 input channel 1 | P10_1 |
| ADAA0I2 | I | A/D Converter 0 input channel 2 | P10_2 |
| ADAA0I3 | I | A/D Converter 0 input channel 3 | P10_3 |
| ADAA0I4 | I | A/D Converter 0 input channel 4 | P10_4 |
| ADAA0I5 | I | A/D Converter 0 input channel 5 | P10_5 |
| ADAA0I6 | I | A/D Converter 0 input channel 6 | P10_6 |
| ADAA0I7 | I | A/D Converter 0 input channel 7 | P10_7 |
| ADAA0I8 | I | A/D Converter 0 input channel 8 | P10_8 |
| ADAA0I9 | I | A/D Converter 0 input channel 9 | P10_9 |
| ADAA0I10 | I | A/D Converter 0 input channel 10 | P10_10 |
| ADAA0I11 | I | A/D Converter 0 input channel 11 | P10_11 |
| ADAA0I12 | I | A/D Converter 0 input channel 12 | P10_12 |
| ADAA0I13 | I | A/D Converter 0 input channel 13 | P10_13 |
| ADAA0I14 | I | A/D Converter 0 input channel 14 | P10_14 |
| ADAA0I15 | I | A/D Converter 0 input channel 15 | P10_15 |
| ADAA0TRG0 | I | A/D Converter 0 trigger 0 | P0_0, P10_9 |
| ADAA0TRG1 | I | A/D Converter 0 trigger 1 | P0_3, P10_10 |
| ADAA0TRG2 | I | A/D Converter 0 trigger 2 | P0_2, P10_11 |
| APO | O | Wake-up Sequencer analog input signals sources activation | P0_15 |
| CSIG0RY | I/O | Clocked Serial I/F G 0 handshake signal | P3_4 |
| | O | | P4_9 |
| CSIG0SC | I/O | Clocked Serial I/F G 0 data clock | P0_15, P3_5, P4_5 |
| CSIG0SI | I | Clocked Serial I/F G 0 serial data input | P0_13, P3_7, P4_3 |
| CSIG0SO | O | Clocked Serial I/F G 0 serial data output | P0_14, P3_6, P4_4 |
| $\overline{\text{CSIG0SSI}}$ | I | Clocked Serial I/F G 0 slave select input | P0_12 |
| CSIG4RY | I | Clocked Serial I/F G 4 handshake signal | P4_10 |
| CSIG4SC | I/O | Clocked Serial I/F G 4 data clock | P0_3, P4_8 |
| CSIG4SI | I | Clocked Serial I/F G 4 serial data input | P0_2, P4_6 |
| CSIG4SO | O | Clocked Serial I/F G 4 serial data output | P0_1, P4_7 |
| $\overline{\text{CSIG4SSI}}$ | I | Clocked Serial I/F G 4 slave select input | P0_0 |

Table 2-37 Alphabetic pin function list (2/4)

| Pin name | I/O | Pin function | Port |
|---|---|---|---|
| CSIG7RY | I/O | Clocked Serial I/F G 7 handshake signal | P1_5 |
| CSIG7SC | I/O | Clocked Serial I/F G 7 data clock | P1_4 |
| CSIG7SI | I | Clocked Serial I/F G 7 serial data input | P1_2 |
| CSIG7SO | O | Clocked Serial I/F G 7 serial data output | P1_3 |
| $\overline{\text{CSIG7SSI}}$ | I | Clocked Serial I/F G 7 slave select input | P1_6 |
| $\overline{\text{DCUTRDY}}$ | O | Debug I/F ready signal | JP0_5 |
| DCUTCK | I | Debug I/F clock | JP0_2 |
| DCUTDI | I | Debug I/F data input | JP0_0 |
| DCUTDO | O | Debug I/F data output | JP0_1 |
| DCUTMS | I | Debug I/F mode select | JP0_3 |
| $\overline{\text{DCUTRST}}$ | I | Debug I/F reset | JP0_4 |
| DPIN0 | I | Wake-up Sequencer digital input signal 0 | P0_0 |
| DPIN1 | I | Wake-up Sequencer digital input signal 1 | P0_1 |
| DPIN2 | I | Wake-up Sequencer digital input signal 2 | P0_2 |
| DPIN3 | I | Wake-up Sequencer digital input signal 3 | P0_3 |
| DPIN4 | I | Wake-up Sequencer digital input signal 4 | P0_4 |
| DPIN5 | I | Wake-up Sequencer digital input signal 5 | P0_5 |
| DPIN6 | I | Wake-up Sequencer digital input signal 6 | P0_6 |
| DPIN7 | I | Wake-up Sequencer digital input signal 7 | P0_7 |
| DPIN8 | I | Wake-up Sequencer digital input signal 8 | P0_8 |
| DPIN9 | I | Wake-up Sequencer digital input signal 9 | P0_9 |
| DPIN10 | I | Wake-up Sequencer digital input signal 10 | P0_10 |
| DPIN11 | I | Wake-up Sequencer digital input signal 11 | P0_11 |
| DPIN12 | I | Wake-up Sequencer digital input signal 12 | P0_12 |
| DPIN13 | I | Wake-up Sequencer digital input signal 13 | P0_13 |
| DPO | O | Wake-up Sequencer digital input signals sources activation | P0_14 |
| EnVDD | – | Port buffer voltage supply | – |
| EnVSS | – | Port buffer ground | – |
| FCN0RX | I | CAN I/F 0 receive input | P0_5, P1_7, P4_0 |
| FCN0TX | O | CAN I/F 0 transmit output | P0_4, P1_1, P4_1 |
| FCN1RX | I | CAN I/F 1 receive input | P0_6, P1_1, P4_2 |
| FCN1TX | O | CAN I/F 1 transmit output | P0_7, P1_2, P4_3 |
| FCN2RX | I | CAN I/F 2 receive input | P0_11, P1_8, P4_4 |
| FCN2TX | O | CAN I/F 2 transmit output | P0_9, P1_9, P4_5 |
| FCN3RX | I | CAN I/F 3 receive input | P0_10, P1_3, P4_6 |
| FCN3TX | O | CAN I/F 3 transmit output | P0_8, P1_4, P4_7 |
| FCN4RX | I | CAN I/F 4 receive input | P0_15, P1_5, P4_8 |
| FCN4TX | O | CAN I/F 4 transmit output | P0_12, P1_6, P4_9 |
| FCN5RX | I | CAN I/F 5 receive input | P0_14, P1_10, P4_10 |
| FCN5TX | O | CAN I/F 5 transmit output | P0_13, P1_11, P4_11 |
| FLCS0SCI | I | Flash programmer synchroneous I/F clock input | JP0_2 |

Table 2-37 Alphabetic pin function list (3/4)

| Pin name | I/O | Pin function | Port |
|---|---|---|---|
| FLCS0SI | I | Flash programmer synchroneous I/F data input | JP0_0 |
| FLCS0SO | O | Flash programmer synchroneous I/F data output | JP0_1 |
| FLMD0 | I | Primary operating mode select pin | – |
| FLMD1 | I | Secondary operating mode select pin | P0_1 |
| FLUR0RTX | I/O | Flash programmer asynchroneous I/F data input/output | JP0_0 |
| I0VDD | – | Flash voltage supply | – |
| IICB0SCL | I/O | $I^2C$ Interface 0 clock signal | P0_9 |
| IICB0SDA | I/O | $I^2C$ Interface 0 data/address signal | P0_8 |
| INTP0 | I | External interrupt input 0 | P0_0, JP0_0, P27_0 |
| INTP1 | I | External interrupt input 1 | P0_1, JP0_1, P27_1 |
| INTP2 | I | External interrupt input 2 | P0_2, P4_4, JP0_2, P27_2 |
| INTP3 | I | External interrupt input 3 | P0_3, P1_9, JP0_3, P27_3 |
| INTP4 | I | External interrupt input 4 | P0_7, P1_10, P4_7, P27_4 |
| INTP5 | I | External interrupt input 5 | P0_8, P1_11, P27_5 |
| INTP6 | I | External interrupt input 6 | P0_9, P1_12, |
| INTP7 | I | External interrupt input 7 | P0_13, P1_13, |
| INTP8 | I | External interrupt input 8 | P0_12, P1_14 |
| INTP9 | I | External interrupt input 9 | P0_10, P1_15 |
| INTP10 | I | External interrupt input 10 | P0_11 |
| INTP11 | I | External interrupt input 11 | P0_4 |
| INTP12 | I | External interrupt input 12 | P0_5 |
| KRI0 | I | Key Return input 0 | P0_12, P4_8 |
| KRI1 | I | Key Return input 1 | P0_6, P4_7 |
| KRI2 | I | Key Return input 2 | P0_7, P4_6 |
| KRI3 | I | Key Return input 3 | P0_8, P4_5 |
| KRI4 | I | Key Return input 4 | P0_9, P3_5 |
| KRI5 | I | Key Return input 5 | P0_13, P3_4 |
| KRI6 | I | Key Return input 6 | P0_14, P3_3 |
| KRI7 | I | Key Return input 7 | P0_15, P3_2 |
| NMI | I | External non-maskable interrupt | P0_6, JP0_5 |
| OSCVDD | – | Oscillator voltage supply | – |
| OSCVSS | – | Oscillator ground | – |
| REGnC | – | Voltage regulators capacitor connections | – |
| REGnVDD | – | Votage regulators input | – |
| REGnVSS | – | Votage regulators ground | – |
| $\overline{\text{RESET}}$ | I | External reset input | – |
| TAUB0I1 | I | Timer Array Unit B 0 channel 1input | P1_1 |
| TAUB0I2 | I | Timer Array Unit B 0 channel 2 input | P1_2, P3_2 |
| TAUB0I3 | I | Timer Array Unit B 0 channel 3 input | P1_3, P3_3 |
| TAUB0I4 | I | Timer Array Unit B 0 channel 4 input | P1_4, P3_4 |
| TAUB0I5 | I | Timer Array Unit B 0 channel 5 input | P1_5, P3_5 |

Table 2-37 Alphabetic pin function list (4/4)

| Pin name | I/O | Pin function | Port |
|---|---|---|---|
| TAUB0I6 | I | Timer Array Unit B 0 channel 6 input | P1_6, P3_6 |
| TAUB0I7 | I | Timer Array Unit B 0 channel 7 input | P1_7, P3_7 |
| TAUB0I8 | I | Timer Array Unit B 0 channel 8 input | P1_8 |
| TAUB0I9 | I | Timer Array Unit B 0 channel 9 input | P1_9 |
| TAUB0I10 | I | Timer Array Unit B 0 channel 10 input | P1_10 |
| TAUB0I11 | I | Timer Array Unit B 0 channel 11 input | P1_11 |
| TAUB0I12 | I | Timer Array Unit B 0 channel 12 input | P1_12 |
| TAUB0I13 | I | Timer Array Unit B 0 channel 13 input | P1_13, P4_0 |
| TAUB0I14 | I | Timer Array Unit B 0 channel 14 input | P1_14, P4_1 |
| TAUB0I15 | I | Timer Array Unit B 0 channel 15 input | P1_15, P4_2 |
| TAUB0O1 | O | Timer Array Unit B 0 channel 1 output | P1_1, P0_1 |
| TAUB0O2 | O | Timer Array Unit B 0 channel 2 output | P1_2, P3_2, P0_2 |
| TAUB0O3 | O | Timer Array Unit B 0 channel 3 output | P1_3, P3_3 |
| TAUB0O4 | O | Timer Array Unit B 0 channel 4 output | P1_4, P3_4 |
| TAUB0O5 | O | Timer Array Unit B 0 channel 5 output | P1_5, P3_5, P0_8 |
| TAUB0O6 | O | Timer Array Unit B 0 channel 6 output | P1_6, P3_6, P0_9 |
| TAUB0O7 | O | Timer Array Unit B 0 channel 7 output | P1_7, P3_7 |
| TAUB0O8 | O | Timer Array Unit B 0 channel 8 output | P1_8 |
| TAUB0O9 | O | Timer Array Unit B 0 channel 9 output | P1_9 |
| TAUB0O10 | O | Timer Array Unit B 0 channel 10 output | P1_10 |
| TAUB0O11 | O | Timer Array Unit B 0 channel 11 output | P1_11 |
| TAUB0O12 | O | Timer Array Unit B 0 channel 12 output | P1_12 |
| TAUB0O13 | O | Timer Array Unit B 0 channel 13 output | P1_13, P4_0 |
| TAUB0O14 | O | Timer Array Unit B 0 channel 14 output | P1_14, P4_1 |
| TAUB0O15 | O | Timer Array Unit B 0 channel 15 output | P1_15, P4_2 |
| TAUJ0I0 | I | Timer Array Unit J 0 channel 0 input | P0_12, JP0_0 |
| TAUJ0I1 | I | Timer Array Unit J 0 channel 1 input | P0_13, JP0_1 |
| TAUJ0I2 | I | Timer Array Unit J 0 channel 2 input | P0_14, JP0_2 |
| TAUJ0I3 | I | Timer Array Unit J 0 channel 3 input | P0_15, JP0_3 |
| TAUJ0O0 | O | Timer Array Unit J 0 channel 0 output | P0_12, JP0_0 |
| TAUJ0O1 | O | Timer Array Unit J 0 channel 1 output | P0_13, JP0_1 |
| TAUJ0O2 | O | Timer Array Unit J 0 channel 2 output | P0_14, JP0_2 |
| TAUJ0O3 | O | Timer Array Unit J 0 channel 3 output | P0_15, JP0_3 |
| URTE2RX | I | Asynchroneous Serial I/F 2 receive data input | P0_1, P4_1 |
| URTE2TX | O | Asynchroneous Serial I/F 2 receive data output | P0_2, P4_2 |
| URTE10RX | I | Asynchroneous Serial I/F 10 receive data input | P0_9, P4_4 |
| URTE10TX | O | Asynchroneous Serial I/F 10 receive data output | P0_8, P4_3 |
| URTE11RX | I | Asynchroneous Serial I/F 11 receive data input | P0_7, P0_11, P4_7 |
| URTE11TX | O | Asynchroneous Serial I/F 11 receive data output | P0_6, P0_10, P4_6 |
| X1, X2 | – | Main oscillator resonator connections | – |

### 2.4.7 Port and pin functions in stand-by modes

Details about the port functions in stand-by modes are given in the chapter *"Stand-by Controller"*.

### 2.4.8 Port and pin functions during and after reset

**Table 2-38 Ports and pins function during and after reset**

| Port | During reset | After reset | Comment |
|---|---|---|---|
| JP0_0 to JP0_3, JP0_5 | high impedance | mode dependent | These ports are also used for the On-Chip Debug I/F. Thus their function after reset depends on the operation mode.[a] |
| JP0_4 | input with internal pull-down resistor | mode dependent | |
| P0_0, P0_2 to P0_15 | high impedance | high impedance | |
| P0_1 | input | input | This port is also used as FLMD1 mode selection input.[a] |
| Port groups P1, P3, P4, P10, P27 | high impedance | high impedance | |
| FLMD0 | input | input | |
| $\overline{\text{RESET}}$ | input | input | |

a) Refer to the section *"Mode pins and JP0 connections"* in the *"CPU System Functions"* chapter.

### 2.4.9 Recommended connection of unused pins

In the following recommendations are given how to treat pins, which are not used on the application board.

As a reference the final table lists all pins which must be connected in any case.

#### (1) Unused pins with port functionality

**Table 2-39 Recommended connection of unused pins with port functionality**

| Port | After reset | Connect to | Comment |
|---|---|---|---|
| P0_1/FLMD1[a] | The function of these ports depends on the operation mode.<br>Refer to the section *"Mode pins and JP0 connections"* in the *"CPU System Functions"* chapter. | | |
| JP0_4/ $\overline{\text{DCUTRST}}$[a] | | | |
| JP0_0 to JP0_3, JP0_5 | | | |
| P0_0, P0_2 to P0_15, P5 | high impedance[b] | leave open | If the high impedance reset configuration is changed to<br>• output: leave open<br>• input: connect to E0VDD or E0VSS via resistor |
| Port groups P1, P3, P4 | high impedance[b] | leave open | If the high impedance reset configuration is changed to<br>• output: leave open<br>• input: connect to E1VDD or E1VSS via resistor |
| Port group P10 | high impedance[b] | leave open | If the high impedance reset configuration is changed to<br>• output: leave open<br>• input: connect to A0VDD or A0VSS via resistor |
| Port group P27 | high impedance[b] | leave open | If the high impedance reset configuration is changed to<br>• output: leave open<br>• input: connect to E1VDD or E1VSS via resistor |

a) Refer also to *"Mandatory connection of pins"* below.
b) Refer to the note below.

**Note** It is recommended to leave pins, which are in high impedance state after reset release, in this state, if they are not used by the application.

#### (2) Unused pins without port functionality

**Table 2-40 Recommended connection of unused pins without port functionality**

| Pin | Connect to | Comment |
|---|---|---|
| X1 | connect to OSCVSS | If MainOsc not used |
| X2 | leave open | |
| $\overline{\text{RESET}}$ | connect to E0VDD | If external $\overline{\text{RESET}}$ is not used, the device is only reset by on-chip Power-On-Clear POCRES |

### (3) Mandatory connection of pins

**Table 2-41 Mandatory connection pins**

| Pin | Comment |
|---|---|
| FLMD0<br>FLMD1/P0_1<br>DCUTRST/JP0_4 | FLMD0 determines the operation mode of the device.<br>FLMD1 and DCUTRST must not be left unconnected.<br>Refer to the section *"Mode pins and JP0 connections"* in the *"CPU System Functions"* chapter. |
| REGnVDD/REGnVSS | All power supply pins must always be connected, although the external power supplies for the isolated areas (REGnVDD) may be switched off in DEEPSTOP stand-by mode. |
| A0VDD, A0VSS | |
| EnVDD, EnVSS | |
| OSCVDD, OSCVSS | |
| I0VDD | |

## 2.5 Port Filters

The input signals at some pins are passed through a filter to remove noise and glitches. The microcontroller supports different types of analog and digital filters.

The first section provides an overview, which port input signals are equipped with which kind of filter type, their control registers and bits, the register addresses, and the power domain they are located on.
The last paragraph of this section informs about the clock signals for the port filters

A detailed description of the analog and digital filter types and their control registers follows in the section *"Port Filters Functional Description"*.

### 2.5.1 Port filters assignment

The following tables list the input signals, which are equipped with an analog filter or a digital filter.

#### (1) Input signals with analog filters type A

The analog filters type A are controlled by means of the following register:

- Filter control registers FCLAnCTLm (m = 0 to 7)
  For each port with an analog filter a dedicated register FCLAnCTLm is provided, whereas each group "n" can handle up to 8 input signals "m".

**Table 2-42 Input signals with analog filters type A (1/2)**

| Signal | FCLA instance | | |
|---|---|---|---|
| | Register | | Address |
| INTP0 | FCLA0 | CTL0 | FF41 4000$_H$ |
| INTP1 | | CTL1 | FF41 4004$_H$ |
| INTP2 | | CTL2 | FF41 4008$_H$ |
| INTP3 | | CTL3 | FF41 400C$_H$ |
| INTP4 | | CTL4 | FF41 4010$_H$ |
| INTP5 | | CTL5 | FF41 4014$_H$ |
| INTP6 | | CTL6 | FF41 4018$_H$ |
| INTP7 | | CTL7 | FF41 401C$_H$ |
| INTP8 | FCLA1 | CTL0 | FF41 4020$_H$ |
| INTP9 | | CTL1 | FF41 4024$_H$ |
| INTP10 | | CTL2 | FF41 4028$_H$ |
| INTP11 | | CTL3 | FF41 402C$_H$ |
| INTP12 | | CTL4 | FF41 4030$_H$ |
| NMI | FCLA2 | CTL0 | FF41 4040$_H$ |

**Table 2-42 Input signals with analog filters type A (2/2)**

| Signal | FCLA instance | | |
|---|---|---|---|
| | Register | | Address |
| KR0 | FCLA3 | CTL0 | FF41 4060$_H$ |
| KR1 | | CTL1 | FF41 4064$_H$ |
| KR2 | | CTL2 | FF41 4068$_H$ |
| KR3 | | CTL3 | FF41 406C$_H$ |
| KR4 | | CTL4 | FF41 4070$_H$ |
| KR5 | | CTL5 | FF41 4074$_H$ |
| KR6 | | CTL6 | FF41 4078$_H$ |
| KR7 | | CTL7 | FF41 407C$_H$ |

### (2) Input signals with analog filters type B

The analog filters type B are controlled by means of the following register:

- Filter control registers FCLAnCTLm (m = 0 to 7)
  For each port with an analog filter a dedicated register FCLAnCTLm is provided, whereas each group “n” can handle up to 8 input signals “m”.

**Table 2-43 Input signals with analog filters type B**

| Signal | FCLA instance | | |
|---|---|---|---|
| | Register | | Address |
| TAUJ0I0 | FCLA4 | CTL0 | FF41 4080$_H$ |
| TAUJ0I1 | | CTL1 | FF41 4084$_H$ |
| TAUJ0I2 | | CTL2 | FF41 4088$_H$ |
| TAUJ0I3 | | CTL3 | FF41 408C$_H$ |

### (3) Input signals with analog filters type C

The analog filters type C have no control registers.

**Table 2-44 Input signals with analog filters type C**

| Signal | FCLA instance | | |
|---|---|---|---|
| | Register | | Address |
| **Always-On-Area:** | | | |
| FLMD0 | – | – | – |
| FLMD1 | | – | – |
| $\overline{\text{RESET}}$ | | – | – |

**(4) Input signals with digital filters type D**

The digital filters type D are controlled by means of the following register:

- Filter control registers FCLAnCTLm (m = 0 to 7)
  For each port with a digital filter a dedicated register FCLAnCTLm is provided, whereas each group “n” can handle up to 8 input signals “m”.
- Digital noise elimination control registers DNFAnCTL
  Each DNFAnCTL control register can handle a group “n” with up to 16 input signals with digital filters.
- Digital noise elimination enable registers DNFAnEN
  The bits DNFAnNFEN[15:0] of this register enable digital filters of the group “n” of up to 16 input signals.

**Caution** If digital filtering shall not be applied to the input signal, the filter bypass must be activated by FCLAnCTLm.FCLAnBYPSm = 1.

**Table 2-45 Input signals with digital filters type D (1/2)**

| Signal | DNFA instance | | | | FCLA instance | | |
|---|---|---|---|---|---|---|---|
| | Register | Address | Filter enable bit | | Register | | Address |
| TAUB0I1 | DNFA0EN<br>DNFA0ENH<br>DNFA0ENL | FF41 1004$_H$<br>FF41 1008$_H$<br>FF41 100C$_H$ | DNFA0EN.DNFA0 | NFEN1 | FCLA5 | CTL1 | FF41 5004$_H$ |
| TAUB0I2 | | | | NFEN2 | | CTL2 | FF41 5008$_H$ |
| TAUB0I3 | | | | NFEN3 | | CTL3 | FF41 500C$_H$ |
| TAUB0I4 | | | | NFEN4 | | CTL4 | FF41 5010$_H$ |
| TAUB0I5 | | | | NFEN5 | | CTL5 | FF41 5014$_H$ |
| TAUB0I6 | | | | NFEN6 | | CTL6 | FF41 5018$_H$ |
| TAUB0I7 | | | | NFEN7 | | CTL7 | FF41 501C$_H$ |
| TAUB0I8 | | | | NFEN8 | FCLA6 | CTL0 | FF41 5020$_H$ |
| TAUB0I9 | | | | NFEN9 | | CTL1 | FF41 5024$_H$ |
| TAUB0I10 | | | | NFEN10 | | CTL2 | FF41 5028$_H$ |
| TAUB0I11 | | | | NFEN11 | | CTL3 | FF41 502C$_H$ |
| TAUB0I12 | | | | NFEN12 | | CTL4 | FF41 5030$_H$ |
| TAUB0I13 | | | | NFEN13 | | CTL5 | FF41 5034$_H$ |
| TAUB0I14 | | | | NFEN14 | | CTL6 | FF41 5038$_H$ |
| TAUB0I15 | | | | NFEN15 | | CTL7 | FF41 503C$_H$ |
| URTE10RX | DNFA1CTL<br>DNFA1EN<br>DNFA1ENH<br>DNFA1ENL | FF41 1020$_H$<br>FF41 1024$_H$<br>FF41 1028$_H$<br>FF41 102C$_H$ | DNFA1EN.DNFA1 | NFEN0 | FCLA7 | CTL0 | FF41 5040$_H$ |
| URTE11RX | | | | NFEN1 | | CTL1 | FF41 5044$_H$ |
| CSIG4SC | | | | NFEN2 | | CTL2 | FF41 5048$_H$ |
| CSIG4SI | | | | NFEN3 | | CTL3 | FF41 504C$_H$ |
| CSIG4RY | | | | NFEN4 | | CTL4 | FF41 5050$_H$ |
| $\overline{\text{CSIG4SSI}}$ | | | | NFEN5 | | CTL5 | FF41 5054$_H$ |
| ADAA0TRG0 | | | | NFEN6 | FCLA8 | CTL0 | FF41 5060$_H$ |
| ADAA0TRG1 | | | | NFEN7 | | CTL1 | FF41 5064$_H$ |
| ADAA0TRG2 | | | | NFEN8 | | CTL2 | FF41 5068$_H$ |

**Table 2-45 Input signals with digital filters type D (2/2)**

| Signal | DNFA instance | | | | FCLA instance | | |
|---|---|---|---|---|---|---|---|
| | Register | Address | Filter enable bit | | Register | | Address |
| CSIG7SC | DNFA9CTL DNFA9EN DNFA9ENH DNFA9ENL | FF41 20E0$_H$ FF41 20E4$_H$ FF41 20E8$_H$ FF41 20EC$_H$ | DNFA9EN.DNFA9 | NFEN8 | FCLA23 | CTL0 | FF41 61C0$_H$ |
| CSIG7RY | | | | NFEN9 | | CTL1 | FF41 61C4$_H$ |
| CSIG7SI | | | | NFEN10 | | CTL2 | FF41 61C8$_H$ |
| $\overline{\text{CSIG7SSI}}$ | | | | NFEN11 | | CTL3 | FF41 61CC$_H$ |
| CSIG0SC | DNFA10CTL DNFA10EN DNFA10ENH DNFA10ENL | FF41 2100$_H$ FF41 2104$_H$ FF41 2108$_H$ FF41 210C$_H$ | DNFA10EN.DNFA10 | NFEN0 | FCLA24 | CTL0 | FF41 61E0$_H$ |
| CSIG0RY | | | | NFEN1 | | CTL1 | FF41 61E4$_H$ |
| CSIG0SI | | | | NFEN2 | | CTL2 | FF41 61E8$_H$ |
| $\overline{\text{CSIG0SSI}}$ | | | | NFEN3 | | CTL3 | FF41 61EC$_H$ |
| URTE2RX | DNFA11CTL DNFA11EN DNFA11ENH DNFA11ENL | FF41 2120$_H$ FF41 2124$_H$ FF41 2128$_H$ FF41 212C$_H$ | DNFA11EN.DNFA11 | NFEN8 | FCLA27 | CTL0 | FF41 6240$_H$ |

### 2.5.2 Port filters clock supply

Following table sumamrizes the clock supplies for the port filter types on the different power domains:

**Table 2-46 Port filter clock supply**

| Power domain | Filters clocks | Connected to |
|---|---|---|
| Always-On-Area | PCLK | Clock Controller CKSCLK_A02 |
| Isolated-Area-0 | PCLK | Clock Controller CKSCLK_001 |
| | DNFATCKI | Clock Controller CKSCLK_016 |

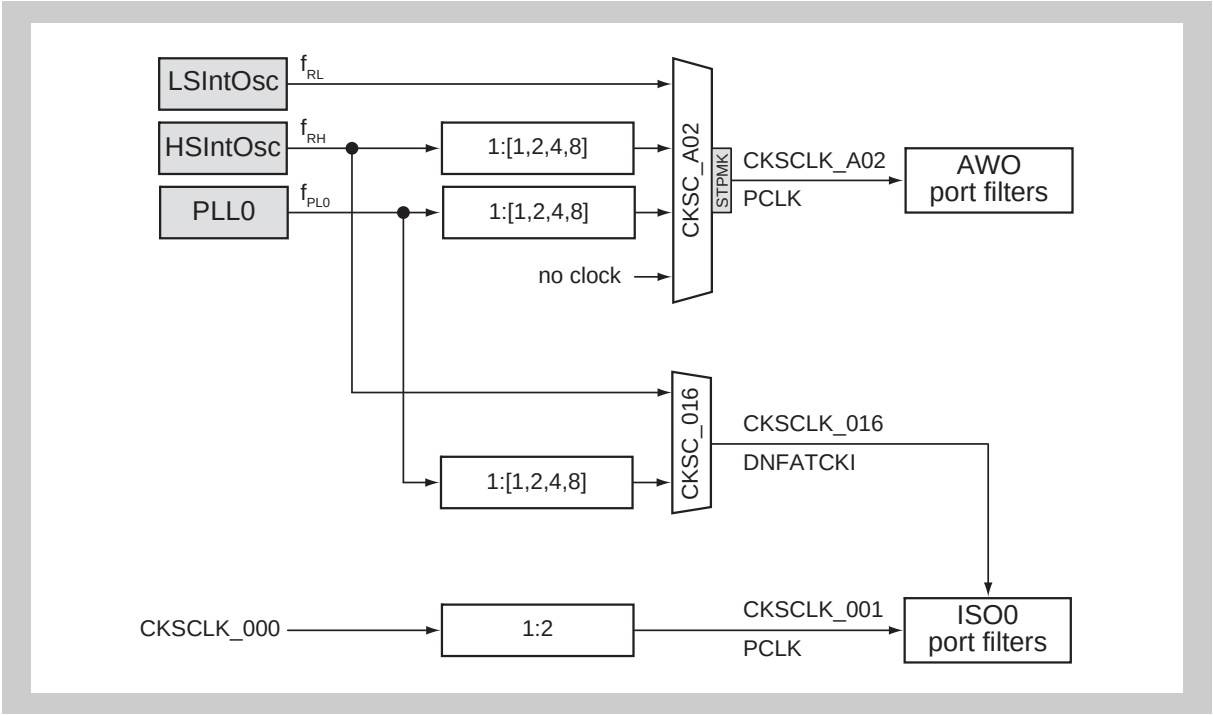



**Figure 2-2 Port filters clock supply**

### 2.5.3 Port filters reset

The port filters and its registers are initialized by the following reset signal:

**Table 2-47 Port filters reset signal**

| Port filters power domain | Reset signal |
|---|---|
| Always-On-Area | • Reset Controller SYSRES |
| Isolated-Area-0 | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

## 2.6 Port Filters Functional Description

Depending on the purpose of the external input signal to be filtered, the external signals are passed through different filter types:

- Analog filters
  Analog filter have a fixed filter characteristic.
  - Type A: analog filtered signals with edge or level detection
    The output signals are used to signal an external event, whereas the timing of the external signal is not of concern, but its level or level changes.
    An external interrupt is a typical case for such event signals.
  - Type B: analog filtered signals with filter bypass option
    The timing of the filtered output signals is retained with this filter type. Bypassing of the filter is also possible.
    A typical example for such signals is a timer input signal to measure its frequency.
  - Type C: analog filter only
    The input signal are always passed through an analog filter, which can not be bypassed.
    Such filters are typically used for external $\overline{\text{RESET}}$ inputs and mode signals.
- Digital filters
  The characteristic of a digital filter can be adjusted to the application's needs.
  - Type D: configurable digital filtered signals with filter bypass option
    The timing of the filtered output signals is retained with this filter type. Bypassing of the filter is also possible.
    A typical example for such signals is a timer input signal to measure its frequency.

### 2.6.1 Analog filters

**Analog filter characteristic**

The characteristics of the analog filter as well as of the level and edge detectors are specified in the Data Sheet document.

**Analog filters control registers**

For each input signal, that is equipped with an analog filter, a dedicated control register FCLAnCTLm is provided.

The registers are ordered in groups of 8 registers with the same index n. The register index m ranges from 0 to 7:

FCLA group n: FCLAnCTL0 to FCLAnCTL7

The assignment of the input signals to the control registers and their addresses is given in the tables in the previous section *"Port filters assignments"*.

**Analog filter in stand-by mode**

All analog filters are located on the Always-On-Area.
The behaviour of an analog filter and its wake-up capability depend on the filter type. Refer to the description of the analog filter types below.

### (1) Analog filter type A

The block diagram of the analog filter type A is shown in the diagram below.

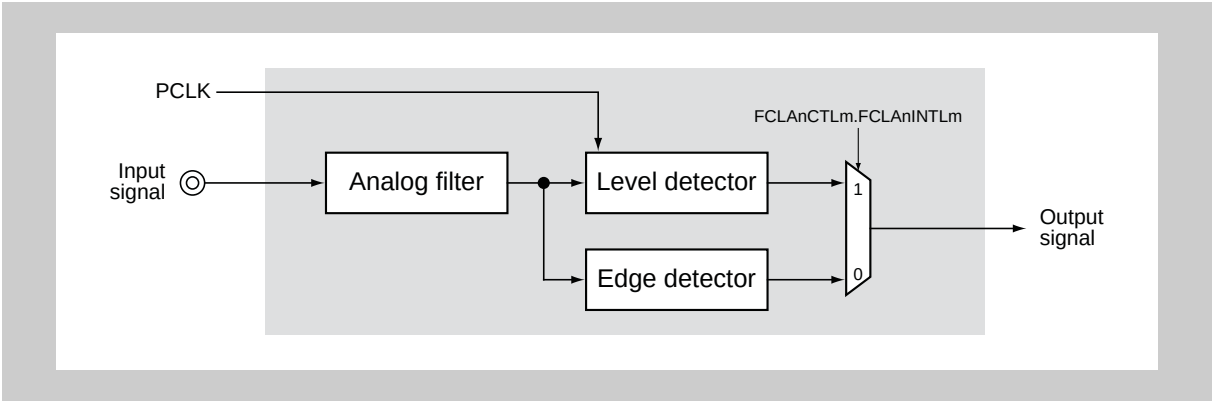



**Figure 2-3 Analog filter type A block diagram**

After passing the external signal through an analog filter to eliminate noise and spikes, the event detection evaluates the level or any level change, i.e. an edge, of the signal and generates an output accordingly.

The detection mode is selected by the control bit FCLAnCTLm.FCLAnINTLm:

- FCLAnINTLm = 0: edge detection mode
  The detection of a rising or falling edge can be activated separately by FCLAnCTLm.FCLAnINTRm and FCLAnCTLm.FCLAnINTFm respectively.
- FCLAnINTLm = 1: level detection mode
  The detection of a high level or low level can be selected by FCLAnCTLm.FCLAnINTRm.

The table below summarizes the detection conditions of the analog filter.

**Table 2-48 Analog filter event detection conditions**

| FCLAnINTLm | FCLAnINTFm | FCLAnINTRm | Edge detection | Level detection |
|---|---|---|---|---|
| 0 | 0 | 0 | no detection | not active |
| | | 1 | rising edge | |
| | 1 | 0 | falling edge | |
| | | 1 | both edges | |
| 1 | X | 0 | not active | low level |
| | | 1 | | high level |

**Default configuration**

The default configuration of the analog filter type A is as follows:

- analog filter with edge detection

**Analog filter type A in stand-by mode**

In case the clock PCLK is stopped in stand-by mode, the analog filter type A can only operate with the edge detection. Thus set FCLAnINTLm = 0 and select the required edge detection if the input signal shall be used as a stand-by mode wake-up signal.

### (2) Analog filter type B

The block diagram of the analog filter type B is shown in the diagram below.

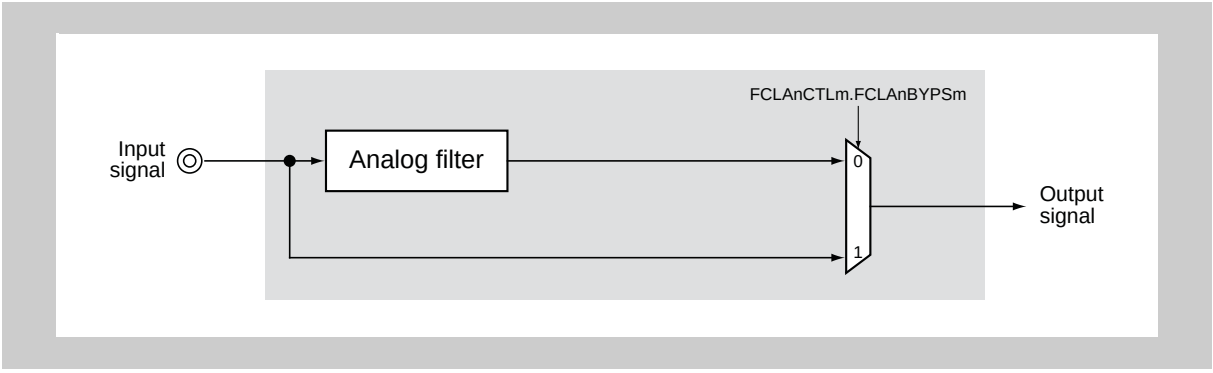



**Figure 2-4 Analog filter type B block diagram**

The analog filter can optionally be bypassed:

- DCLAnCTLm.FCLAnBYPSm = 0: the filtered signal is output
- DCLAnCTLm.FCLAnBYPSm = 1: the unfiltered input signal is output

**Default configuration**

The default configuration of the analog filter type B is as follows:

- analog filter active

**Analog filter type B in stand-by mode**

The output signal of an analog filter type B can always be used as a stand-by mode wake-up signal.

### (3) Analog filter type C

The block diagram of the analog filter type C is shown in the diagram below.

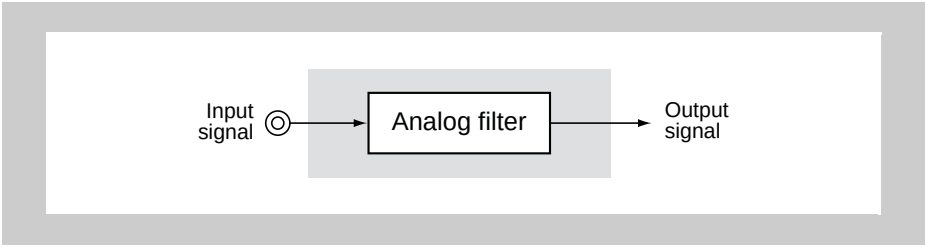



**Figure 2-5 Analog filter type C block diagram**

The output signal is always the analog filtered input signal.

**Analog filter type C in stand-by mode**

The output signal of an analog filter type C can always be used as a stand-by mode wake-up signal.

### 2.6.2 Digital filters

**Digital filter characteristic**

The digital filters allow to adjust the filter characteristics to the needs of the application.

The input signal is sampled with the sampling frequency $f_s$.

If a specified number of successive samples yield the same (high or low) level, the signal level is judged as valid and the filter output signal is set accordingly.

If an external signal level change is detected within the specified number of samples (same level samples s), the signal level is judged as noise - or a spike - and the filter output signal does not change.

The length of an external signal pulse to be judged as noise depends on the sampling frequency and the specified number of same level samples s.

Both parameters can be specified:

- DNFAnCTL.DNFAnPRS[2:0] allows to select the sampling frequency to

$$f_s = \frac{f_{DNFATCKI}}{2^{DNFAnPRS[2:0]}}$$

  where $f_{DNFATCKI}$ is the frequency of the DNFATCKI clock.

- DNFAnCTL.DNFAnNFSTS[1:0] determines the number s of same level samples:

$$s = DNFAnNFSTS[1:0] + 2$$

External signal pulses, shorter than

$$t_{wDNF(min)} = (s-1)x(0)\frac{1}{f_s}$$

are always suppressed. That means also, pulses with a width $\geq t_{wDNF(min)}$ may pass the filter.

External signal pulses, longer than

$$t_{wDNF(max)} = sx\frac{1}{f_s}$$

are always judged as valid and are passed on to the filter output. That means also, pulses with a width $\leq t_{wDNF(max)}$ may be suppressed.

Consequently, external signal pulses, with a $t_{wDNF}$ width in the range

$$(s-1)x\frac{1}{f_s} \leq t_{wDNF} \leq sx\frac{1}{f_s}$$

may be suppressed or judged as valid.

The filter effects a delay $t_{dDNF}$ between the filter input and output pulse in the range

$$(s-1)x\frac{1}{f_s} + \frac{2}{f_{DNFATCKI}} \leq t_{dDNF} \leq sx\left(\frac{1}{f_s} + \frac{3}{f_{DNFATCKI}}\right)$$

The filter operation is illustrated in the figure below with DNFAnNFSTS[1:0] = $01_B$, i.e. s = 3 same level samples.

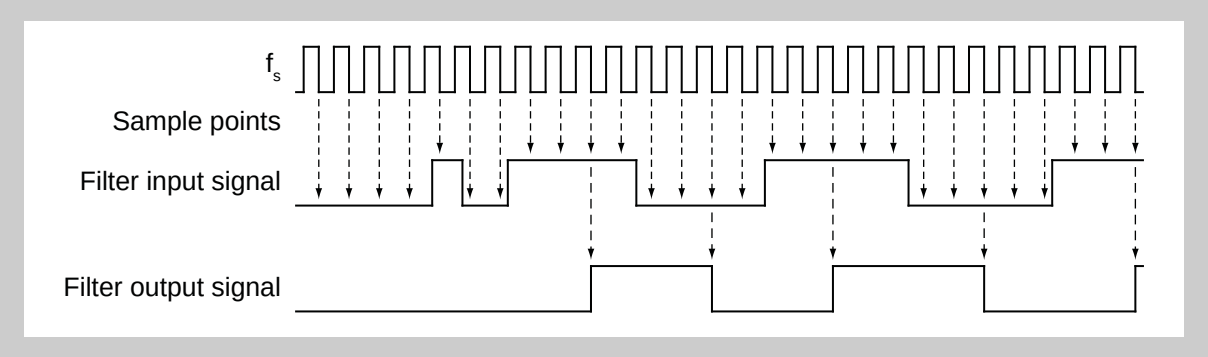



**Figure 2-6 Digital filter function**

**Digital filter groups**

The input signals with digital filters are ordered in groups of up to 16 signals. The digital filter characteristics, specified by DNFAnCTL.DNFAnPRS[2:0] and DNFAnNFSTS[1:0] apply to the filters of the entire group.
However the digital filter for each signal can be enabled respectively disabled separately by DNFAnEN.DNAFnNFENm.

**Cautions**

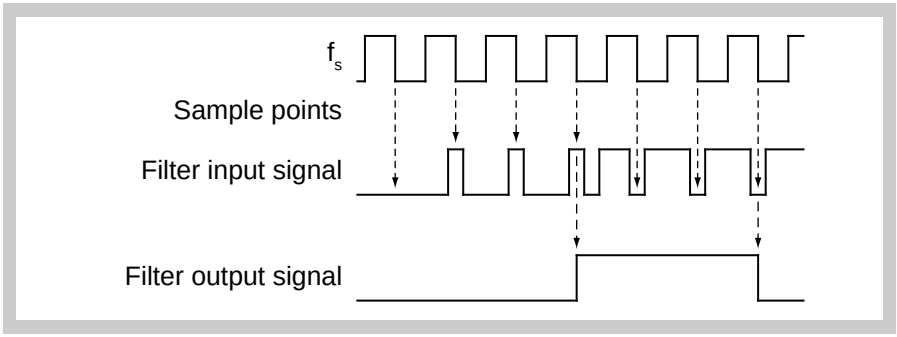
1. After enabling the digital filter by DNFAnEN.DNAFnNFENm = 1, the digital filter is in normal operation after the time period

   $$DNFAnNFSTS[1:0] \times 1/f_S + 2 \times 1/f_{DNFATCKI}$$

   and may generate unintended output signals during that time period.

   Wait the above time span before enabling the function, the signal is supplied to.

2. If the levels of the external signal changes multiple times within a sample period $1/f_S$ (undersampling), the digital filter may not detect these level changes and thus may not suppress such signals.
   An example is show in the figure below (for s = 3).



**Digital filter in stand-by mode**

- DEEPSTOP mode:
  All digital filter are located on an isolated power domain, that can be switched off during DEEPSTOP. Thus they are not in operation, when the respective power domain is in DEEPSTOP mode.
- STOP mode:
  Digital noise elimination requires the clock supply DNFATCKI to operate. Since the DNFATCKI clock is never stopped, digitally filtered signals can always serve as STOP mode wake-up event.

**Digital filters control registers**

For each group of up to 16 digital filters a common digital filter setup register DNFAnCTL and digital filter enable register DNFAnEN is provided with the same index n.

While the filter setup by DNFAnCTL effects the entire group, the control bits DNFAnENm in the filter enable register DNFAnEN allows to enable respectively disable each filter separately. The register index m is in the range from 0 to 15:

DNFAnCTL is the control register of group n for the digital filters m = 0 to 15, enabled/disabled by the DNFAnEN.DNAFnEN0 to DNFAnEN.DNAFnEN15 control bits.

The edge detection setup is done via the filter dedicated control register FCLAnCTLm.

The FCLAnCTLm registers are ordered in groups of 8 registers with the same index n. The register index m is in the range from 0 to 7:

FCLA group n: FCLAnCTL0 to FCLAnCTL7

The assignment of the input signals to the control registers and their addresses is given in the table *“Input signals with digital filters”* in the previous section *“Analog and digital filter assignments”*.

**Caution** Do not change any control register settings, while the concerned digital filter is enabled by DNFAnEN.DNAFnENm = 1. Otherwise an unintended filter output may be generated.

### (1) Digital filter type D

The block diagram of the digital filter type D is shown in the diagram below.

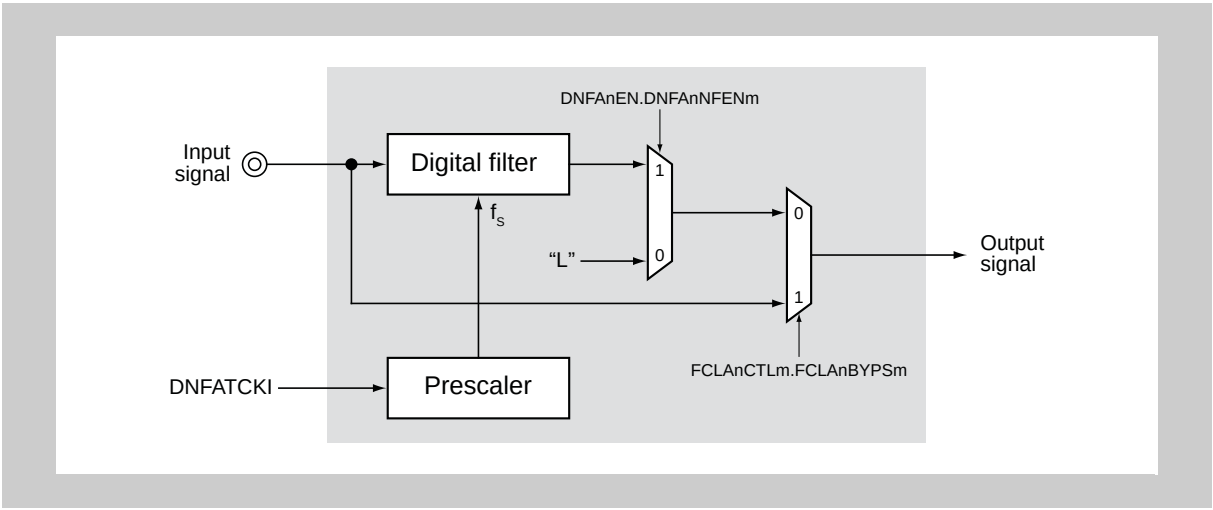



**Figure 2-7 Digital filter type D block diagram**

The output signal depends on register settings, as shown in the table below:

**Table 2-49 Output options of digital filter type D**

| FCLAnCTLm. FCLAnBYPSm | DNFAnEN. DNFAnNFENm | Output signal |
|---|---|---|
| 0 | 0 | fixed low level (input signal blocked) |
| | 1 | filtered input signal |
| 1 | X | not filtered input signal (filter bypass) |

**Default configuration**

The default configuration of the digital filter type D is as follows:

- input signal blocked

**Cautions**

1. Per default, the input signal is blocked. Thus the digital filter type D must be configured to let the - filtered or non-filtered - input signal pass.
2. If digital filtering shall not be applied to the input signal, the filter bypass must be activated by FCLAnCTLm.FCLAnBYPSm = 1.

### 2.6.3 Filter control registers

The analog and digital filters are controlled and operated by the following registers:

**Table 2-50 Filter registers overview**

| Register Name | Shortcut | Address |
|---|---|---|
| Filter control register m | FCLAnCTLm | The addresses are given in the tables in the previous section *"Port filters assignments"*. |
| Digital noise elimination control register | DNFAnCTL | |
| Digital noise elimination enable register | DNFAnEN | |

### (1) FCLAnCTLm – Filter control register

This register controls the analog and digital filter operation.

**Access** This register can be read/written in 8-bit units.

**Address** The assignment of the input signals to the FCLAnCTLm registers and their addresses is given in the tables in the previous section *"Port filters assignments"*.

**Initial Value** $00_H$. This register is initialized by any reset.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCLAn BYPSm | 0 | 0 | 0 | 0 | FCLAn INTLm | FCLAn INTFm | FCLAn INTRm |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 2-51 FCLAnCTLm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | FCLAn BYPSm | Filter bypass control:<br>0: filter bypass disabled<br>1: filter bypass enabled<br>**Note:**<br>This bit is only effective for analog filters type B and digital filters type D. |
| 2 | FCLAn INTLm | Level/edge detection mode selection<br>0: edge detection enabled<br>1: level detection enabled<br>**Note:**<br>This bit is only effective for analog filters type A. |
| 1 | FCLAn INTFm | • In level detection mode (FCLAnINTLm = 1): FCLAnINTFm has no effect<br>• In edge detection mode (FCLAnINTLm = 0): falling edge detection control<br>0: falling edge detection disabled<br>1: falling edge detection enabled<br>**Note:**<br>This bit is only effective for analog filters type A. |
| 0 | FCLAn INTRm | • In level detection mode (FCLAnINTLm = 1): detection level selection<br>0: low level detection<br>1: high level detection<br>• In edge detection mode (FCLAnINTLm = 0): rising edge detection control<br>0: rising edge detection disabled<br>1: rising edge detection enabled<br>**Note:**<br>This bit is only effective for analog filters type A. |

### (2) DNFAnCTL – Digital noise elimination control register

This register specifies the filter characteristics of the digital noise elimination filter.

**Note** This register is only effective for digital filters type D.

**Access** This register can be read/written in 8-bit units.

**Address** The assignment of the input signals to the DNFAnCTL registers and their addresses is given in the tables in the previous section *"Port filters assignments"*.

**Initial Value** $00_H$. This register is initialized by any reset.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | DNFAnNFSTS[1:0] | | 0 | 0 | DNFAnPRS[2:0] | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 2-52 DNFAnCTL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 6 to 5 | DNFAn NFSTS[1:0] | DNFAnNFSTS[1:0] specifies the number of same level samples, i.e. the number of samples with the same level to judge an external signal pulse as valid. |
| 2 to 0 | DNFAn PRS[2:0] | Digital filter sampling clock selection |

| DNFAnNFSTS[1:0] | Number of same level samples |
|---|---|
| $00_B$ | 2 samples |
| $01_B$ | 3 samples |
| $10_B$ | 4 samples |
| $11_B$ | 5 samples |

| DNFAnPRS[2:0] | Sampling clock frequency |
|---|---|
| $000_B$ | DNFATCKI / 1 |
| $001_B$ | DNFATCKI / 2 |
| $010_B$ | DNFATCKI / 4 |
| $011_B$ | DNFATCKI / 8 |
| $100_B$ | DNFATCKI / 16 |
| $101_B$ | DNFATCKI / 32 |
| $110_B$ | DNFATCKI / 64 |
| $111_B$ | DNFATCKI / 128 |

### (3) DNFAnEN – Digital noise elimination enable register

This register enables/disables the digital noise elimination for a certain input signal.

**Note** This register is only effective for digital filters type D.

**Access** This register can be read/written in 16-bit units.
The high byte DNFAnNFEN[15:8] and low byte DNFAnNFEN[7:0] can also be separately accessed in 8-bit units via the registers DNFAnENH.DNFAnNFEN[15:8] and DNFAnENL.DNFAnNFEN[7:0].

**Address** The assignment of the input signals to DNFAnEN registers and their addresses is given in the tables in the previous section *"Port filters assignments"*.

**Initial Value** $0000_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| DNFAn NFEN15 | DNFAn NFEN14 | DNFAn NFEN13 | DNFAn NFEN12 | DNFAn NFEN11 | DNFAn NFEN10 | DNFAn NFEN9 | DNFAn NFEN8 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DNFAn NFEN7 | DNFAn NFEN6 | DNFAn NFEN5 | DNFAn NFEN4 | DNFAn NFEN3 | DNFAn NFEN2 | DNFAn NFEN1 | DNFAn NFEN0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 2-53 DNFAnEN register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | DNFAn NFEN[15:0] | Digital noise elimination control<br>0: digital noise elimination disabled<br>1: digital noise elimination enabled |

# Chapter 3 CPU System Functions

This chapter describes the registers of the CPU, the operation modes, the address space and the memory areas.

## 3.1 Overview

The CPU is founded on Harvard architecture and it supports a RISC instruction set. Basic instructions can be executed in one clock period. Optimized five-stage pipelining is supported. This improves instruction execution speed.

In order to make the microcontroller ideal for use in digital control applications, a 32-bit hardware multiplier enables this CPU to support multiply instructions, saturated multiply instructions, bit operation instructions, etc.

**CPU**

- CPU
  - Core: V850E2
  - Architecture: V850E2-V3 Architecture Class
- Instruction execution times:

| Device | Minimum instruction execution time | Maximum CPU clock |
|---|---|---|
| V850E2/FF4-G | 15.7 ns | 64 MHz |
| V850E2/FG4-G | 15.7 ns | 64 MHz |

- 32 × 32 bits general registers
- 5 stage single issue pipeline
- Internal 32-bit architecture
- 4 GB linear address space for program and data

- Processor protection functions
  - Memory Protection Unit (MPU)
    Protection against illegal execution from or data manipulation of CPU memory areas (four unified protection areas)
  - System Register Protection (SRP)
    Protection against damage to the system registers by a non-trusted programs

**Instruction set**

- V850E2 instruction set (98 instructions) compatible to former V850 instruction sets plus additional powerful instructions for reduced code size and increasing execution speed
- Signed multiplication operations in 1 clock
  - 16 bits × 16 bits → 32 bits
  - 32 bits × 32 bits → 32 bits or 64 bits
  - 32 bits × 32 bits → 64 bits
- Saturated operation instructions with overflow/underflow detection
- 32-bit shift instructions in1 clock
- Bit manipulation instructions (bit set, clear, not, test)
- Load/store instructions with long/short format
- Load/store displacement extension
- Signed load instructions
- QDIV/QDIVU for quick divide
- CAXI instruction for Mutex
- MAC operation
  32 bit x 32 bit + 64 bit → 64 bit

## 3.2 Memory Protection Unit (MPU)

**Caution** DPA5 is always used in “base/mask specification mode”.

If the address of an accessed data is located below the configured upper boundary address of DPA5 and due to the length of the data, the access is crossing the upper boundary, an access violation will not be generated, although upper bytes of the unaligned access are located above the configured upper boundary of DPA5.

Therefore it is recommended to avoid any unaligned accesses, which could lead to crossing of the upper boundary of the DPA5 region.

Refer to V850E2 Architecture User’s Manual (<tbd>) for the DPA5U and DPA5L registers description.

## 3.3 CPU Access Bus Structures and Latencies

The CPU accesses the configuration, control and status registers of all functional modules of the microcontroller in different ways, depending on where they are located:

- modules of the CPU Subsystem:
  refer the section to *"CPU Subsystem modules access"* below
- modules externally to the CPU Subsystem (PBUS modules):
  refer the section to *"PBUS modules access"* below.

**Note** For a detailed description of the CPU Subsystem refer to the section *"CPU Subsystem"* below in this chapter.

### 3.3.1 CPU Subsystem modules access

For accessing the registers of modules on the CPU Subsystem the dedicated busses LSPB and GSPB are provided. Both are controlled only by the CPU. Refer also to the section *"CPU Subsystem"* in this chapter.

**Table 3-1 V850E2 CPU Subsystem control busses**

| Registers of module | CPU master bus |
|---|---|
| | rKB |
| Interrupt Controller (INTC) | R/W |
| DMA Controller (DMAC) | R/W |

**H/W lock function** The rKB bus supports the hardware lock function.
Thus bit manipulation instructions (CLR1, NOT1, SET1, TST1) can be applied to all registers accessed via the LSPB and GSPB bus, provided the register allows 8-bit access.

**Byte/halfword access** The rKB bus supports byte and halfword accesses.
Thus each byte or halfword of all registers accessed via the rKB bus can be accessed separately.

### 3.3.2 PBUS modules access

The CPU communicates with PBUS modules - these are not located on the CPU Subsystem - via the CPU Subsystem's PBUS master interfaces.

**H/W lock function** The PBUS master I/F does not support the hardware lock function.

**Byte/halfword access** All PBUS and HBUS modules are accessed on 32-bit word aligned addresses. Further each read or write is carried out with 32-bit width on the bus.

However the CPU can also perform byte (load/store byte) and halfword (load/store halfword) access instructions with the following results:

- Byte read accesses
  - The CPU instructions LD.B and SLD.B (load of signed byte data) loads the word aligned byte to bits [7:0] of a CPU general purpose register. The general purpose register bits [31:8] will be sign extended.
  - The CPU instructions LD.BU and SLD.BU (load of unsigned byte data) loads the word aligned byte to bits [7:0] of a CPU general purpose register. The general purpose register bits [31:8] will be zero-extended.
- Halfword read accesses
  - The CPU instructions LD.H and SLD.H (load of signed halfword data) loads the word aligned halfword to bits [15:0] of a CPU general purpose register. The general purpose register bits [31:16] will be sign extended.
  - The CPU instructions LD.HU and SLD.H loads the word aligned halfword to bits [15:0] of a CPU general purpose register. The general purpose register bits [31:16] will be zero-extended.
- Write access
  A byte (CPU instructions ST.B, SST.B) or halfword (CPU instructions ST.H, SST.H) write access corrupts the not written bytes of the word address.

**Caution** Do not perform byte or halfword write accesses, unless the write target is only 8 bit (for byte access) or 16 bit respectively (for halfword access) wide, so that the upper 3 bytes or 2 bytes respectively can not become corrupted.

**Bus clocks**

The CPU Subsystem is supplied with the CKSCLK_000 clock (CPUCLK). If the CPU (respectively the DMA Controller) accesses modules, external to the CPU Subsystem, these modules are members of different clock domains. Thus the bus clocks may have to be synchronized, when the CPU accesses the concerned modules.

This section explains the bus structures and its clocks and defines access latencies, when different bus clocks have to be synchronized.

**Bus clocks notation**

The bus clocks PCLK have an index, that indicates the clock, the bus is operating with. For instance, $PCLK_{000}$ denotes a PBUS with the PCLK clock synchronous to CKSCLK_000, or in general:

- $PCLK_{mn}$ is a PBUS clock synchronous to CKSCLK_mn

**Caution**

Since access to modules on clock domains with bus synchronizers may induce latencies of several cycles of the CPUCLK, the flow of the CPU program may become inconsistent.
That means a CPU instruction, that accesses CPU Subsystem modules (in particular INTC, DMAC), follows an instruction, that accesses a module with bus synchronizer latencies, the access to the CPU Subsystem module may become effective before the first one (module access via bus synchronizer).

Refer to the further descriptions in this section for details.

The following diagram shows the bus structure for CPU accesses to PBUS modules.

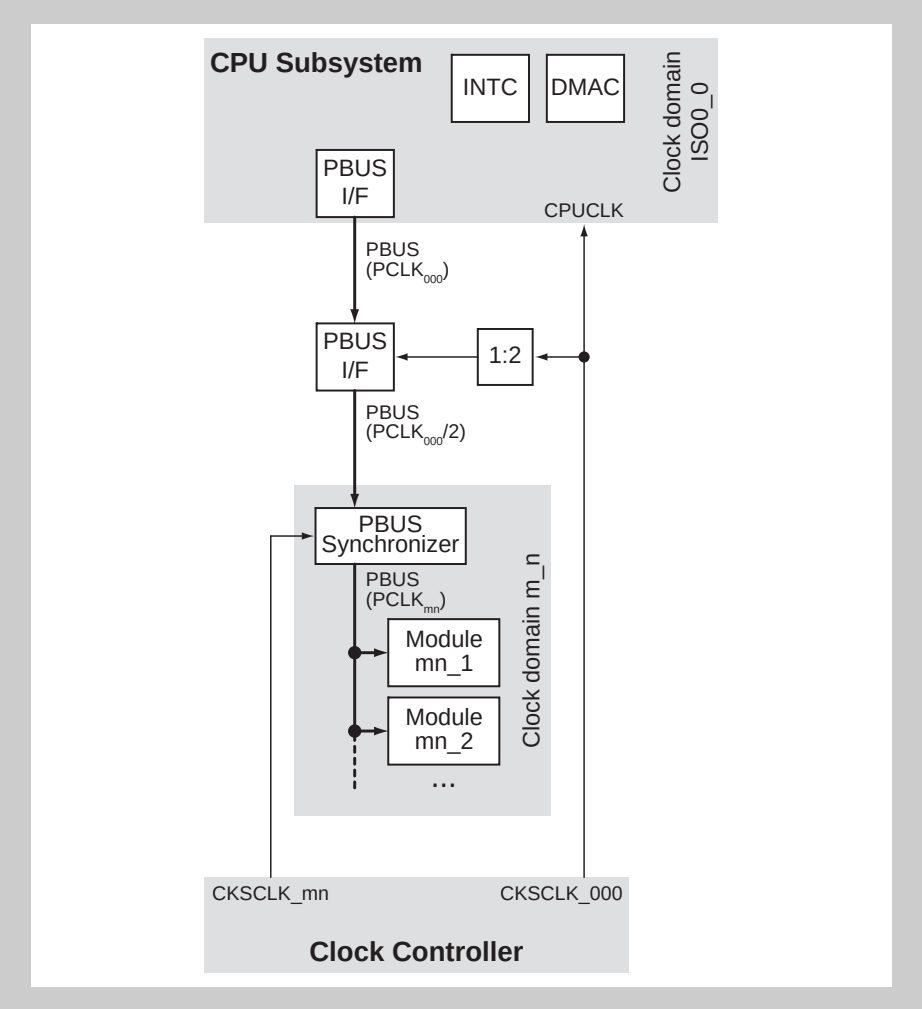



**Figure 3-1 CPU access bus structures**

The CPU accesses all modules, except those of the CPU Subsystem, via their PBUS interface with the $PCLK_{000}/2$ clock.

**CPU Subsystem modules**

The CPU accesses modules of the CPU Subsystem (DMA Controller, Interrupt Controller) via CPU Subsystem internal busses. The bus clock of these busses is CPUCLK (CKSCLK_000). Thus no bus synchronization is required.

**Clock domains m_n**

All PBUS interfaces of the clock domains m_n, i.e. AWO_n and ISO0_n are supplied with clocks from different clock domains CKSCLK_mn.

Thus CPU accesses with $PCLK_{000}/2$ = CKSCLK_000/2 to the clock domain's m_n PBUS interfaces are synchronized to the target clock domain CKSCLK_mn.
Refer to the section *"PBUS Synchronizer"* below for information about the PBUS synchronizers latencies.

### 3.3.3 PBUS Synchronizer

The PBUS synchronizers synchronize the bus clock $PCLK_{000}/2$ from the CPU to the target clock domain's PBUS clock $PCLK_{mn}$.

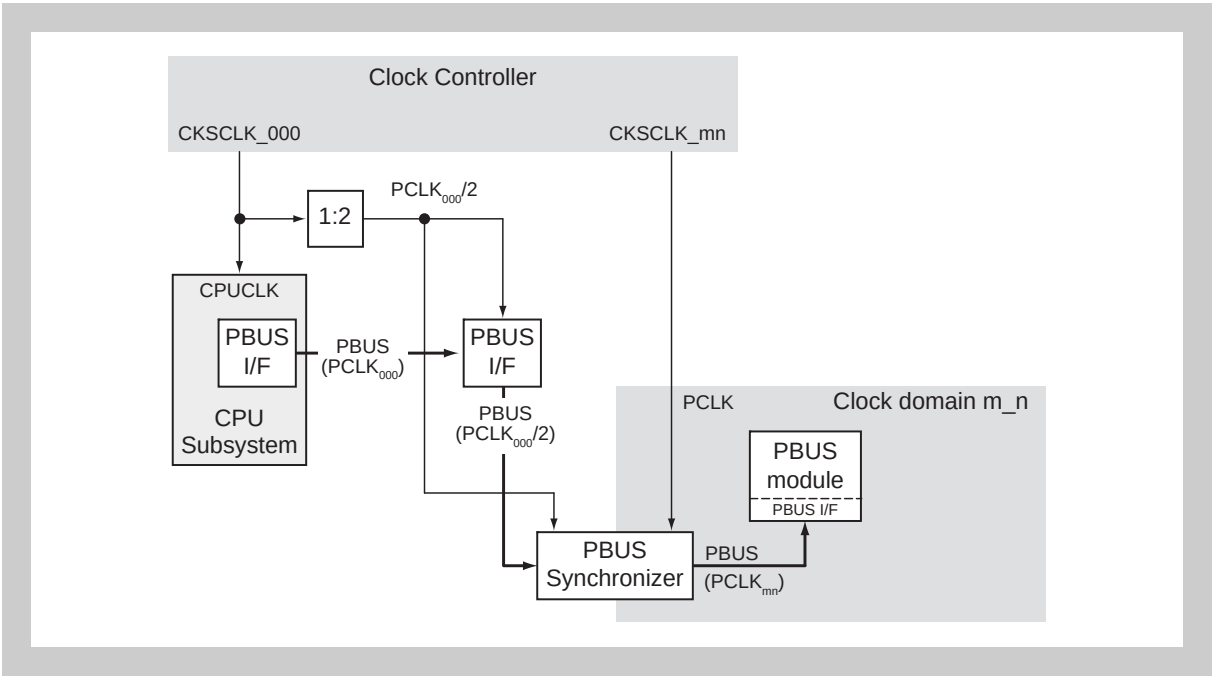



**Figure 3-2 PBUS synchronizer**

The synchronizers generate latencies of CPU accesses to the PBUS modules on the clock domain m_n.

The maximum latency of a clock domain m_n PBUS access is calculated by the following formula:

$$\text{Latency cycles} = 4 \times PCLK_{000}/2 + 4 \times PCLK_{mn}$$

**Notes**

1. The synchronization latency always applies, even if both clocks are configured to use same clock source or frequency.
2. The PBUS synchronizers are shown in the clock domain figures in section *"Clock Domain Figures"* in the chapter *"Clock Controller"*.

**(1) Overall PBUS latencies**

The overall access latency of a CPU access to a PBUS module is calculated as follows:

- Read access:

$$T_{accRD} = 16 \cdot PCLK_{000} + 6 \cdot PCLK_{mn} + W_{RD} \cdot PCLK_{mn}$$

- Write access:

$$T_{accWR} = 6 \cdot PCLK_{000} + 6 \cdot PCLK_{mn} + W_{WR} \cdot PCLK_{mn}$$

$W_{RD}$ and $W_{WR}$ specify the number of wait clocks, induced by the PBUS module upon read respectively write access. Refer to the section *"Module wait clocks insertion"* below for the $W_{RD}$ and $W_{WR}$ values of the different modules.

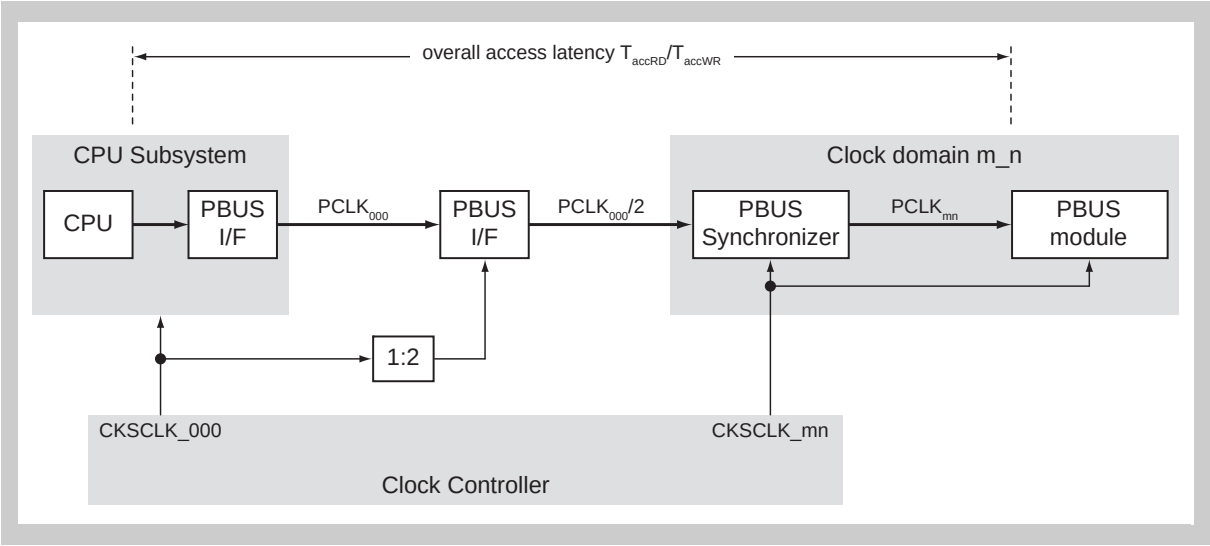



**Figure 3-3 Overall PBUS access latency**

**CPU write access**

If the PBUS is not occupied, i.e. the PBUS is idle, the write access is immediately forwarded and the CPU continues with the next instruction.

If the PBUS is occupied because of a former CPU write or a DMA access, the CPU is stopped until the PBUS is idle.

**CPU read access**

Upon a CPU read access to any PBUS module, CPU operation stops until the requested data is available.

### (2) Module wait clocks insertion

The following modules insert wait states when accessed by the CPU.

**Note** All other module registers, which are not listed in the following table, do not inserted any wait clocks ($W_{RD}$ = $W_{WR}$ = 0).

**Table 3-2 Module registers access wait clocks**

| Module | Registers | Read wait clocks $W_{RD}$ | Write wait clocks $W_{WR}$ |
|---|---|---|---|
| CAN Controller (FCN) | FCNnMmDAT[7:0]B<br>FCNnMmDTLGB<br>FCNnMmSTRB<br>FCNnDNBMRX[3:0]<br>FCNnMmCTL<br>FCNnCMLISTR<br>FCNnCMLOSTR | 2 PCLK | 2 PCLK |
| | FCNnCMRGRX<br>FCNnCMTGTX<br>FCNnMmDAT[6,4,2,0]H<br>FCNnMmMID0H<br>FCNnMmMID1H | 2 PCLK | 1 PCLK |
| | FCNnMmDAT[4,0]W<br>FCNnMmMID0W | 3 PCLK | 2 PCLK |
| | FCNnCMCLCTL.FCNnCMCLSERC | n.a. (write only) | 1 to 10 PCLK |
| | FCNnCMCLCTL<br>except FCNnCMCLSERC bit | 0 PCLK | 3 PCLK |

## 3.4 CPU Subsystem

This section gives an overview about the CPU Subsystems.

The CPU Subsystems comprise

- the CPU and CPU dedicated components, e.g. the processor protection functions
- busses to instruction and data memory
- various interfaces to the other microcontroller modules, e.g. the PBUS and data flash interface
- the Interrupt Controller (INTC)
- the DMA Controller (DMAC)
- On-chip Debug unit
- several bus systems with bus arbiters that allow access of the bus masters to all other modules

### 3.4.1 Power and clock domain

The CPU Subsystem lies on the Isolated-Area-0 and its clock CPUCLK is supplied from the domain clock CKSCLK_000.

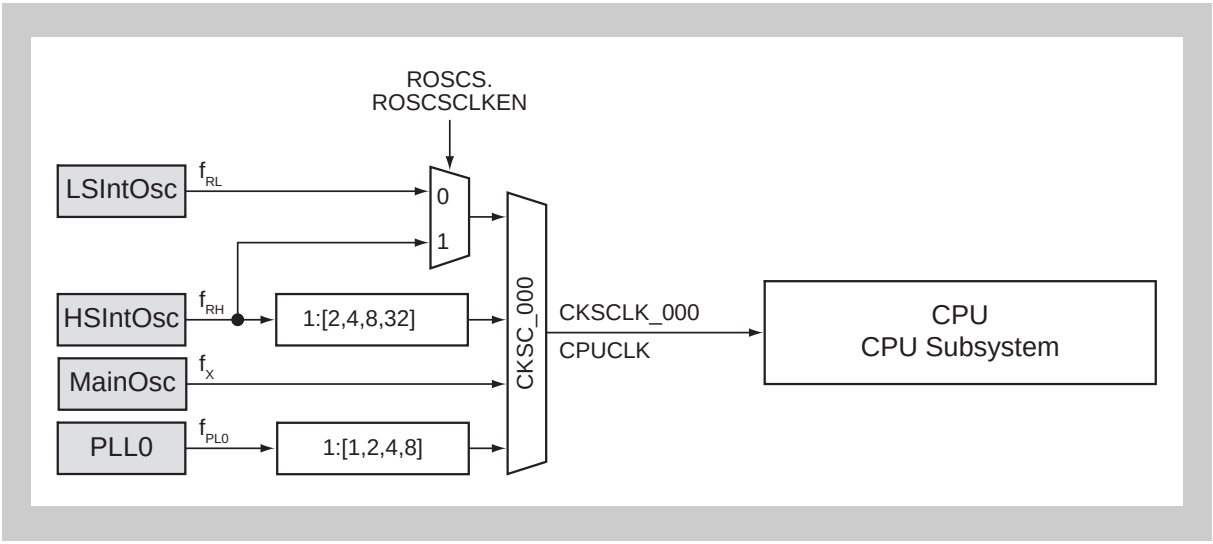



**Figure 3-4 CPU Subsystem clock supply**

**CPU Subsystem H/W reset**

The CPU Subsystem and its registers are initialized by the following reset signal:

**Table 3-3 CPU Subsystem reset signal**

| CPU Subsystem | Reset signal |
|---|---|
| CPU Subsystem | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

### 3.4.2 CPU Subsystem busses overview

The table below gives an overview about the CPU Subsystem busses and their general purpose.

Refer to the following section for detailed information about the CPU Subsystems.

**Table 3-4 Bus systems**

| CPU Subsystem | Bus | Purpose | Bus master |
|---|---|---|---|
| internal | CPU instruction bus | CPU access to<br>• code flash<br>• data RAM<br>• PBUS modules<br>• Data flash | CPU |
| | CPU data bus | CPU access to<br>• code flash<br>• data RAM<br>• PBUS modules<br>• Data flash | CPU |
| | rKB register bus | Access to registers of<br>• Interrupt Controller INTC<br>• DMA Controller DMAC | CPU |
| | DMA data bus | DMAC access to<br>• code flash<br>• data RAM<br>• PBUS modules<br>• Data flash | DMAC |
| external | PBUS | Access to all PBUS modules via the PBUS master I/F | CPU, DMAC[a] |

a) Refer to the following sections for more information about the buses arbitration.

### 3.4.3 CPU Subsystem

The diagram below shows a CPU Subsystem block diagram of the V850E2/Fx4-G product series.

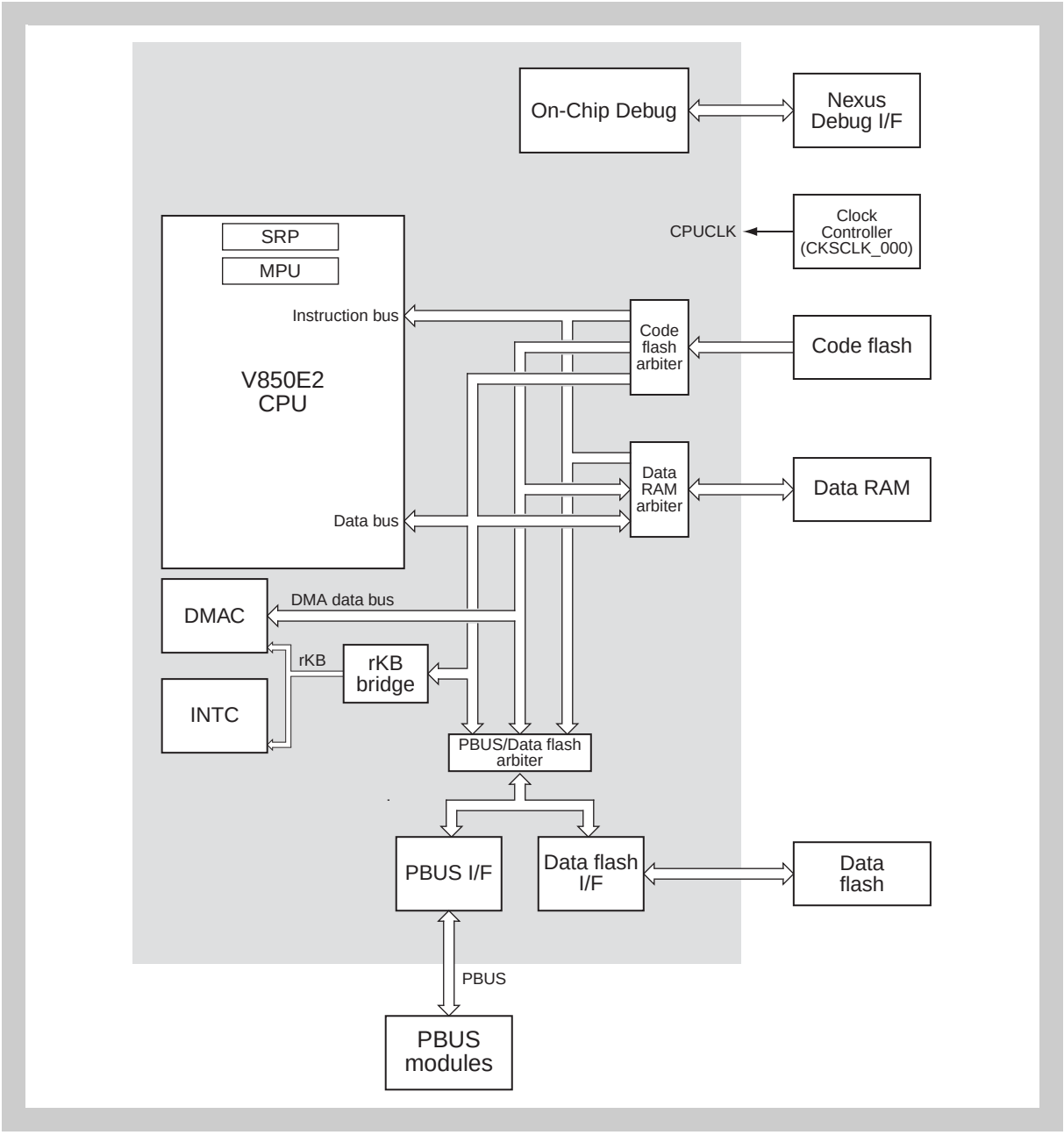



**Figure 3-5 CPU Subsystem**

All busses for transferring data between the various modules are controlled by two masters:

- CPU
- DMA Controller (DMAC)

**Table 3-5 CPU Subsystem data/instruction busses**

| Master | Bus | Code flash | Data RAM | Data flash | PBUS I/F |
|---|---|---|---|---|---|
| CPU | Instruction (32 bit) | R | R | – | – |
| | Data (32 bit) | R | R/W | R/W | R/W |
| DMA Controller | DMA data (32 bit) | R | R/W | R | R/W |

### (1) Code flash access

**Table 3-6 Code flash access**

| | | |
|---|---|---|
| Code flash | Code flash arbiter | ← CPU instruction bus |
| | | ← CPU data bus |
| | | ← DMA data bus |

### (2) Data RAM access

**Table 3-7 Data RAM access**

| | | |
|---|---|---|
| Data RAM | Data RAM arbiter | ← CPU instruction bus |
| | | ← CPU data bus |
| | | ← DMA data bus |

**Caution** Before fetching any instruction code from the data RAM, make sure to initialize the 16-byte boundary area of the data RAM that contains the instruction code to fetch.

A 16-byte boundary area is an area from the address XXXX $XXX0_H$ to XXXX $XXXF_H$.

For initializing the data RAM, any data values can be written, but the data RAM area must be initialized before an instruction is fetched from it.

If an instruction is fetched from an uninitialized data RAM area, a memory protection exception (MEP) might occur.

**Note** In general, it is recommended to initialize all the data RAM before reading it.

#### (3) Data flash I/F access

Table 3-8 Data flash I/F access

| | | |
|---|---|---|
| Data flash I/F | PBUS/data flash arbiter | ← CPU instruction bus |
| | | ← CPU data bus |
| | | ← DMA data bus |

**Data flash wait cycles**

Concerning data flash access refer also to the section *"Data flash wait cycle control"* below in this chapter.

#### (4) CPU Subsystem busses arbitration policies

The following tables specify the arbitration policies of the CPU Subsystem busses arbiters.

Table 3-9 CPU Subsytem busses arbitration policies

| Arbiter | Policy | Masters |
|---|---|---|
| Code flash | Fixed priority | High: DMA |
| | | Medium: CPU data bus |
| | | Low: CPU instruction bus |
| Data RAM | Fixed priority | High: DMA |
| | | Medium: CPU data bus |
| | | Low: CPU instruction bus |
| Data flash | Fixed priority | High: DMA |
| | | Medium: CPU data bus |
| | | Low: CPU instruction bus |

### 3.4.4 V850E2 system manual

Detailed descriptions of the various CPU Subsystem functions can be found in the following documents:

Table 3-10 CPU Subsystem functions information sources

| Function | V850E2S Architecture <tbd> | This manual (R01UH0366EJ0100) |
|---|---|---|
| V850E2 CPU (including instruction set) | √ | – |
| Processor protection functions (MPU, SPR) | √ | – |
| DMA Controller (DMAC) | – | √ |
| Interrupt Controller (INTC) | – | √ |
| Code/Data flash | – | √ |
| Data RAM | – | √ |

## 3.5 Data flash wait cycle control

CPU read accesses to the data flash require insertion of some wait cycles to access the flash memory.

The minimum number of wait cycles, in units of CPU Subsystem clock cycles CPUCLK, has to be selected via the wait cycle control register DCLKWAIT.

**Caution** The data flash wait cycle control register DCLKWAIT has to be set before the first CPU access to the data flash.

### (1) DCLKWAIT – data flash wait cycle control register

This register is used to configure the number of wait cycles for accessing the data flash.

**Access** This register can be read/written in 8-bit units.

**Address** FF43 6000$_H$

**Initial Value** 1F$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | WAIT[4:0] | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 3-11 DCLKWAIT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 4 to 0 | WAIT[4:0] | Sets the number of wait cycles, as shown in the table below:<br>11$_H$: 17 wait cycles<br>1F$_H$: 31 wait cycles<br>All other settings are prohibited. |

## 3.6 Operation modes

This section describes the operation modes of the V850E2/Fx4-G and how the modes are selected.

The following operation modes are available:

- Normal operation mode
- Serial flash programming mode

After release of the Power-On-Clear reset or external $\overline{\text{RESET}}$ the microcontroller starts to fetch instructions from an internal boot ROM which contains the internal firmware. The firmware checks the FLMD0 pin, and optionally also the FLMD1 pin, to set the operation mode after reset release according to the table below.

**Table 3-12 Selection of operation modes**

| Pins | | Operation mode |
|---|---|---|
| FLMD0 | FLMD1 (P0_1) | |
| VSS | VSS | Normal operation mode |
| | VDD | Setting prohibited |
| VDD | VSS | Serial flash programming mode |
| | VDD | Setting prohibited |

For starting the microcontroller in a certain operation mode properly the mode pins and the ports of the JP0 port group have to be set up correctly, as described in the section *"Mode pins and JP0 connections"* below.

### 3.6.1 Normal operation mode

In normal operation mode, the internal flash memory is not re-programmed.

After reset release, the firmware branches to the start of the active boot swap cluster. User program execution is started.

**Debug mode** Debug mode is basically a normal operation mode, but the debugger changes the device into debug mode via $\overline{\text{DCUTRST}}$ upon reset release.

### 3.6.2 Flash programming mode

In serial flash programming mode, the internal flash memory is erased and reprogrammed.

After reset release, the firmware initiates loading of the user's program code from the external flash programmer and programs the flash memory.

After detaching the external flash programmer, the microcontroller can be started up with the new user's program in normal operation mode.

For more information see chapter *"Flash Memory"*.

## 3.7 Mode pins and JP0 connections

This section describes the wiring of the

- port group JP0 (JP0_0 to JP0_5)
- mode signals FLMD0, FLMD1 (P0_1)

in different microcontroller modes.

The function of the above mentioned pins depends on the mode:

**Table 3-13 JP0 and mode pin functions**

| Pin/port | Operation mode | | | |
|---|---|---|---|---|
| | Normal operation | Debugger | Flash programming | |
| | | | Asynchronous | Synchronous |
| JP0_0 | I/O port | DCUTDI | FLUR0RTX | FLCS0SI |
| JP0_1 | I/O port | DCUTDO | I/O port | FLCS0SO |
| JP0_2 | I/O port | DCUTCK | I/O port | FLCS0SCI |
| JP0_3 | I/O port | DCUTMS | I/O port | I/O port |
| JP0_4 | I/O port | $\overline{\text{DCUTRST}}$ | I/O port | I/O port |
| JP0_5 | I/O port | $\overline{\text{DCUTRDY}}$ | I/O port | I/O port |
| FLMD0 | FLMD0 | FLMD0 | FLMD0 | FLMD0 |
| P0_1 | FLMD1/I/O port[a] | FLMD1/I/O port[a] | FLMD1/I/O port[a] | FLMD1/I/O port[a] |

a) FLMD1 is used for mode setting, when the reset is released. Afterwards P0_1 can be used for general port and alternative functions.

**Notes**

1. FLMD0 has no port functionality and is exclusively used for mode setting.
2. Usage as I/O port means, the port can be used in port input/output and alternative modes.

**Pull-up/pull-down resistors**

The size of the resistors, shown in the following descriptions, depends on several parameters like current flow through the resistors in active level, secure level detection with maximum specified leakage current and maximum drive level of the driving port. Refer to the Data Sheet for details.

### 3.7.1 Normal operation mode

In normal operation mode the JP0 port group can be used for general port and alternative functions.

Normal operation mode is selected by

| FLMD0 | FLMD1 (P0_1)[a] |
|---|---|
| low level | low level |

a) FLMD1 can be used for general port and alternative functions after reset release.

FLMD0 has to be low level and therefore needs to be connected to GND via the resistor R1.

FLMD1 and $\overline{\text{DCUTRST}}$ have to be low level at reset release and therefore need to be connected to GND via the resistors R2. Afterwards P0_1 and JP0_4 can be used for general port and alternative functions.

The target value of R1 is 82 kΩ. For the exact dimensioning of the FLMD0 pull-down resistor R1 refer to the Data Sheet.

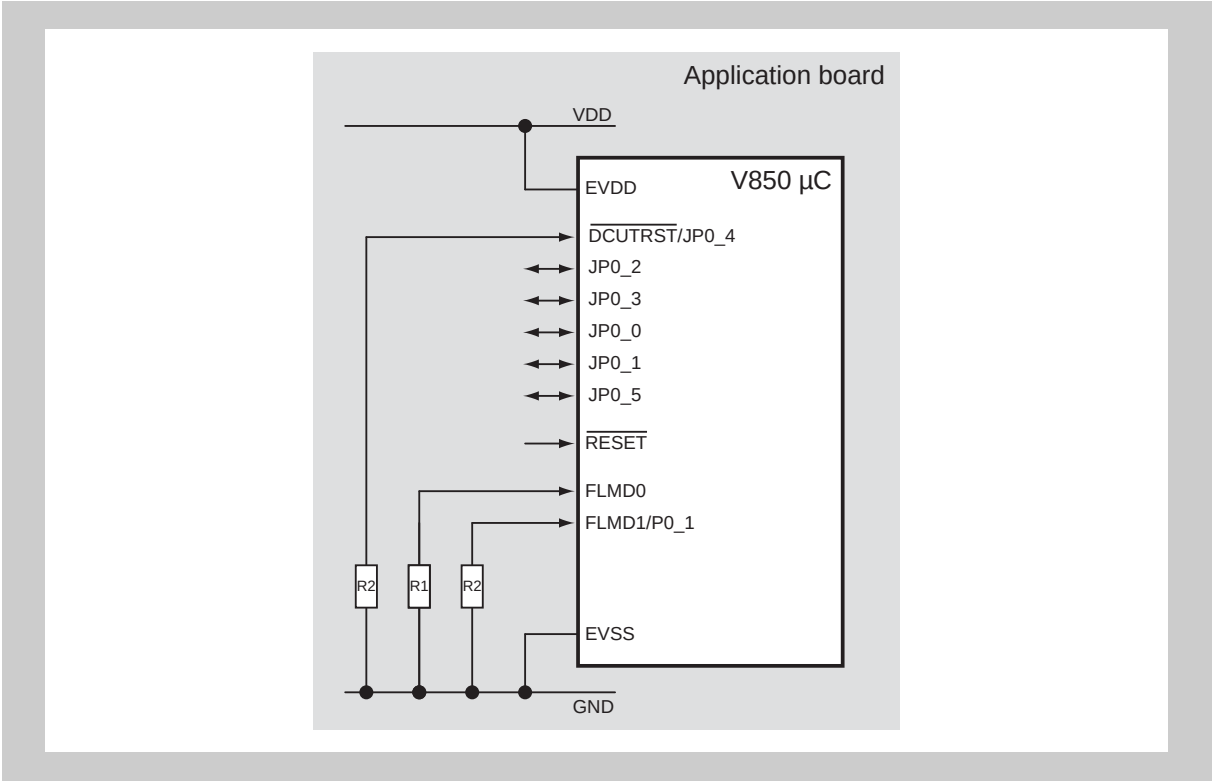



**Figure 3-6 Pin connections in normal operation mode**

### 3.7.2 Debug mode

In debug mode the debugger is connected to the pins of the port group JP0.

Debug mode is basically a normal operation mode, but the debugger changes the device into debug mode via DCUTRST upon reset release.

Debug mode is selected by

| FLMD0 | FLMD1 (P0_1)[a] |
|---|---|
| low level | low level |

a) FLMD1 can be used for general port and alternative functions after reset release.

**TDO, TRDY pull-up resistors**

Since the microcontroller's DCUTDO and DCUTRDY outputs are in high-impedance state during RESET, the R3 resistors maintain a high level at the On-chip Debugger's TDO and TRDY inputs during reset time.
The pull-up resistors R3 at TDO and TRDY may be recommended to be mounted on the debugger adapter board or cable in order to avoid temporarily undefined levels of TDO and TRDY. Refer to the debugger's description for details.
The R3 resistors are not required in normal operation.

**Notes**

1. In debug mode on-chip pull-up resistors are automatically connected to DCUTCK, DCUTMS and DCUTDI.
2. In debug mode all ports of the debugger interface are automatically configured and are not affected by any port register settings.

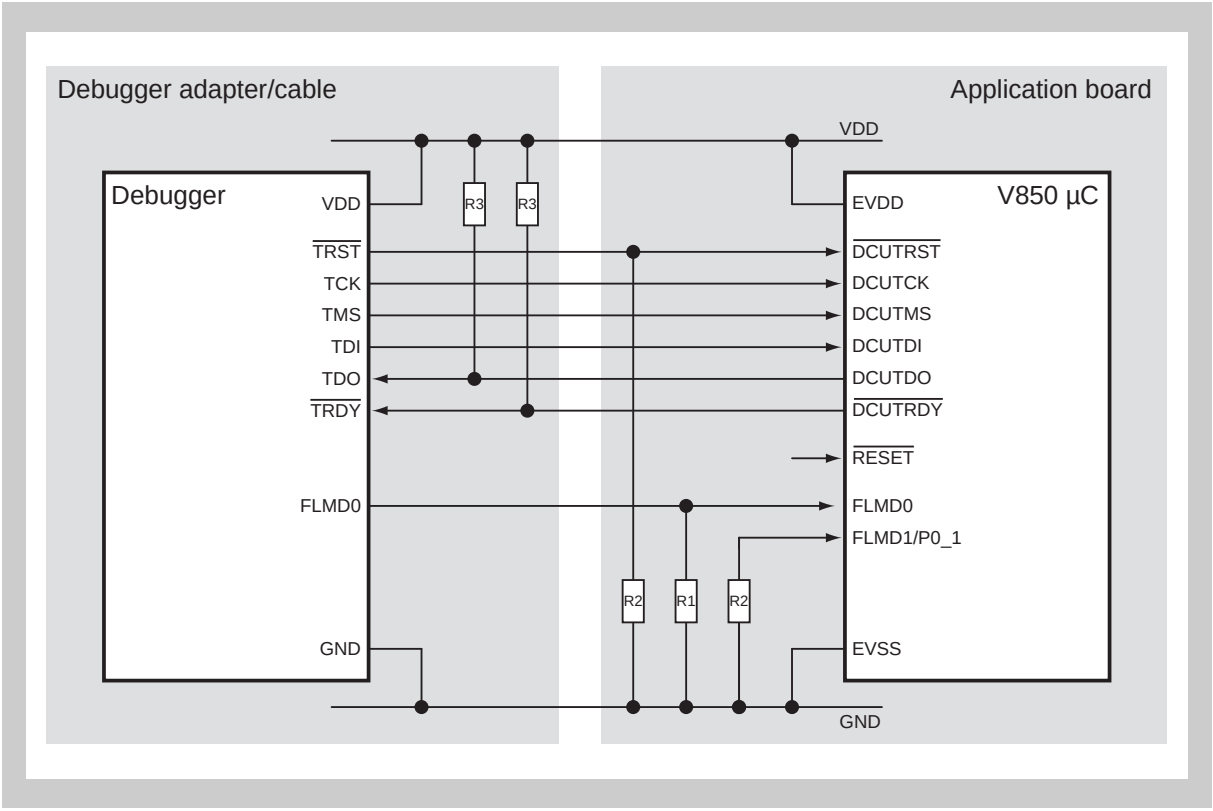



**Figure 3-7 Pin connections in debug mode**

**Further information** For further information about the debugger refer to the chapter *“On-Chip Debug Unit (OCD)”* and the debugger documentation

“User’s Manual QB-V850MINI, QB-V850MINIL”,

document number U17638EJxVxUM00 (xVx denotes the current version number).

### 3.7.3 Flash programming mode

In flash programming mode some ports of port group JP0 are connected to the flash programmer.

Flash programming mode is selected by

| FLMD0 | FLMD1 (P0_1) |
|---|---|
| high level | low level |

The PG-FP5 flash programmer communicates with the V850 microcontroller via

- a single wire asynchronous interface FLUR0RTX
- a 3-wire synchronous interface FLCS0SCI, FLCS0SI, FLCS0SO
  Ports JP0_3 to JP0_5 can be used for general port and alternative functions.

Flash programming mode is determined by FLMD0 at high and FLMD1 at low level. Both can be connected to the PG-FP5 flash programmer, that sets the pins to the correct levels. However FLMD1 connection to the flash programmer is not mandatory since FLMD1 is already set to low via the pull-down resistor R2.

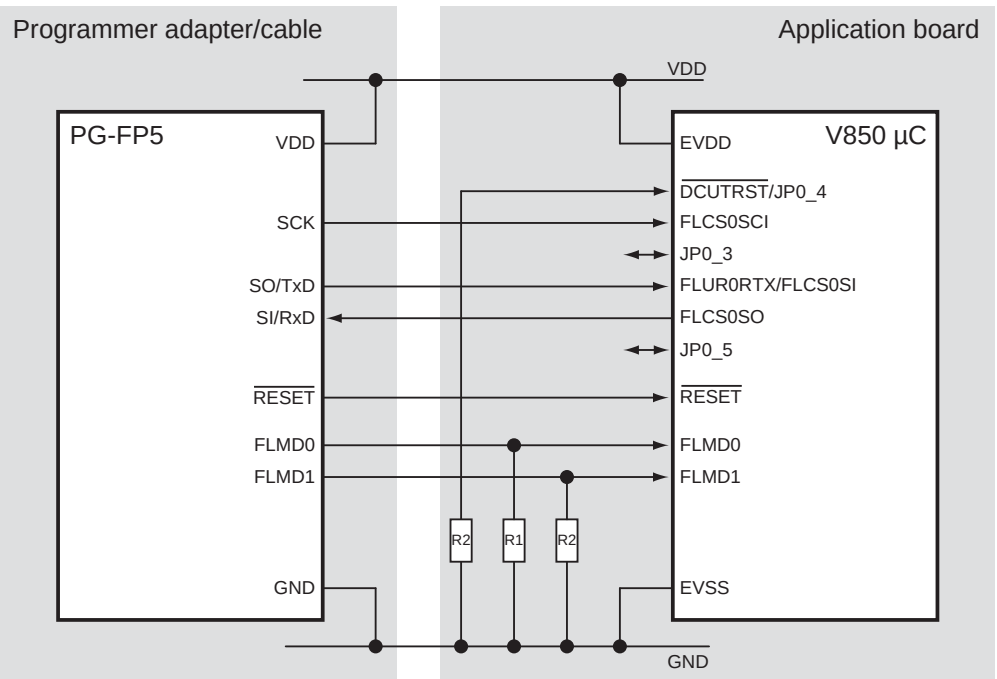



**Figure 3-8 Pin connections in flash programming mode (FLMD1 by PG-FP5)**

**Further information** For further information about the debugger refer to the chapter *"Flash Programming with Flash Programmer"* in the chapter *"Flash Memory"* and the PG-FP5 flash programmer documentation

"User's Manual PG-FP5",

document number R20UT0008EJxxxx (xxxx denotes the current version number).

## 3.8 Address Space

In the following sections, the address space of the CPU is explained. Size and addresses of CPU address space and physical address space are explained. The address range of data space and program space together with their wrap-around properties are presented.

### 3.8.1 CPU data address and physical program address space

The CPU supports the following address space:

- 4 GB CPU data address space
  With the 32-bit general purpose registers, addresses for a 4 GB memory can be generated. This is the maximum address space supported by the CPU.
- 64 MB physical program address space
  The CPU provides 64 MB physical address space to access instruction opcodes in the program memory. This means that a maximum of 64 MB internal or external program memory can be accessed.

### 3.8.2 Program and data space

The figure below shows the assignment of the CPU address space to data and program space.

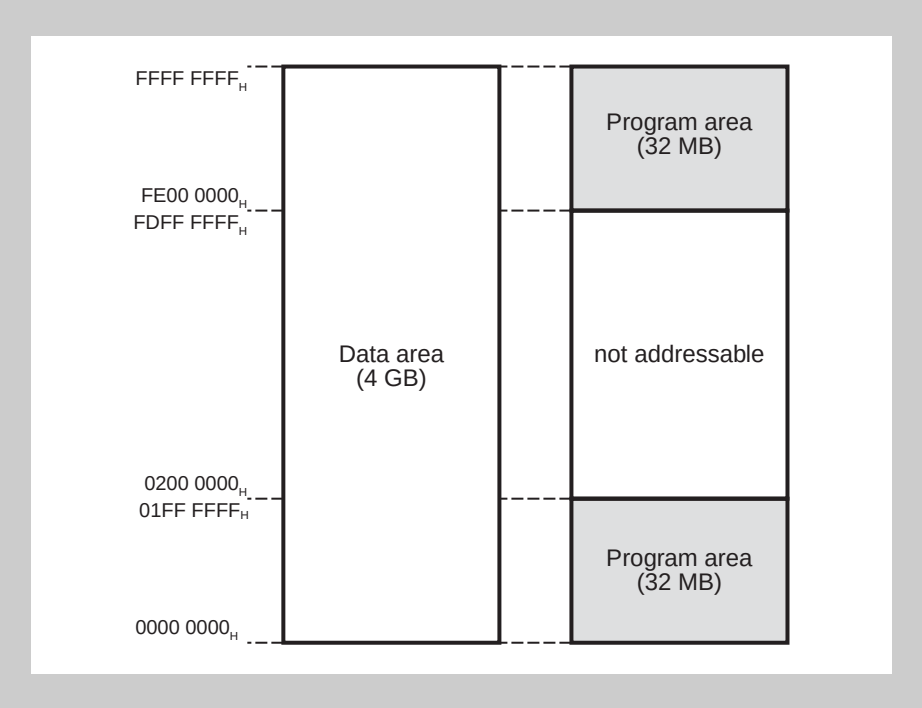



**Figure 3-9 CPU address space**

**(1) Wrap-around of data space**

If an operand address calculation exceeds 32 bits, only the lower 32 bits of the result are considered. Therefore, the addresses 0000 $0000_H$ and FFFF $FFFF_H$ are contiguous addresses. This results in a wrap-around of the data space:

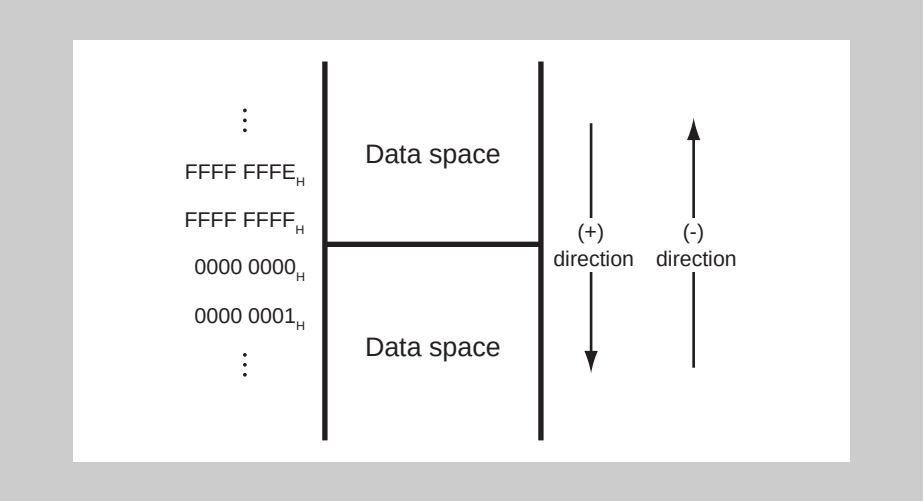



**Figure 3-10 Wrap-around of data space**

**(2) Wrap-around of program space**

If an instruction address calculation exceeds 25 bits, only the lower 25 bits of the result are considered. Therefore, the addresses 0000 $0000_H$ and 01FF $FFFF_H$ are contiguous addresses. This results in a wrap-around of the program space:

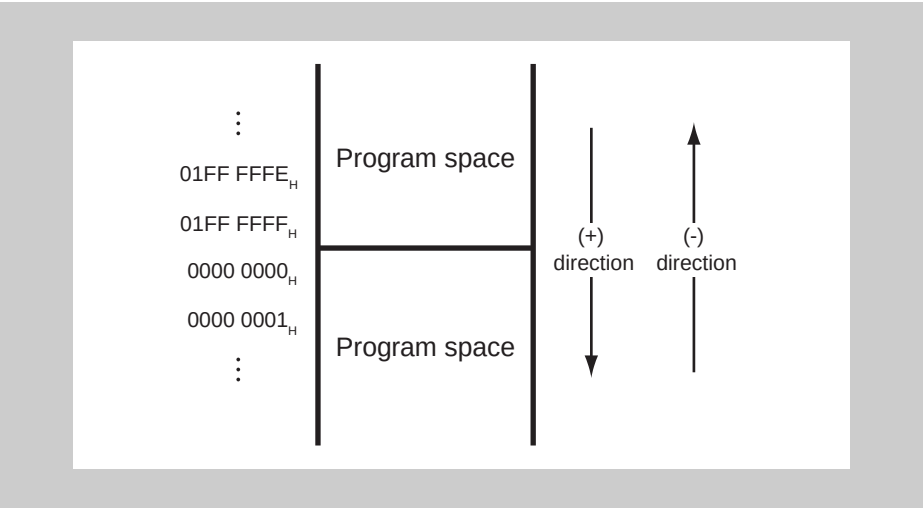



**Figure 3-11 Wrap-around of program space**

# 3.9 V850E2/Fx4-G CPU Address Map

In the following sections, the address map of the CPU and DMA is introduced. Specific memory areas are described in detail.

## 3.9.1 DMA address map

The DMA Controller can access all CPU address areas, except the address range

$FFFF\ 6000_H$ to $FFFF\ 7FFF_H$

This area can only be accessed by the CPU.

### 3.9.2 Memory map

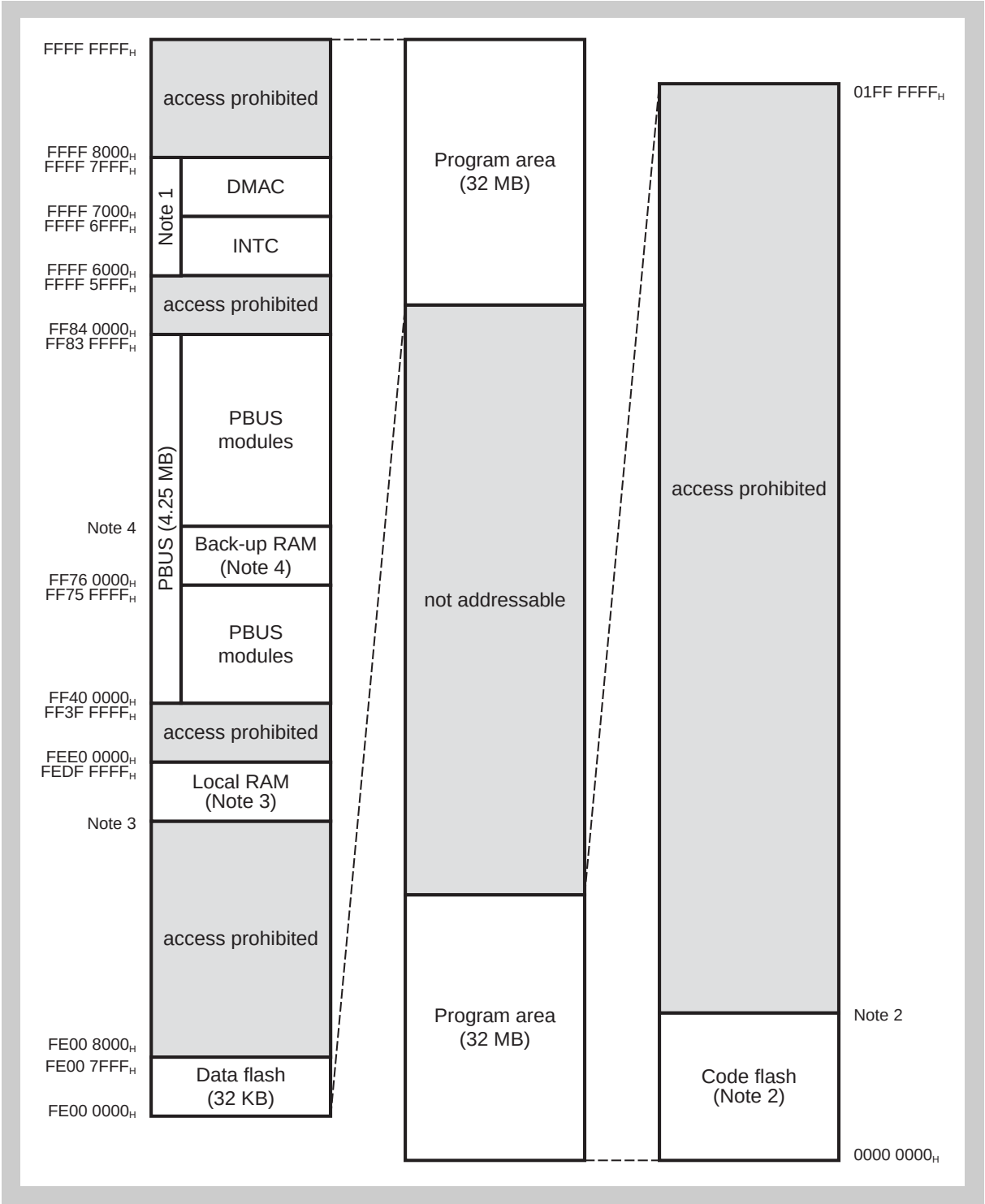



**Figure 3-12 Memory map**

**Notes** 1. These areas are not accessible by the DMA Controller.

2. The code flash size, and thus the address range, is device dependent:

| Series name | Product name | Size | Address range |
|---|---|---|---|
| FF4-G-256K | µPD70F4177 | 256 KB | 0000 $0000_H$ - 0003 $FFFF_H$ |
| FG4-G-256K | µPD70F4179 | | |
| FF4-G-512K | µPD70F4178 | 512 KB | 0000 $0000_H$ - 0007 $FFFF_H$ |
| FG4-G-512K | µPD70F4180 | | |

3. The Data RAM size, and thus the address range, is device dependent:

| Series name | Product name | Size | Address range |
|---|---|---|---|
| FF4-G-256K | µPD70F4177 | 32 KB | FEDF $8000_H$ - FEDF $FFFF_H$ |
| FG4-G-256K | µPD70F4179 | | |
| FF4-G-512K | µPD70F4178 | 64 KB | FEDF $0000_H$ - FEDF $FFFF_H$ |
| FG4-G-512K | µPD70F4180 | | |

4. The Back-up RAM size, and thus the address range, is device dependent:

| Series name | Product name | Size | Address range |
|---|---|---|---|
| FF4-G-256K | µPD70F4177 | 4 KB | FF76 $0000_H$ - FF76 $0FFF_H$ |
| FG4-G-256K | µPD70F4179 | | |
| FF4-G-512K | µPD70F4178 | | |
| FG4-G-512K | µPD70F4180 | | |

## 3.10 Back-up RAM (BURAM)

The Back-up RAMs are PBUS module.

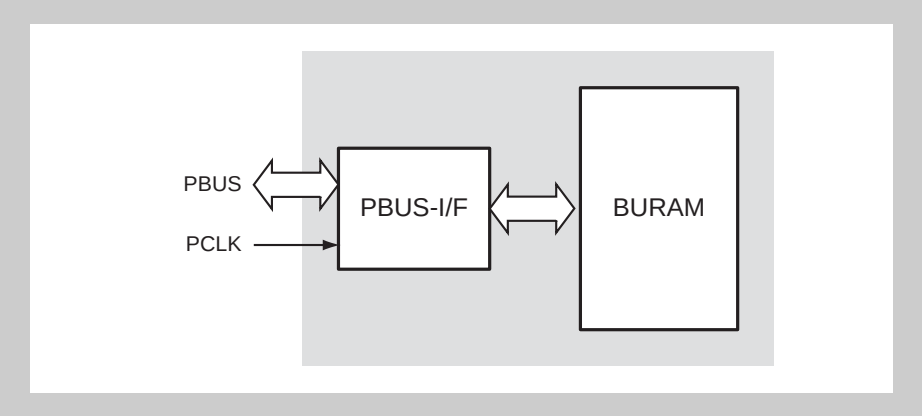



**Figure 3-13 Block diagram of the Back-up RAM**

**Access** This BURAM can be read/written in 32-bit units.

**BURAM addresses** The BURAM addresses are listed in the following table:

**Table 3-14 BURAM address**

| Device | Size | Address |
|---|---|---|
| Fx4-G-256K<br>Fx4-G-512K | 1K × 32 bit | FF76 $0000_H$ - FF76 $0FFF_H$ |

**Clock supply** The BURAM is supplied with the following clock.

**Table 3-15 BURAM clock supply**

| BURAM | BURAM clock | Connected to |
|---|---|---|
| BURAM | PCLK | Clock Controller CKSCLK_A05 |

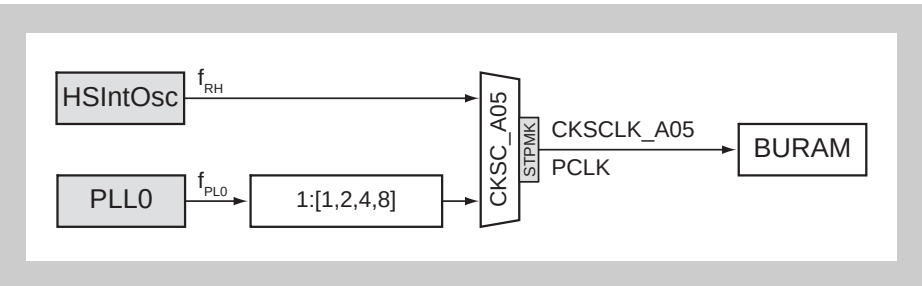



**Figure 3-14 BURAM clock supply**

**Write permission** Write access to the Back-up RAM must be explicitely enabled via the Back-up RAM control register BURC.

### 3.10.1 Back-up RAM protection

Write access to the Back-up RAM is disabled after reset release. Thus write access must be explicitly permitted by setting the write access permission bit BURC.BURWE = 1.

If a write access to the Back-up RAM occurs, while write is prohibited (BURC.BURWE = 0), the error bit BURAE.BURAERR is set.

The following registers control and monitor the write access to the Back-up RAM:

**Table 3-16 Back-up RAM registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| Back-up RAM control register | BURC | FF76 FE00$_H$ |
| Back-up RAM access error register | BURAE | FF76 FE04$_H$ |
| Back-up RAM access error clear register | BURAEC | FF76 FE08$_H$ |

#### (1) BURC – Back-up RAM control register

This register is used to permit respectively prohibit write access to the Back-up RAM.

**Access** This register can be read/written in 8-bit units.

**Address** FF76 FE00$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | BURWE |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 3-17 BURC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | BURWE | Back-up RAM write permission control:<br>0: write to Back-up RAM prohibited<br>1: write to Back-up RAM permitted |

### (2) BURAE – Back-up RAM access error register

This register reflects an erroneous write access to the Back-up RAM.

**Access** This register can be read in 8-bit units.

**Address** FF76 FE04$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | BURA ERR |
| R | R | R | R | R | R | R | R |

**Table 3-18 BURAE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | BURA ERR | Back-up RAM write access error flag<br>0: no Back-up RAM write access error<br>1: Back-up RAM write access error has occurred |

### (3) BURAEC – Back-up RAM access error clear register

This register is used to clear the Back-up RAM write access error flag BURAERR.

**Access** This register can be read/written in 8-bit units.

**Address** FF76 FE08$_H$

**Initial Value** Reading this register returns always 00$_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | BURA ERRC |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 3-19 BURAEC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | BURA ERRC | Back-up RAM write access error flag BURAERR clear<br>0: no function<br>1: clear BURAERR |

## 3.11 Write protected Registers

Write protected registers are protected from inadvertent write access due to erroneous program execution, etc.

Writing to a write protected register requires a special register protection unlock sequence.

### 3.11.1 Register protection clusters

The protected registers are bundled in certain register protection clusters.

The protection mechanism treats all registers of the same cluster as a single protection unit.

If the protection unlock sequence for a register is initiated, no access to any other register of the same protection cluster is allowed. Otherwise the unlock sequence is disrupted and the register write fails.
The diagram below shows a disruption of the unlock sequence by an access to the same cluster within an interrupt service routine.

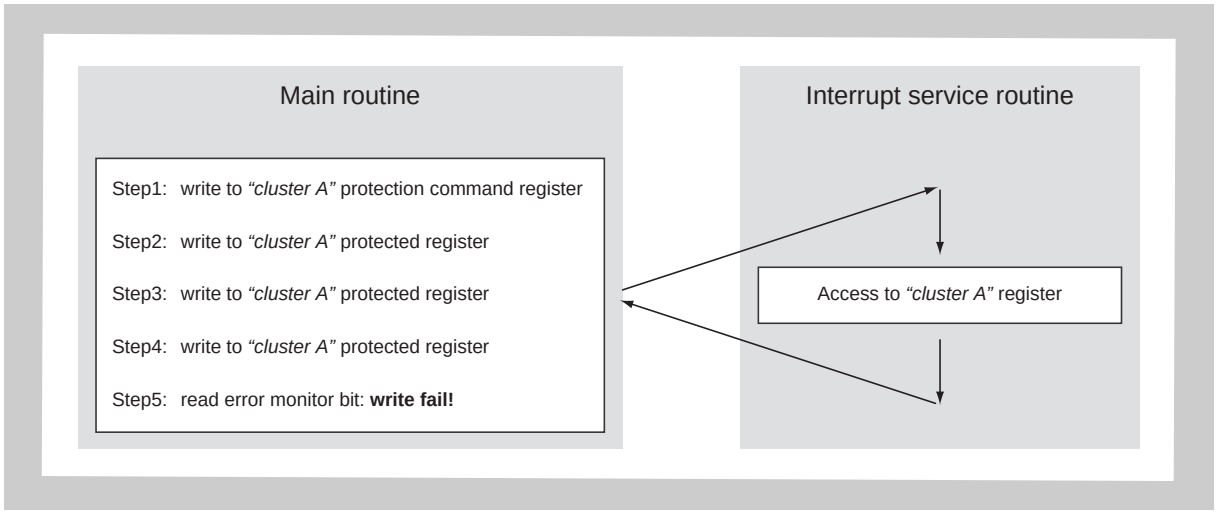



**Figure 3-15 Disruption of register protection unlock sequence**

Access to a register of another protection cluster during the unlock sequence does not disrupt the unlock sequence and the register write can be completed successfully.
The following diagram below shows such situation.

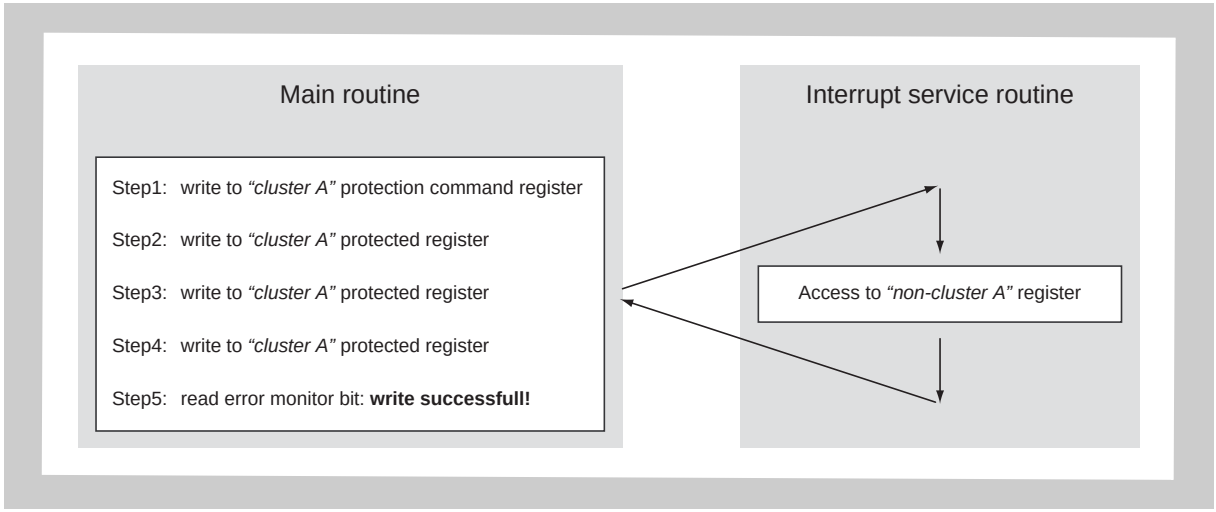



**Figure 3-16 Successful register protection unlock sequence**

The V850E2/Fx4-G register protection clusters are described in section *"V850E2/Fx4-G write protected registers"*.

### 3.11.2 Register protection unlock sequence

Write access to a write protected register is only possible within a special protection unlock sequence:

1. Write the fixed value $A5_H$ to the protection command register
2. Write the desired value to the protected register.
3. Write the bit-wise inversion of the desired value to the protected register.
4. Write the desired value to the protected register.
5. Verify successful write of the desired value to the protected register by verifying that the error monitor bit in the protection status register is "0". In case the write was not successful, indicated by the error monitor bit set to "1", the entire sequence has to be restarted at step 1.

In case of any access to another register between step 1 to step 4 of the above sequence, the protection mechanism behaves as follows:

- If the second register belongs to the same cluster, the write to the protected register fails (indicated by the error monitor bit set to "1"). The entire sequence has to be restarted at step 1.
- If the second register does not belong to the same cluster, the protection unlock sequence is not disrupted and the write to the first register can be completed successfully.

### 3.11.3 Register protection and interrupt/emulation break

If an interrupt/emulation break occurs during the protection unlock sequence, the protection mechanism behaves as follows:

**(1) Interrupts during protection unlock sequence**

If an interrupt is acknowledged during the above protection unlock sequence and the interrupt service routine does not access any register of the same register protection cluster, the protection unlock sequence is not disrupted and the write to the protected register can be successfully completed after returning from the interrupt service routine.

**(2) Emulator break during protection sequence**

If an emulation break occurs during the above protection unlock sequence, e.g. because of a breakpoint hit, the register protection is suspended until normal operation is resumed.
This means even if any register of the same cluster is accessed during the break, the protection unlock sequence is not disrupted and the error monitor bit is not set to “1”.

### 3.11.4 V850E2/Fx4-G write protected registers

The following table lists all V850E2/Fx4-G write protected registers:

**Table 3-20 Write protected registers**

| Module | Protected register | Protection registers | | Protection cluster |
|---|---|---|---|---|
| | | Command | Status | |
| Clock Controller | CKSC_0n | PROTCMD0 | PROTS0 | Control protection cluster 0 |
| Clock Controller | PLLE0 | PROTCMD2 | PROTS2 | Control protection cluster 2 |
| | MOSCE | | | |
| | ROSCE | | | |
| | CKSC_An | | | |
| Stand-by Controller | PSC0 | | | |
| Reset Controller | SWRESA | | | |
| | LVICNT | | | |
| On-Chip Debug control | IDMODII | PROT0PCMD | PROTS3 | OCD control protection cluster |
| Clock Monitors | CLMAnCTL0 | CLMAnPCMD | CLMAnPS | Clock Monitor protection cluster |
| Port control[a] | PODCn, JPODCn | PPCMDn | PPROTSn | Port protection cluster 1 to 4 |
| Self-programming control | FLMDCNT | FLMDPCMD | FLMDPS | Self-programming protection cluster |

a) Each port group n has its own protection command and status register. Refer to the section *“Port protection clusters”* below for details of port control registers protection.

**(1) Port protection clusters**

Following port registers feature write protection:

• Port open drain control registers PODCn, JPODCn

The above listed port control registers of certain port groups n are assigned to four port protection clusters:

**Table 3-21 Port protection clusters**

| Port protection cluster | Port groups |
|---|---|
| 1 | JP0 |
| 2 | P0 |
| 3 | P1, P3, P4, P10, P27 |

**Note** Each port group n has its own port protection command register PPCMDn and port protection status register PPROTSn.
However any port protection command register of the same port protection cluster can be used in the protection unlock sequence for enabling write access to all protected registers of the port protection cluster.

**Register width** The protected port control registers are 32-bit registers and the related protection command registers PPCMDn are 8-bit registers.

**Caution** All protected registers must be accessed by 32-bit accesses.

Thus the protection unlock sequence looks as follows:

1. Write the fixed value $A5_H$ to the protection command register PPCMDn.
2. Write the desired value to the protected register with the upper 16 bit bit[31:16] set to “0” (0000 $xxxx_H$).
3. Write the bit-wise inversion of the desired value to the protected register, thus the upper 16 bit bit[31:16] are set to “1” (FFFF $xxxx_H$).
4. Write the desired value to the protected register with the upper 16 bit bit[31:16] set to “0” (0000 $xxxx_H$).
5. Verify successful write of the desired value to the protected register by verifying that PPROTSn.PPROTSnERR = 0.

### 3.11.5 V850E2/Fx4-G Protection registers overview

The register write protection is controlled and operated by the following registers:

**Table 3-22 Protection command registers overview (1/2)**

| Register Name | Shortcut | Address |
|---|---|---|
| **Control protection clusters:** | | |
| Protection command register 0 | PROTCMD0 | FF42 4000$_H$ |
| Protection command register 2 | PROTCMD2 | FF42 0300$_H$ |
| Protection command register 3 | PROTCMD3 | FF42 0308$_H$ |
| Protection status register 0 | PROTS0 | FF42 4004$_H$ |
| Protection status register 2 | PROTS2 | FF42 0304$_H$ |
| Protection status register 3 | PROTS3 | FF42 030C$_H$ |
| **Clock monitors cluster:** | | |
| **CLMA0:** | | |
| Protection command register | CLMA0PCMD | FF80 2010$_H$ |
| Protection status register | CLMA0PS | FF80 2014$_H$ |
| **CLMA2:** | | |
| Protection command register | CLMA2PCMD | FF80 4010$_H$ |
| Protection status register | CLMA2PS | FF80 4014$_H$ |
| **CLMA3:** | | |
| Protection command register | CLMA3PCMD | FF80 5010$_H$ |
| Protection status register | CLMA3PS | FF80 5014$_H$ |
| **Port protection clusters** | | |
| **Port protection cluster 1** | | |
| Protection command register | JPPCMD0 | FF44 04C0$_H$ |
| Protection status register | JPPROTS0 | FF44 04B0$_H$ |
| **Port protection cluster 2** | | |
| Protection command registers | PPCMD0 | FF40 4C00$_H$ |
| Protection status registers | PPROTS0 | FF40 4B00$_H$ |
| **Port protection cluster 3** | | |
| Protection command registers | PPCMD1 | FF40 4C04$_H$ |
| | PPCMD3 | FF40 4C0C$_H$ |
| | PPCMD4 | FF40 4C10$_H$ |
| | PPCMD10 | FF40 4C28$_H$ |
| | PPCMD27 | FF40 4C6C$_H$ |
| Protection status registers | PPROTS1 | FF40 4B04$_H$ |
| | PPROTS3 | FF40 4B0C$_H$ |
| | PPROTS4 | FF40 4B10$_H$ |
| | PPROTS10 | FF40 4B28$_H$ |
| | PPROTS27 | FF40 4B6C$_H$ |
| **Self-programming protection cluster** | | |
| FLMD protection command register | FLMDPCMD | FF43 8004$_H$ |

**Table 3-22 Protection command registers overview (2/2)**

| Register Name | Shortcut | Address |
|---|---|---|
| FLMD protection error status register | FLMDPS | FF43 $8008_H$ |
| **OCD control protection cluster** | | |
| OCD protection command register | PROT0PCMD | FF02 $0308_H$ |
| OCD protection error status register | PROT0PS | FF02 $030C_H$ |

### 3.11.6 Control protection clusters registers details

#### (1) PROTCMDn – Protection command registers

These registers are the protection command registers for initiating the write protection unlock sequence for write protected registers.

**Index n** "n" denotes the number of the protection command registers, refer to the table *"Protection command registers overview"* in the previous section.

**Access** This register can be written in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** Reading this register returns an undefined value.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| W | W | W | W | W | W | W | W |

The usage of these registers is detailed in section *"Register protection unlock sequence"* above.

**Table 3-23 PROTCMDn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | – | Protection commands to enable writing to Isolated-Area-m registers |

### (2) PROTSn – Protection status registers

This registers shows the status of the protection unlock sequence operated via PROTCMDn.

**Index n** "n" denotes the number of the protection command registers, refer to the table *"Protection command registers overview"* in the previous section.

**Caution** This register must not be written.

**Access** This register can be read in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | PROTSn ERR |
| R | R | R | R | R | R | R | R |

**Table 3-24 PROTSn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | PROTSn ERR | Protected write sequence error monitor<br>0: no protection error<br>1: protection error occurred |

## 3.11.7 Clock monitors protection cluster registers details

### (1) CLMAnPCMD – CLMAn protection command register

This register is the protection command register for the CLMAnCTL0 register.

**Index n** "n" denotes the number of the Clock Monitor, refer to the table *"Protection command registers overview"* in the previous section.

**Access** This register can be written in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** Reading this register returns always $00_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | W | W | W | W | W | W | W |

**Table 3-25 CLMAnPCMD register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | – | Protection commands to enable writing to CLMAnCTL0 |

### (2) CLMAnPS – CLMAn protection status register

This register is used to verify whether the write protected register CLMAnCTL0 has been written successfully or not.

**Index n** "n" denotes the number of the Clock Monitor, refer to the table *"Protection command registers overview"* in the previous section.

**Access** This register can be read in 8-bit units.
Writing to this register is ignored.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | CLMAn PRERR |
| R | R | R | R | R | R | R | R |

**Table 3-26 CLMAnPS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | CLMAnPRERR | Indicates whether the write protected register CLMAnCTL0 has been written successfully:<br>0: Write operation successful<br>1: Write operation failed |

### 3.11.8 Port protection clusters registers details

#### (1) PPCMDn – Port protection command register

PPCMDn is the protection command register for port group n.

**Index n** "n" denotes the port group, refer to the table *"Protection command registers overview"* in the previous section.

**Access** This register can be written in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** Reading this register returns an undefined value.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| W | W | W | W | W | W | W | W |

**Table 3-27 PPCMDn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | – | Protection commands to enable writing to several port registers |

#### (2) PPROTSn – Port protection status register

PPROTSn is the protection status registers for write protected registers of port group n. It shows the status of the protection sequence operated via PPCMDn.

**Index n** "n" denotes the port group, refer to the table *"Protection command registers overview"* in the previous section.

**Caution** This register must not be written.

**Access** This register can be read in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | PPROTSn PRERR |
| R | R | R | R | R | R | R | R |

**Table 3-28 PPROTSn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | PPROTSn PRERR | Protected write sequence error monitor<br>0: no protection error<br>1: protection error occured |

### 3.11.9 Self-programming protection cluster registers details

#### (1) FLMDPCMD – FLMD protection command register

This register is the protection command register for the FLMDCNT register.

**Access** This register can be written in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** Reading this register returns always $00_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| W | W | W | W | W | W | W | W |

**Table 3-29 FLMDPCMD register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | – | Protection commands to enable writing to FLMDCNT |

#### (2) FLMDPS – FLMD protection error status register

This register is used to verify whether the write protected register FLMDCNT has been written successfully or not.

**Access** This register can be read in 8-bit units.
Writing to this register is ignored.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | FLMD PRERR |
| R | R | R | R | R | R | R | R |

**Table 3-30 FLMDPS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | FLMDPRERR | Indicates whether the write protected register FLMDCNT has been written successfully:<br>0: Write operation successful<br>1: Write operation failed |

## 3.11.10 OCD control protection cluster registers details

### (1) PROT0PCMD – OCD protection command register

This register is the protection command register for the OCD control register IDMODI.

**Access** This register can be written in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** Reading this register returns always an undefined value.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OCDPC[7:0] | | | | | | | |
| W | W | W | W | W | W | W | W |

**Table 3-31 PROT0PCMD register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | OCDPC[7:0] | Protection commands to enable writing to IDMODI |

### (2) PROT0PS – OCD protection error status register

This register is used to verify whether the write protected register FLMDCNT has been written successfully or not.

**Access** This register can be read in 8-bit units.

**Address** Refer to the table *"Protection command registers overview"* in the previous section.

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | OCDPRERR |
| R | R | R | R | R | R | R | R |

**Table 3-32 PROT0PS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | OCDPRERR | Indicates whether the write protected register IDMODI has been written successfully:<br>0: Write operation successful<br>1: Write operation failed |

# Chapter 4 Interrupt Functions

This chapter describes the exception processing functions of this microcontroller.

At first an overview is given about all exception groups.

Afterwards all interrupt exceptions of this microcontroller are summarized and the interrupt control registers are described.

Finally the handling of the interrupts exceptions are detailed.

All other exceptions and their handling are described in the

V850E2 32-bit Microcontroller Core Architecture
Document number <tbd>

where “xxxx” denotes the version of this document.

## 4.1 Exceptions and Interrupts

The phenomenon of forcing a branch operation from a currently running program to another program, due to a specific cause, is called an exception.

The exceptions are classified in execption groups. Each exception group is assigned to a certain priority P1 to P7, that determines in which order exceptions are processed, if they occur concurrently.

This microcontroller supports the following types of exceptions.

**Table 4-1 Exception and interrupt cause list (1/2)**

| Exception and interrupt name | Symbol | Cause group | Priority group | Exception level | Reference |
|---|---|---|---|---|---|
| CPU initialization | RESET | Reset input | P1 | - | this manual *“Reset Controller”* |
| FE level non-maskable interrupt | FENMI | FENMI input | P2 | FE | this chapter *“V850E2/Fx4-G Interrupt Requests”* |
| System error exception | SYSERR | SYSERR input | P3 | FE | this chapter *“V850E2/Fx4-G Exceptions”* |
| EI level maskable interrupt | INT | Maskable interrupt input | P4 | EI | this chapter *“V850E2/Fx4-G Interrupt Requests”* |
| Execution protection exception | MIP | Execution protection violation | P5 | FE | V850E2 32-bit Microcontroller Core Architecture User Manual *“Memory Protection”* |
| Memory error exception | MEP | Instruction access error input | P6 | FE | this chapter *“V850E2/Fx4-G Exceptions”* |

**Table 4-1 Exception and interrupt cause list (2/2)**

| Exception and interrupt name | Symbol | Cause group | Priority group | Exception level | Reference |
|---|---|---|---|---|---|
| Data protection exception | MDP | Data protection violation | P7 | FE | V850E2 32-bit Microcontroller Core Architecture User Manual *"Memory Protection"* |
| Coprocessor unusable exception | UCPOP | Coprocessor instruction | | FE | V850E2 32-bit Microcontroller Core Architecture User Manual *"Exceptions"* |
| Reserved instruction exception | RIEX | Reserved instruction | | FE | V850E2 32-bit Microcontroller Core Architecture User Manual *"Exceptions"* |
| FE level software exception | FETRAPEX | FETRAP instruction (vector = $1_H$ to $F_H$) | | FE | V850E2 32-bit Microcontroller Core Architecture User Manual *"Instructions"* |
| EI level software exception | EITRAP0 | TRAP0n instruction (vector = $00_H$ to $0F_H$) | | EI | |
| EI level software exception | EITRAP1 | TRAP1n instruction (vector = $10_H$ to $1F_H$) | | EI | |
| System call exception | SYSCALLEX | SYSCALL instruction (vector = $00_H$ to $FF_H$) | | EI | V850E2 32-bit Microcontroller Core Architecture User Manual *"Exceptions"* |

**Priority order** Priority group P1 has the highest, P7 the lowest priority.

**Request** Exception or interrupt request denotes the status, where an exception or interrupt event occurred and is registered in the CPU or Interrupt Controller respectively to be served.

**Acknowledgement** Exception or interrupt acknowledgement denotes the status, where the CPU branches to an exception or interrupt service or handler routine. Thus the current program flow is suspended.

**Acknowledgement condition** Before an exception or interrupt request is acknowledgement, certain acknowledgement conditions may have to be fulfilled. Note that certain exceptions are acknowledged unconditionally.

**Resume** Indicates whether execution restart from the last position at which program execution was interrupted - i.e. an exception or interrupt was acknowledged - is possible.

**Restore** Indicates whether restoring of the processor status (status of processor resources including general-purpose registers and system registers) at the time of program execution interruption is possible.

## 4.2 V850E2/Fx4-G Exceptions

This section describes the V850E2/Fx4-G exceptions.

For detailed information about how to handle exceptions refer to the chapter *"Exceptions"* in the *"V850E2 32-bit Microcontroller Core Architecture"*.

### 4.2.1 Memory error exceptions MEP

A memory error exception MEP indicates an error, that occurred during a CPU instruction fetch:

- MEP exception level: FE level without acknowledgement conditions
- MEP exception code: FEIC = 0000 $0330_H$
- MEP exception handler offset: $0030_H$
- MEP priority level: P7

MEP exceptions are not maskable.

The MEP exception is generated upon reading instruction code from memory. Note that this must not necessarily be the time the CPU is fetching the instruction for execution. Also a cache or buffer preload can cause an MEP exception.

**MEP sources** The source of an MEP exception can be one of the following:

- uncorrectable ECC double-error detection during instruction fetch from the code flash
- uncorrectable ECC double-error detection during instruction fetch from the data RAM
- instruction fetch attempt from an undefined memory area (reserved area in the CPU address map)

**MEP indicators** The source of an MEP exception can be evaluated by the following:

- Code flash ECC double-error: CECCER.DEDFLG = 1
  Refer to the section *"Code flash error correction"* below for further details
- For data RAM ECC double-error detections and undefined memory area access there is no source indication other than the address of the program counter in the CPU's FEPC register, stored when the MEP occurred.

**Note** Single bit errors are corrected by the ECC and processing is continued using the fetched data after correction. No MEP is generated in this case.

**MEP resume/restore**

Since an MEP is acknowledged unconditionally at the time of reading an instruction code from a memory, which may be different to the time of CPU fetch of this instruction code, neither resuming to the application program nor recovering of the CPU's status after the MEP service routine is possible. Thus a system reset must be applied or a substitute program to recover the status must be executed.

When an additional MEP exception occurs during execution of an MEP exception handler program, the MEP exception is acknowledged again and it's handler routine re-executed.

Moreover, even during MEP exception processing, SYSERR exceptions and FE level non-maskable interrupts may occur. If this happens, SYSERR exception or FE level non-maskable interrupt processing starts at the time the error occurs.

### 4.2.2 System error exceptions SYSERR

In case of a serious system error, a system error exception SYSERR can be generated.

SYSERR exception can be enabled or disabled. A flag register is provided to determine the cause of such errors.

- SYSERR exception level: FE level without acknowledgement conditions
- SYSERR exception code: refer to the table *"System errors summary"* below
- SYSERR exception handler offset: $0030_H$
- SYSERR priority level: P3

SYSERR exceptions are maskable.

#### (1) System error causes

**Error conditions**

The generation of a SYSERR exception can occur under following error conditions:

- error because of an uncorrectable ECC double-error detection during data read from the code flash
- error because of an uncorrectable ECC double-error detection during data read from the data RAM

These error conditions may be fulfilled during a CPU or DMA access.

**SYSERR enable/disable**

For CPU accesses any of the above error conditions can be separately enabled/disabled to generate a SYSERR exception by use of the system error control register SEG_CONT.

For DMA accesses a single control bit in SEG_CONT.DMAE is provided to enable SYSERR generation for any of the above error conditions all together. Refer to the section *"Error response support"* in the chapter *"DMA Controller (DMAC)"* for further information concerning DMAC errors.

**SYSERR flags** The source of a SYSERR exception can be evaluated by the system error flag register SEG_FLAG, that holds

- dedicated flags for each of the error conditions, if the error condition was met during a CPU access.
- a single bit that indicates that any of the error condition was met during a DMA access.

**Notes**

1. If an error occurs during a CPU access, the exact error cause can be evaluated by the dedicated SEG_FLAG flags.
2. Upon occurrence of an error caused by DMA access, SEG_FLAG.DMAF is set regardless of the exact cause of the error. Therefore, the CPU cannot determine the exact error cause from SEG_FLAG.
3. Setting of the flags in SEG_FLAG is independent of the SYSERR enable/ disable setting in the SEG_CONT register.

**Table 4-2 System errors summary**

| Access by | System error cause | Error flag SEG_FLAG. | SYSERR exception if SEG_CONT. | SYSERR exception code in FEIC |
|---|---|---|---|---|
| CPU | data read from code flash | FCHF = 1 | FCHE = 1 | $0000\ 0231_H$ |
| | data read from data RAM | RAMF = 1 | RAME = 1 | |
| DMA | • data read from code flash<br>• data read from data RAM | DMAF = 1 | DMAE = 1 | $0000\ 0232_H$ |

### (2) System error condition details

1. Error during data read from code flash
   This error occurs when two defective bits are detected by the code flash ECC, which can not be corrected, as the result of read access to the code flash area.

2. Error during data read from data RAM
   This error occurs when two defective bits are detected by the data RAM ECC, which can not be corrected, as the result of read access to the data RAM area.

3. Occurrence of error during DMA access
   Upon detection of one of the above errors during DMA read/write accesses, the DMA controller stops the transfer operations of all channels. Refer to the section *"Error response support"* in the chapter *"DMA Controller (DMAC)"* for further information concerning DMAC errors.

### (3) SEG_CONT - System error control register

This register enables respectively disables system error causes to generate a SYSERR exception.

**Access** This register can be read/written

- 16-bit units via the register SEG_CONT
- the lower 8 bits in 8-bit units via the register SEG_CONTL
- the higher 8 bits in 8-bit units via the register SEG_CONTH

**Address** SEG_CONT: FFFF 64B0$_H$
SEG_CONTL: FFFF 64B0$_H$
SEG_CONTH: FFFF 64B1$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DMAE | 0 | 0 | RAME | 0 | 0 | FCHE | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-3 SEG_CONT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | DMAE | Enable SYSERR exception generation during DMA accesses<br>0: SYSERR generation disabled<br>1: SYSERR generation enabled |
| 4 | RAME | Enable SYSERR exception generation during CPU data read accesses to the data RAM<br>0: SYSERR generation disabled<br>1: SYSERR generation enabled |
| 1 | FCHE | Enable SYSERR exception generation during CPU data read accesses to the code flash<br>0: SYSERR generation disabled<br>1: SYSERR generation enabled |

### (4) SEG_FLAG - System error flag register

This register informs about the occurrence of a SYSERR condition.

Note that the concerned flag is set upon a SYSERR condition, independent of SYSERR generation enabled/disabled via the SEG_CONT register.

**Access** This register can be read/written

- 16-bit units via the register SEG_FLAG
- the lower 8 bits in 8-bit units via the register SEG_FLAGL
- the higher 8 bits in 8-bit units via the register SEG_FLAGH

**Address** SEG_FLAG: FFFF 64B2$_H$
SEG_FLAGL: FFFF 64B2$_H$
SEG_FLAGH: FFFF 64B3$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DMAF | 0 | 0 | RAMF | 0 | 0 | FCHF | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-4 SEG_FLAG register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | DMAF | SYSERR condition during DMA accesses<br>0: SYSERR condition has not occurred<br>1: SYSERR condition has occurred |
| 4 | RAMF | SYSERR condition during CPU data read accesses to the data RAM<br>0: SYSERR condition has not occurred<br>1: SYSERR condition has occurred |
| 1 | FCHF | SYSERR condition during CPU data read accesses to the code flash<br>0: SYSERR condition has not occurred<br>1: SYSERR condition has occurred |

### 4.2.3 Code flash error correction

The code flash memory is equipped with an Error Correction module ECC. When the CPU or the DMA accesses the code flash, the ECC module automatically encodes/decodes the ECC code.

**Single-bit errors**

In case of a single-bit error while reading from the code flash, the ECC module corrects this error automatically, i.e. the code flash ECC performs a single-error correction (SEC).

Detection and correction of a single-bit error is indicated by the SEC flag CECCER.SECFLG = 1.
The code flash address, where this single-bit error has occurred, is stored in the CSECADR register.

**Double-bit errors**

In case of double-bit errors while reading from the code flash, the ECC module detects this error, i.e. the code flash ECC performs a double-error detection (DED).

Detection of a double-bit error is indicated by the DED flag CECCER.DEDFLG = 1.
The code flash address, where this double-bit error has occurred, is stored in the CDEDADR register.

Upon detection of a double-bit error an exception can be generated.
Refer to the sections *"Memory error exceptions MEP"* and *"System error exceptions SYSERR"* about further information concerning exception processing of double-error detections.

**Note** Three or more erroneous bits may not detect.

#### (1) Code flash error correction registers overview

This section contains a description of all registers of the code flash error correction module.

**Table 4-5 Code flash Error Correction module registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| Code flash ECC error flag register | CECCER | FF43 2000$_H$ |
| Code flash ECC error flag clear register | CECCERC | FF43 2004$_H$ |
| Code flash ECC single-bit error correction address register | CSECADR | FF43 2008$_H$ |
| Code flash ECC double-bit error detection address register | CDEDADR | FF43 200C$_H$ |

### (2) CECCER - Code flash ECC error flag register

This register informs about a correction of a single-bit (SEC) error and detection of a double-bit (DED) error.

To clear an asserted bit in this register use the CECCERC register.

**Access** These registers can be read in 8-bit units.

**Address** FF43 2000$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | DEDFLG | SECFLG |
| R | R | R | R | R | R | R | R |

**Table 4-6 CECCER register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | DEDFLG | double-bit error detection flag<br>0: double-bit error was not detected<br>1: double-bit error was not detected |
| 0 | SECFLG | single-bit error correction flag<br>0: single-bit error was not detected<br>1: single-bit error was detected |

### (3) CECCERC - Code flash ECC error flag clear register

This register allows to clear the ECC error flags of CECCER.

If a 1 is written to a bit in this register, the corresponding bit in the CECCER register is cleared.

**Access** These registers can be read/written in 8-bit units.

**Address** FF43 2004$_H$

**Initial Value** Reading this registers returns always 00$_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | DEDCLR | SEDCLR |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-7 CECCERC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | DEDCLR | Clear double-bit error detection flag<br>0: no function<br>1: clear CECCER.DEDFLG |
| 0 | SECCLR | Clear single-bit error detection flag<br>0: no function<br>1: clear CECCER.SECFLG |

### (4) CSECADR - Code flash ECC single-bit error correction address register

This register holds the address at which a single-bit error was detected and corrected.

Address storing in and clearing of this register depends on the single-bit error correction flag CECCER.SECFLG:

- The error address is only stored, if CECCER.SECFLG = 0.
- This register is cleared together with the flag CECCER.SECFLG by setting CECCERC.SECCLR = 1.

Thus only the address of the first detected error is stored. The addresses of all others which may follow, do not overwrite the first address, until the single-bit error flag - and thus this register - is cleared by CECCERC.SECLR = 1.

**Access** These registers can be read in 32-bit units.

**Address** FF43 2008$_H$

**Initial Value** 0000 0000$_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | CSECADR[24:16] | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSECADR[7:0] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 4-8 CSECADR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 24 to 0 | CSECADR [24:0] | single-bit error detection and correction address |

### (5) CDEDADR - Code flash ECC double-bit error detection address register

This register holds the address, at which a double-bit error was detected.

Address storing in and clearing of this register depends on the double-bit error detection flag CECCER.DEDFLG:

- The error address is only stored, if CECCER.DEDFLG = 0.
- This register is cleared together with the flag CECCER.DEDFLG by setting CECCERC.DEDCLR = 1.

Thus only the address of the first detected error is stored. The addresses of all others, which may follow, do not overwrite the first address, until the double-bit error flag - and thus this register - is cleared by CECCERC.DEDCLR = 1.

**Access** These registers can be read in 32-bit units.

**Address** FF43 200C$_H$

**Initial Value** 0000 0000$_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | CDECADR[24:16] | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CDECADR[7:0] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 4-9 CDEDADR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 24 to 0 | CDECADR [24:0] | double-bit error detection address |

## 4.3 V850E2/Fx4-G Interrupt Requests

**Interrupt types**

The V850E2/Fx4-G supports following types of interrupts:

- FE level non-maskable interrupt (FENMI)
  - FENMI is served immediately, even if an FE level maskable (FEINT) or EI level maskable (EIINT) interrupt is in service (CPU system register PSW.NP = 1).
  - resume not possible, recover not possible
- EI level maskable interrupt (EIINT)
  - EIINT may only be served, if no FE level interrupt - FENMI or FEINT - is in service (CPU system register PSW.NP = 0).
  - resume possible, recover possible
  - interrupt masking can be specified for each interrupt channel
  - 16 interrupt priority levels can be specified for each interrupt channel.

### 4.3.1 V850E2/Fx4-G interrupt sources

**(1) FE level non-maskable interrupts**

**Priority group** The FE level non-maskable interrupts have the priority P2.

**Return PC** An FE non-maskable interrupt does not allow to resume or recover.

**Control register** The control register of this interrupt is FNC.

**Return instruction** Since neither resume nor recover is possible, a reset must be applied.

**Table 4-10 FE level non-maskable interrupt requests**

| Interrupt | | | Interrupt request | | Unit | Priority group | Exception code | Handler address offset |
|---|---|---|---|---|---|---|---|---|
| Symbol | Control register | | Name | Cause | | | | |
| | Name | Address FFFF... | | | | | | |
| FENMI | FNC | $645C_H$ | NMI0 | NMI0 input | Port | P2 | $0020_H$ | $0020_H$ |
| | | | WDTA0NMI | Watchdog Timer 0 error NMI interrupt | WDTA0 | | | |
| | | | WDTA1NMI | Watchdog Timer 1 error NMI interrupt | WDTA1 | | | |

**FENMI sharing** The source of the FENMI interrupt can be evaluated by a dedicated flag register. Refer to section *4.3.2 “FE level non-maskable interrupt sharing” on page 167* for details.

### (2) EI level maskable interrupts

**Interrupt naming**

The composition of the interrupt request signal names, their assigned interrupt control registers and the bits in these registers follow special rules.

In the following the name of the interrupt request is represented by <*name*>.

- Interrupt request name: **INT**<*name*>
  The prefix "**INT**" is put in front of <*name*>.
- Interrupt request control register: **IC**<*name*>
  The prefix "**IC**" is put in front of <*name*>.
  The 16 bit of the 16-bit register **IC**<*name*> can also be accessed byte-wise with the following names:
  - low byte (bits[7:0]): **IC**<*name*>**L** at the address of **IC**<*name*>
    The suffix "**L**" is appended to the register name **IC**<*name*>.
  - high byte (bits[15:8]): **IC**<*name*>**H** at the address of **IC**<*name*> + 1
    The suffix "**H**" is appended to the register name **IC**<*name*>.
- Interrupt control register bit names: **RF**<*name*>, **MK**<*name*>, **P[2:0]**<*name*>
  The bit prefix "**RF**", "**MK**", "**P[2:0]**" prepends the interrupt <*name*>.
- Each interrupt request is assigned to a certain interrupt channel number n = 0 to 255.
  The functional description of the Interrupt Controller in this chapter refers to interrupt channel number *n* instead of to the interrupt name <*name*>.
  If **INT**<*name*> is assigned to the interrupt channel number *n*, throughout this chapter
  - the interrupt request is named **INT***n*
  - the assigned interrupt control register is named **IC***n*
  - the interrupt control bits are named **RF***n*, **MK***n*, **P[2:0]***n*.

**Example**

The interrupt request of the second TAUA0 channel (<*name*> = *TAUA0I2*) is named

**INT***TAUA0I2*

The related interrupt control registers are

**IC***TAUA0I2*, **IC***TAUA0I2L*, **IC***TAUA0I2H*

The bits in this register are

**RF***TAUA0I2*, **MK***TAUA0I2*, **P[2:0]***TAUA0I2*

If, for instance, the interrupt channel for **INT***TAUA0I2* is n = *22*, the functional description of the Interrupt Controller refers to

**INT***22*,

the related control register as

**IC***22,*

and their control register bit names

**RF***22*, **MK***22*, **P[2:0]***22*.

The following tables list the references between the interrupt channel number *n*, the assigned V850E2/Fx4-G interrupt requests and control register names.

**Priority group** The EI level maskable interrupts have the priority P8.

**Return PC** The program couter (PC) value, set after returning from any interrupt service routine by the EIRET instruction is always the next address.

**Return instruction** The return instruction from EI level maskable interrupt service routines is EIRET.

### (3) V850E2/FF4-G EI level maskable interrupts

Table 4-11 V850E2/FF4-G EI level maskable interrupt requests (1/3)

| Interrupt | | | Interrupt request | | | Default priority | Exception code | Handler address offset |
|---|---|---|---|---|---|---|---|---|
| Channel | Control register | | Name | Cause | Unit | | | |
| | Name | Address FFFF... | | | | | | |
| 0 | ICWDTA0 | $6000_H$ | INTWDTA0 | WDTA0 75% interrupt | WDTA0 | 1 | $0080_H$ | $0080_H$ |
| 1 | ICWDTA1 | $6002_H$ | INTWDTA1 | WDTA1 75% interrupt | WDTA1 | 2 | $0090_H$ | $0090_H$ |
| 2 | ICLVI | $6004_H$ | INTLVI | LVI interrupt | LVI | 3 | $00A0_H$ | $00A0_H$ |
| 3 to 8 | – | $6006_H$ to $6010_H$ | reserved | | | 4 to 9 | $00B0_H$ to $0100_H$ | $00B0_H$ to $0100_H$ |
| 9 | ICP0 | $6012_H$ | INTP0 | Edge detection interrupt | Port | 10 | $0110_H$ | $0110_H$ |
| 10 | ICP1 | $6014_H$ | INTP1 | Edge detection interrupt | Port | 11 | $0120_H$ | $0120_H$ |
| 11 | ICP2 | $6016_H$ | INTP2 | Edge detection interrupt | Port | 12 | $0130_H$ | $0130_H$ |
| 12 | ICP3 | $6018_H$ | INTP3 | Edge detection interrupt | Port | 13 | $0140_H$ | $0140_H$ |
| 13 | ICP4 | $601A_H$ | INTP4 | Edge detection interrupt | Port | 14 | $0150_H$ | $0150_H$ |
| 14 | – | $601C_H$ | reserved | | | 15 | $0160_H$ | $0160_H$ |
| 15 | ICP6 | $601E_H$ | INTP6 | Edge detection interrupt | Port | 16 | $0170_H$ | $0170_H$ |
| 16 | – | $6020_H$ | reserved | | | 17 | $0180_H$ | $0180_H$ |
| 17 | ICP8 | $6022_H$ | INTP8 | Edge detection interrupt | Port | 18 | $0190_H$ | $0190_H$ |
| 18 | – | $6024_H$ | reserved | | | 19 | $01A0_H$ | $01A0_H$ |
| 19 | – | $6026_H$ | reserved | | | 20 | $01B0_H$ | $01B0_H$ |
| 20 | ICTAUB0I0 | $6028_H$ | INTTAUB0I0 | Interrupt for channel 0 | TAUB0 | 21 | $01C0_H$ | $01C0_H$ |
| 21 | ICTAUB0I1 | $602A_H$ | INTTAUB0I1 | Interrupt for channel 1 | TAUB0 | 22 | $01D0_H$ | $01D0_H$ |
| 22 | ICTAUB0I2 | $602C_H$ | INTTAUB0I2 | Interrupt for channel 2 | TAUB0 | 23 | $01E0_H$ | $01E0_H$ |
| 23 | ICTAUB0I3 | $602E_H$ | INTTAUB0I3 | Interrupt for channel 3 | TAUB0 | 24 | $01F0_H$ | $01F0_H$ |
| 24 | ICTAUB0I4 | $6030_H$ | INTTAUB0I4 | Interrupt for channel 4 | TAUB0 | 25 | $0200_H$ | $0200_H$ |
| 25 | ICTAUB0I5 | $6032_H$ | INTTAUB0I5 | Interrupt for channel 5 | TAUB0 | 26 | $0210_H$ | $0210_H$ |
| 26 | ICTAUB0I6 | $6034_H$ | INTTAUB0I6 | Interrupt for channel 6 | TAUB0 | 27 | $0220_H$ | $0220_H$ |
| 27 | ICTAUB0I7 | $6036_H$ | INTTAUB0I7 | Interrupt for channel 7 | TAUB0 | 28 | $0230_H$ | $0230_H$ |
| 28 | ICTAUB0I8 | $6038_H$ | INTTAUB0I8 | Interrupt for channel 8 | TAUB0 | 29 | $0240_H$ | $0240_H$ |
| 29 | ICTAUB0I9 | $603A_H$ | INTTAUB0I9 | Interrupt for channel 9 | TAUB0 | 30 | $0250_H$ | $0250_H$ |
| 30 | ICTAUB0I10 | $603C_H$ | INTTAUB0I10 | Interrupt for channel 10 | TAUB0 | 31 | $0260_H$ | $0260_H$ |
| 31 | ICTAUB0I11 | $603E_H$ | INTTAUB0I11 | Interrupt for channel 11 | TAUB0 | 32 | $0270_H$ | $0270_H$ |
| 32 | ICTAUB0I12 | $6040_H$ | INTTAUB0I12 | Interrupt for channel 12 | TAUB0 | 33 | $0280_H$ | $0280_H$ |
| 33 | ICTAUB0I13 | $6042_H$ | INTTAUB0I13 | Interrupt for channel 13 | TAUB0 | 34 | $0290_H$ | $0290_H$ |
| 34 | ICTAUB0I14 | $6044_H$ | INTTAUB0I14 | Interrupt for channel 14 | TAUB0 | 35 | $02A0_H$ | $02A0_H$ |
| 35 | ICTAUB0I15 | $6046_H$ | INTTAUB0I15 | Interrupt for channel 15 | TAUB0 | 36 | $02B0_H$ | $02B0_H$ |
| 36 to 51 | – | $6048_H$ to $6066_H$ | reserved | | | 37 to 52 | $02C0_H$ to $03B0_H$ | $02C0_H$ to $03B0_H$ |
| 52 | ICADCA0ERR | $6068_H$ | INTADCA0ERR | Error interrupt | ADCA0 | 53 | $03C0_H$ | $03C0_H$ |
| 53 | ICADCA0I0 | $606A_H$ | INTADCA0I0 | End of CG0 conversion | ADCA0 | 54 | $03D0_H$ | $03D0_H$ |
| 54 | ICADCA0I1 | $606C_H$ | INTADCA0I1 | End of CG1 conversion | ADCA0 | 55 | $03E0_H$ | $03E0_H$ |
| 55 | ICADCA0I2 | $606E_H$ | INTADCA0I2 | End of CG2 conversion | ADCA0 | 56 | $03F0_H$ | $03F0_H$ |
| 56 | ICADCA0LLT | $6070_H$ | INTADCA0LLT | Conversion interrupt | ADCA0 | 57 | $0400_H$ | $0400_H$ |

**Table 4-11 V850E2/FF4-G EI level maskable interrupt requests (2/3)**

| Interrupt | | | Interrupt request | | | Default priority | Exception code | Handler address offset |
|---|---|---|---|---|---|---|---|---|
| Channel | Control register | | Name | Cause | Unit | | | |
| | Name | Address FFFF... | | | | | | |
| 57 | ICFCNWUP | $6072_H$ | INTFCNWUP | Wake up interrupt | FCN [5:0][a] | 58 | $0410_H$ | $0410_H$ |
| 58 | – | $6074_H$ | reserved | | | 59 | $0420_H$ | $0420_H$ |
| 59 | – | $6076_H$ | reserved | | | 60 | $0430_H$ | $0430_H$ |
| 60 | ICFCN0ERR | $6078_H$ | INTFCN0ERR | Error interrupt | FCN0 | 61 | $0440_H$ | $0440_H$ |
| 61 | ICFCN0REC | $607A_H$ | INTFCN0REC | Receive interrupt | FCN0 | 62 | $0450_H$ | $0450_H$ |
| 62 | ICFCN0TRX | $607C_H$ | INTFCN0TRX | Transmit interrupt | FCN0 | 63 | $0460_H$ | $0460_H$ |
| 63 | ICCSIG0IRE | $607E_H$ | INTCSIG0IRE | Reception error interrupt | CSIG0 | 64 | $0470_H$ | $0470_H$ |
| 64 | ICCSIG0IR | $6080_H$ | INTCSIG0IR | Reception status interrupt | CSIG0 | 65 | $0480_H$ | $0480_H$ |
| 65 | ICCSIG0IC | $6082_H$ | INTCSIG0IC | Communication status interrupt | CSIG0 | 66 | $0490_H$ | $0490_H$ |
| 66 | ICDMA0 | $6084_H$ | INTDMA0 | DMA channel 0 transfer completion | DMA | 67 | $04A0_H$ | $04A0_H$ |
| 67 | ICDMA1 | $6086_H$ | INTDMA1 | DMA channel 1 transfer completion | DMA | 68 | $04B0_H$ | $04B0_H$ |
| 68 | ICDMA2 | $6088_H$ | INTDMA2 | DMA channel 2 transfer completion | DMA | 69 | $04C0_H$ | $04C0_H$ |
| 69 | ICDMA3 | $608A_H$ | INTDMA3 | DMA channel 3 transfer completion | DMA | 70 | $04D0_H$ | $04D0_H$ |
| 70 | ICDMA4 | $608C_H$ | INTDMA4 | DMA channel 4 transfer completion | DMA | 71 | $04E0_H$ | $04E0_H$ |
| 71 | ICDMA5 | $608E_H$ | INTDMA5 | DMA channel 5 transfer completion | DMA | 72 | $04F0_H$ | $04F0_H$ |
| 72 | ICDMA6 | $6090_H$ | INTDMA6 | DMA channel 6 transfer completion | DMA | 73 | $0500_H$ | $0500_H$ |
| 73 | ICDMA7 | $6092_H$ | INTDMA7 | DMA channel 7 transfer completion | DMA | 74 | $0510_H$ | $0510_H$ |
| 74 | ICIICB0IA | $6094_H$ | INTIICB0IA | Data transmission/reception interrupt | IICB0 | 75 | $0520_H$ | $0520_H$ |
| 75 | ICFCN1ERR | $6096_H$ | INTFCN1ERR | Error interrupt | FCN1 | 76 | $0530_H$ | $0530_H$ |
| 76 | ICFCN1REC | $6098_H$ | INTFCN1REC | Receive interrupt | FCN1 | 77 | $0540_H$ | $0540_H$ |
| 77 | ICFCN1TRX | $609A_H$ | INTFCNTRX | Transmit interrupt | FCN1 | 78 | $0550_H$ | $0550_H$ |
| 78 | ICTAUJ0I0 | $609C_H$ | INTTAUJ0I0 | Interrupt for channel 0 | TAUJ0 | 79 | $0560_H$ | $0560_H$ |
| 79 | ICTAUJ0I1 | $609E_H$ | INTTAUJ0I1 | Interrupt for channel 1 | TAUJ0 | 80 | $0570_H$ | $0570_H$ |
| 80 | ICTAUJ0I2 | $60A0_H$ | INTTAUJ0I2 | Interrupt for channel 2 | TAUJ0 | 81 | $0580_H$ | $0580_H$ |
| 81 | ICTAUJ0I3 | $60A2_H$ | INTTAUJ0I3 | Interrupt for channel 3 | TAUJ0 | 82 | $0590_H$ | $0590_H$ |
| 82 | ICOSTM0 | $60A4_H$ | INTOSTM0 | OSTM0 interrupt | OSTM0 | 83 | $05A0_H$ | $05A0_H$ |
| 83 | ICCSIG4IRE | $60A6_H$ | INTCSIG4IRE | Reception error interrupt | CSIG4 | 84 | $05B0_H$ | $05B0_H$ |
| 84 | ICCSIG4IR | $60A8_H$ | INTCSIG4IR | Reception status interrupt | CSIG4 | 85 | $05C0_H$ | $05C0_H$ |
| 85 | ICCSIG4IC | $60AA_H$ | INTCSIG4IC | Communication status interrupt | CSIG4 | 86 | $05D0_H$ | $05D0_H$ |
| 86 to 100 | – | $60AC_H$ to $60C8_H$ | reserved | | | 87 to 101 | $05E0_H$ to $06C0_H$ | $05E0_H$ to $06C0_H$ |
| 101 | ICP11 | $60CA_H$ | INTP11 | Edge detection interrupt | Port | 102 | $06D0_H$ | $06D0_H$ |
| 102 | ICP12 | $60CC_H$ | INTP12 | Edge detection interrupt | Port | 103 | $06E0_H$ | $06E0_H$ |
| 103 to 105 | – | $60CE_H$ to $60D2_H$ | reserved | | | 104 to 106 | $06F0_H$ to $0710_H$ | $06F0_H$ to $0710_H$ |
| 106 | ICKR0 | $60D4_H$ | INTKR0 | Key return interrupt | KR0 | 107 | $0720_H$ | $0720_H$ |
| 107 | ICLMA10IS | $60D6_H$ | INTLMA10IS | Status interrupt | LMA10 | 108 | $0730_H$ | $0730_H$ |
| 108 | ICLMA10IR | $60D8_H$ | INTLMA10IR | Reception completion interrupt | LMA10 | 109 | $0740_H$ | $0740_H$ |
| 109 | ICLMA10IT | $60DA_H$ | INTLMA10IT | Transmission interrupt | LMA10 | 110 | $0750_H$ | $0750_H$ |

**Table 4-11 V850E2/FF4-G EI level maskable interrupt requests (3/3)**

| Interrupt | | | Interrupt request | | | Default priority | Exception code | Handler address offset |
|---|---|---|---|---|---|---|---|---|
| Channel | Control register | | Name | Cause | Unit | | | |
| | Name | Address FFFF... | | | | | | |
| 110 | ICLMA11IS | $60DC_H$ | INTLMA11IS | Status interrupt | LMA11 | 111 | $0760_H$ | $0760_H$ |
| 111 | ICLMA11IR | $60DE_H$ | INTLMA11IR | Reception completion interrupt | LMA11 | 112 | $0770_H$ | $0770_H$ |
| 112 | ICLMA11IT | $60E0_H$ | INTLMA11IT | Transmission interrupt | LMA11 | 113 | $0780_H$ | $0780_H$ |
| 113 | ICFCN2ERR | $60E2_H$ | INTFCN2ERR | Error interrupt | FCN2 | 114 | $0790_H$ | $0790_H$ |
| 114 | ICFCN2REC | $60E4_H$ | INTFCN2REC | Receive interrupt | FCN2 | 115 | $07A0_H$ | $07A0_H$ |
| 115 | ICFCN2TRX | $60E6_H$ | INTFCN2TRX | Transmit interrupt | FCN2 | 116 | $07B0_H$ | $07B0_H$ |
| 116 | ICFCN3ERR[b] | $60E8_H$ | INTFCN3ERR | Error interrupt | FCN3 | 117 | $07C0_H$ | $07C0_H$ |
| 117 | ICFCN3REC[b] | $60EA_H$ | INTFCN3REC | Receive interrupt | FCN3 | 118 | $07D0_H$ | $07D0_H$ |
| 118 | ICFCN3TRX[b] | $60EC_H$ | INTFCN3TRX | Transmit interrupt | FCN3 | 119 | $07E0_H$ | $07E0_H$ |
| 119 | ICFCN4ERR[b] | $60EE_H$ | INTFCN4ERR | Error interrupt | FCN4 | 120 | $07F0_H$ | $07F0_H$ |
| 120 | ICFCN4REC[b] | $60F0_H$ | INTFCN4REC | Receive interrupt | FCN4 | 121 | $0800_H$ | $0800_H$ |
| 121 | ICFCN4TRX[b] | $60F2_H$ | INTFCN4TRX | Transmit interrupt | FCN4 | 122 | $0810_H$ | $0810_H$ |
| 122 | ICFCN5ERR[b] | $60F4_H$ | INTFCN5ERR | Error interrupt | FCN5 | 123 | $0820_H$ | $0820_H$ |
| 123 | ICFCN5REC[b] | $60F6_H$ | INTFCN5REC | Receive interrupt | FCN5 | 124 | $0830_H$ | $0830_H$ |
| 124 | ICFCN5TRX[b] | $60F8_H$ | INTFCN5TRX | Transmit interrupt | FCN5 | 125 | $0840_H$ | $0840_H$ |
| 125 to 127 | – | $60FA_H$ to $60FE_H$ | reserved | | | 126 to 128 | $0850_H$ to $0870_H$ | $0850_H$ to $0870_H$ |

a) In µPD70F4177, the target of this interrupt is FCN[2:0].
b) This interrupt is not available in µPD70F4177.

### (4) V850E2/FG4-G EI level maskable interrupts

Table 4-12 V850E2/FG4-G EI level maskable interrupt requests (1/3)

| Interrupt | | | Interrupt request | | | Default priority | Exception code | Handler address offset |
|---|---|---|---|---|---|---|---|---|
| Channel | Control register | | Name | Cause | Unit | | | |
| | Name | Address FFFF... | | | | | | |
| 0 | ICWDTA0 | $6000_H$ | INTWDTA0 | WDTA0 75% interrupt | WDTA0 | 1 | $0080_H$ | $0080_H$ |
| 1 | ICWDTA1 | $6002_H$ | INTWDTA1 | WDTA1 75% interrupt | WDTA1 | 2 | $0090_H$ | $0090_H$ |
| 2 | ICLVI | $6004_H$ | INTLVI | LVI interrupt | LVI | 3 | $00A0_H$ | $00A0_H$ |
| 3 to 8 | – | $6006_H$ to $6010_H$ | reserved | | | 4 to 9 | $00B0_H$ to $0100_H$ | $00B0_H$ to $0100_H$ |
| 9 | ICP0 | $6012_H$ | INTP0 | Edge detection interrupt | Port | 10 | $0110_H$ | $0110_H$ |
| 10 | ICP1 | $6014_H$ | INTP1 | Edge detection interrupt | Port | 11 | $0120_H$ | $0120_H$ |
| 11 | ICP2 | $6016_H$ | INTP2 | Edge detection interrupt | Port | 12 | $0130_H$ | $0130_H$ |
| 12 | ICP3 | $6018_H$ | INTP3 | Edge detection interrupt | Port | 13 | $0140_H$ | $0140_H$ |
| 13 | ICP4 | $601A_H$ | INTP4 | Edge detection interrupt | Port | 14 | $0150_H$ | $0150_H$ |
| 14 | ICP5 | $601C_H$ | INTP5 | Edge detection interrupt | Port | 15 | $0160_H$ | $0160_H$ |
| 15 | ICP6 | $601E_H$ | INTP6 | Edge detection interrupt | Port | 16 | $0170_H$ | $0170_H$ |
| 16 | ICP7 | $6020_H$ | INTP7 | Edge detection interrupt | Port | 17 | $0180_H$ | $0180_H$ |
| 17 | ICP8 | $6022_H$ | INTP8 | Edge detection interrupt | Port | 18 | $0190_H$ | $0190_H$ |
| 18 | ICP9 | $6024_H$ | INTP9 | Edge detection interrupt | Port | 19 | $01A0_H$ | $01A0_H$ |
| 19 | ICP10 | $6026_H$ | INTP10 | Edge detection interrupt | Port | 20 | $01B0_H$ | $01B0_H$ |
| 20 | ICTAUB0I0 | $6028_H$ | INTTAUB0I0 | Interrupt for channel 0 | TAUB0 | 21 | $01C0_H$ | $01C0_H$ |
| 21 | ICTAUB0I1 | $602A_H$ | INTTAUB0I1 | Interrupt for channel 1 | TAUB0 | 22 | $01D0_H$ | $01D0_H$ |
| 22 | ICTAUB0I2 | $602C_H$ | INTTAUB0I2 | Interrupt for channel 2 | TAUB0 | 23 | $01E0_H$ | $01E0_H$ |
| 23 | ICTAUB0I3 | $602E_H$ | INTTAUB0I3 | Interrupt for channel 3 | TAUB0 | 24 | $01F0_H$ | $01F0_H$ |
| 24 | ICTAUB0I4 | $6030_H$ | INTTAUB0I4 | Interrupt for channel 4 | TAUB0 | 25 | $0200_H$ | $0200_H$ |
| 25 | ICTAUB0I5 | $6032_H$ | INTTAUB0I5 | Interrupt for channel 5 | TAUB0 | 26 | $0210_H$ | $0210_H$ |
| 26 | ICTAUB0I6 | $6034_H$ | INTTAUB0I6 | Interrupt for channel 6 | TAUB0 | 27 | $0220_H$ | $0220_H$ |
| 27 | ICTAUB0I7 | $6036_H$ | INTTAUB0I7 | Interrupt for channel 7 | TAUB0 | 28 | $0230_H$ | $0230_H$ |
| 28 | ICTAUB0I8 | $6038_H$ | INTTAUB0I8 | Interrupt for channel 8 | TAUB0 | 29 | $0240_H$ | $0240_H$ |
| 29 | ICTAUB0I9 | $603A_H$ | INTTAUB0I9 | Interrupt for channel 9 | TAUB0 | 30 | $0250_H$ | $0250_H$ |
| 30 | ICTAUB0I10 | $603C_H$ | INTTAUB0I10 | Interrupt for channel 10 | TAUB0 | 31 | $0260_H$ | $0260_H$ |
| 31 | ICTAUB0I11 | $603E_H$ | INTTAUB0I11 | Interrupt for channel 11 | TAUB0 | 32 | $0270_H$ | $0270_H$ |
| 32 | ICTAUB0I12 | $6040_H$ | INTTAUB0I12 | Interrupt for channel 12 | TAUB0 | 33 | $0280_H$ | $0280_H$ |
| 33 | ICTAUB0I13 | $6042_H$ | INTTAUB0I13 | Interrupt for channel 13 | TAUB0 | 34 | $0290_H$ | $0290_H$ |
| 34 | ICTAUB0I14 | $6044_H$ | INTTAUB0I14 | Interrupt for channel 14 | TAUB0 | 35 | $02A0_H$ | $02A0_H$ |
| 35 | ICTAUB0I15 | $6046_H$ | INTTAUB0I15 | Interrupt for channel 15 | TAUB0 | 36 | $02B0_H$ | $02B0_H$ |
| 36 to 51 | – | $6048_H$ to $6066_H$ | reserved | | | 37 to 52 | $02C0_H$ to $03B0_H$ | $02C0_H$ to $03B0_H$ |
| 52 | ICADCA0ERR | $6068_H$ | INTADCA0ERR | Error interrupt | ADCA0 | 53 | $03C0_H$ | $03C0_H$ |
| 53 | ICADCA0I0 | $606A_H$ | INTADCA0I0 | End of CG0 conversion | ADCA0 | 54 | $03D0_H$ | $03D0_H$ |
| 54 | ICADCA0I1 | $606C_H$ | INTADCA0I1 | End of CG1 conversion | ADCA0 | 55 | $03E0_H$ | $03E0_H$ |
| 55 | ICADCA0I2 | $606E_H$ | INTADCA0I2 | End of CG2 conversion | ADCA0 | 56 | $03F0_H$ | $03F0_H$ |
| 56 | ICADCA0LLT | $6070_H$ | INTADCA0LLT | Conversion interrupt | ADCA0 | 57 | $0400_H$ | $0400_H$ |

**Table 4-12 V850E2/FG4-G EI level maskable interrupt requests (2/3)**

| Interrupt | | | Interrupt request | | | Default priority | Exception code | Handler address offset |
|---|---|---|---|---|---|---|---|---|
| Channel | Control register | | Name | Cause | Unit | | | |
| | Name | Address FFFF... | | | | | | |
| 57 | ICFCNWUP | $6072_H$ | INTFCNWUP | Wake up interrupt | FCN [5:0][a] | 58 | $0410_H$ | $0410_H$ |
| 58 | – | $6074_H$ | reserved | | | 59 | $0420_H$ | $0420_H$ |
| 59 | – | $6076_H$ | reserved | | | 60 | $0430_H$ | $0430_H$ |
| 60 | ICFCN0ERR | $6078_H$ | INTFCN0ERR | Error interrupt | FCN0 | 61 | $0440_H$ | $0440_H$ |
| 61 | ICFCN0REC | $607A_H$ | INTFCN0REC | Receive interrupt | FCN0 | 62 | $0450_H$ | $0450_H$ |
| 62 | ICFCN0TRX | $607C_H$ | INTFCN0TRX | Transmit interrupt | FCN0 | 63 | $0460_H$ | $0460_H$ |
| 63 | ICCSIG0IRE | $607E_H$ | INTCSIG0IRE | Reception error interrupt | CSIG0 | 64 | $0470_H$ | $0470_H$ |
| 64 | ICCSIG0IR | $6080_H$ | INTCSIG0IR | Reception status interrupt | CSIG0 | 65 | $0480_H$ | $0480_H$ |
| 65 | ICCSIG0IC | $6082_H$ | INTCSIG0IC | Communication status interrupt | CSIG0 | 66 | $0490_H$ | $0490_H$ |
| 66 | ICDMA0 | $6084_H$ | INTDMA0 | DMA channel 0 transfer completion | DMA | 67 | $04A0_H$ | $04A0_H$ |
| 67 | ICDMA1 | $6086_H$ | INTDMA1 | DMA channel 1 transfer completion | DMA | 68 | $04B0_H$ | $04B0_H$ |
| 68 | ICDMA2 | $6088_H$ | INTDMA2 | DMA channel 2 transfer completion | DMA | 69 | $04C0_H$ | $04C0_H$ |
| 69 | ICDMA3 | $608A_H$ | INTDMA3 | DMA channel 3 transfer completion | DMA | 70 | $04D0_H$ | $04D0_H$ |
| 70 | ICDMA4 | $608C_H$ | INTDMA4 | DMA channel 4 transfer completion | DMA | 71 | $04E0_H$ | $04E0_H$ |
| 71 | ICDMA5 | $608E_H$ | INTDMA5 | DMA channel 5 transfer completion | DMA | 72 | $04F0_H$ | $04F0_H$ |
| 72 | ICDMA6 | $6090_H$ | INTDMA6 | DMA channel 6 transfer completion | DMA | 73 | $0500_H$ | $0500_H$ |
| 73 | ICDMA7 | $6092_H$ | INTDMA7 | DMA channel 7 transfer completion | DMA | 74 | $0510_H$ | $0510_H$ |
| 74 | ICIICB0IA | $6094_H$ | INTIICB0IA | Data transmission/reception interrupt | IICB0 | 75 | $0520_H$ | $0520_H$ |
| 75 | ICFCN1ERR | $6096_H$ | INTFCN1ERR | Error interrupt | FCN1 | 76 | $0530_H$ | $0530_H$ |
| 76 | ICFCN1REC | $6098_H$ | INTFCN1REC | Receive interrupt | FCN1 | 77 | $0540_H$ | $0540_H$ |
| 77 | ICFCN1TRX | $609A_H$ | INTFCN1TRX | Transmit interrupt | FCN1 | 78 | $0550_H$ | $0550_H$ |
| 78 | ICTAUJ0I0 | $609C_H$ | INTTAUJ0I0 | Interrupt for channel 0 | TAUJ0 | 79 | $0560_H$ | $0560_H$ |
| 79 | ICTAUJ0I1 | $609E_H$ | INTTAUJ0I1 | Interrupt for channel 1 | TAUJ0 | 80 | $0570_H$ | $0570_H$ |
| 80 | ICTAUJ0I2 | $60A0_H$ | INTTAUJ0I2 | Interrupt for channel 2 | TAUJ0 | 81 | $0580_H$ | $0580_H$ |
| 81 | ICTAUJ0I3 | $60A2_H$ | INTTAUJ0I3 | Interrupt for channel 3 | TAUJ0 | 82 | $0590_H$ | $0590_H$ |
| 82 | ICOSTM0 | $60A4_H$ | INTOSTM0 | OSTM0 interrupt | OSTM0 | 83 | $05A0_H$ | $05A0_H$ |
| 83 | ICCSIG4IRE | $60A6_H$ | INTCSIG4IRE | Reception error interrupt | CSIG4 | 84 | $05B0_H$ | $05B0_H$ |
| 84 | ICCSIG4IR | $60A8_H$ | INTCSIG4IR | Reception status interrupt | CSIG4 | 85 | $05C0_H$ | $05C0_H$ |
| 85 | ICCSIG4IC | $60AA_H$ | INTCSIG4IC | Communication status interrupt | CSIG4 | 86 | $05D0_H$ | $05D0_H$ |
| 86 to 88 | – | $60AC_H$ to $60B0_H$ | reserved | | | 87 to 89 | $05E0_H$ to $0600_H$ | $05E0_H$ to $0600_H$ |
| 89 | ICLMA2IS | $60B2_H$ | INTLMA2IS | Status interrupt | LMA2 | 90 | $0610_H$ | $0610_H$ |
| 90 | ICLMA2IR | $60B4_H$ | INTLMA2IR | Reception completion interrupt | LMA2 | 91 | $0620_H$ | $0620_H$ |
| 91 | ICLMA2IT | $60B6_H$ | INTLMA2IT | Transmission interrupt | LMA2 | 92 | $0630_H$ | $0630_H$ |
| 92 to 97 | – | $60B8_H$ to $60C2_H$ | reserved | | | 93 to 98 | $0640_H$ to $0690_H$ | $0640_H$ to $0690_H$ |
| 98 | ICCSIG7IRE | $60C4_H$ | INTCSIG7IRE | Reception error interrupt | CSIG7 | 99 | $06A0_H$ | $06A0_H$ |
| 99 | ICCSIG7IR | $60C6_H$ | INTCSIG7IR | Reception status interrupt | CSIG7 | 100 | $06B0_H$ | $06B0_H$ |
| 100 | ICCSIG7IC | $60C8_H$ | INTCSIG7IC | Communication status interrupt | CSIG7 | 101 | $06C0_H$ | $06C0_H$ |

**Table 4-12 V850E2/FG4-G EI level maskable interrupt requests (3/3)**

| Interrupt | | | Interrupt request | | | Default priority | Exception code | Handler address offset |
|---|---|---|---|---|---|---|---|---|
| Channel | Control register | | Name | Cause | Unit | | | |
| | Name | Address FFFF... | | | | | | |
| 101 | ICP11 | $60CA_H$ | INTP11 | Edge detection interrupt | Port | 102 | $06D0_H$ | $06D0_H$ |
| 102 | ICP12 | $60CC_H$ | INTP12 | Edge detection interrupt | Port | 103 | $06E0_H$ | $06E0_H$ |
| 103 to 105 | – | $60CE_H$ to $60D2_H$ | reserved | | | 104 to 106 | $06F0_H$ to $0710_H$ | $06F0_H$ to $0710_H$ |
| 106 | ICKR0 | $60D4_H$ | INTKR0 | Key return interrupt | KR0 | 107 | $0720_H$ | $0720_H$ |
| 107 | ICLMA10IS | $60D6_H$ | INTLMA10IS | Status interrupt | LMA10 | 108 | $0730_H$ | $0730_H$ |
| 108 | ICLMA10IR | $60D8_H$ | INTLMA10IR | Reception completion interrupt | LMA10 | 109 | $0740_H$ | $0740_H$ |
| 109 | ICLMA10IT | $60DA_H$ | INTLMA10IT | Transmission interrupt | LMA10 | 110 | $0750_H$ | $0750_H$ |
| 110 | ICLMA11IS | $60DC_H$ | INTLMA11IS | Status interrupt | LMA11 | 111 | $0760_H$ | $0760_H$ |
| 111 | ICLMA11IR | $60DE_H$ | INTLMA11IR | Reception completion interrupt | LMA11 | 112 | $0770_H$ | $0770_H$ |
| 112 | ICLMA11IT | $60E0_H$ | INTLMA11IT | Transmission interrupt | LMA11 | 113 | $0780_H$ | $0780_H$ |
| 113 | ICFCN2ERR | $60E2_H$ | INTFCN2ERR | Error interrupt | FCN2 | 114 | $0790_H$ | $0790_H$ |
| 114 | ICFCN2REC | $60E4_H$ | INTFCN2REC | Receive interrupt | FCN2 | 115 | $07A0_H$ | $07A0_H$ |
| 115 | ICFCN2TRX | $60E6_H$ | INTFCN2TRX | Transmit interrupt | FCN2 | 116 | $07B0_H$ | $07B0_H$ |
| 116 | ICFCN3ERR[b] | $60E8_H$ | INTFCN3ERR | Error interrupt | FCN3 | 117 | $07C0_H$ | $07C0_H$ |
| 117 | ICFCN3REC[b] | $60EA_H$ | INTFCN3REC | Receive interrupt | FCN3 | 118 | $07D0_H$ | $07D0_H$ |
| 118 | ICFCN3TRX[b] | $60EC_H$ | INTFCN3TRX | Transmit interrupt | FCN3 | 119 | $07E0_H$ | $07E0_H$ |
| 119 | ICFCN4ERR[b] | $60EE_H$ | INTFCN4ERR | Error interrupt | FCN4 | 120 | $07F0_H$ | $07F0_H$ |
| 120 | ICFCN4REC[b] | $60F0_H$ | INTFCN4REC | Receive interrupt | FCN4 | 121 | $0800_H$ | $0800_H$ |
| 121 | ICFCN4TRX[b] | $60F2_H$ | INTFCN4TRX | Transmit interrupt | FCN4 | 122 | $0810_H$ | $0810_H$ |
| 122 | ICFCN5ERR[b] | $60F4_H$ | INTFCN5ERR | Error interrupt | FCN5 | 123 | $0820_H$ | $0820_H$ |
| 123 | ICFCN5REC[b] | $60F6_H$ | INTFCN5REC | Receive interrupt | FCN5 | 124 | $0830_H$ | $0830_H$ |
| 124 | ICFCN5TRX[b] | $60F8_H$ | INTFCN5TRX | Transmit interrupt | FCN5 | 125 | $0840_H$ | $0840_H$ |
| 125 to 127 | – | $60FA_H$ to $60FE_H$ | reserved | | | 126 to 128 | $0850_H$ to $0870_H$ | $0850_H$ ?0870 H |

a) In µPD70F4179, the target of this interrupt is FCN[2:0].
b) This interrupt is not available in µPD70F4179.

### 4.3.2 FE level non-maskable interrupt sharing

The FE level non-maskable interrupt FENMI is shared among several interrupt sources.

#### (1) FENMIF - FENMI factor register

This register contains information about which interrupt has generated the FE level non-maskable interrupt FENMI.

**Access** This register can be read in 32-bit units.

**Address** FF45 $0000_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by any reset.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | WDTA1 NMIF | WDTA0 NMIF | NMI0F |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 4-13 FENMIF register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | WDTA1NMIF | Watchdog Timer WDTA1NMIF flag<br>0: no WDTA1NMIF occured<br>1: WDTA1NMIF has occured |
| 1 | WDTA0NMIF | Watchdog Timer WDTA0NMIF flag<br>0: no WDTA0NMIF occured<br>1: WDTA0NMIF has occured |
| 0 | NMI0F | Port interrupt NMI0 flag<br>0: no NMI0 occured<br>1: NMI0 has occured |

### (2) FENMIFC - FENMI factor clear register

This register clears the FE level non-maskable flags of the FENMIF register.

**Access** This register can be written in 32-bit units.

**Address** FF45 0008$_H$

**Initial Value** Reading this registers returns always 0000 0000$_H$.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | WDTA1 NMIFC | WDTA0 NMIFC | NMI0FC |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 4-14 FENMIFC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | WDTA1NMIFC | Watchdog Timer WDTA1NMIF flag clear<br>0: no function<br>1: clear FENMIF.WDTA1NMIF |
| 1 | WDTA0NMIFC | Watchdog Timer WDTA0NMIF flag clear<br>0: no function<br>1: clear FENMIF.WDTA0NMIF |
| 0 | NMI0FC | Port interrupt NMI0 flag clear<br>0: no function<br>1: clear FENMIF.NMI0F |

## 4.4 External Interrupts

### 4.4.1 Edge Detection Configuration

The external interrupts can be configured to generate an interrupt request upon a rising or falling edge or upon both edges of the external pin.

Following registers are used to specify the edge:

**Table 4-15 External interrupt edge detection registers**

| Interrupt | Register |
|---|---|
| INTP0 | FCLA0CTL0 |
| INTP1 | FCLA0CTL1 |
| INTP2 | FCLA0CTL2 |
| INTP3 | FCLA0CTL3 |
| INTP4 | FCLA0CTL4 |
| INTP5 | FCLA0CTL5 |
| INTP6 | FCLA0CTL6 |
| INTP7 | FCLA0CTL7 |
| INTP8 | FCLA1CTL0 |
| INTP9 | FCLA1CTL1 |
| INTP10 | FCLA1CTL2 |
| INTP11 | FCLA1CTL3 |
| INTP12 | FCLA1CTL4 |
| NMI | FCLA2CTL0 |

Refer to the section *“Port filters assignment”* in the chapter *“Port Functions”* for details of these registers.

### 4.4.2 External interrupts as trigger and wake-up signals

The external interrupts can be used as

- DMA trigger factors
- A/D Converter H/W triggers
- stand-by wake-up factors

**Table 4-16 External interrupts as triggers and wake-up**

| Interrupt | DMA trigger factor[a] | ADAAn H/W trigger[b] | Stand-by wake-up factor[c] |
|---|---|---|---|
| INTP0 | 1 | ADAA0TTIN201 | WUFLm07 |
| INTP1 | 2 | ADAA0TTIN101 | WUFLm08 |
| INTP2 | 3 | ADAA0TTIN001 | WUFLm09 |
| INTP3 | 4 | ADAA0TTIN202 | WUFLm10 |
| INTP4 | 5 | ADAA0TTIN102 | WUFLm11 |
| INTP5 | 6 | ADAA0TTIN002 | WUFLm12 |
| INTP6 | 7 | – | WUFLm13 |
| INTP7 | 8 | – | WUFLm14 |
| INTP8 | 9 | – | WUFLm15 |
| INTP9 | 10 | – | WUFLm16 |
| INTP10 | 11 | – | WUFLm17 |
| INTP11 | – | – | WUFLm18 |
| INTP12 | – | – | WUFLm19 |
| NMI | – | – | WUFLm00 |

a) Refer to the chapter *5 “DMA Controller (DMAC)” on page 196* for details.
b) Refer to the chapter *22 “A/D Converter (ADAA)” on page 1143* for details.
c) The stand-by wake-up factors are indicated by their respective bit in the wake-up factor register WUFLm. refer to the chapter *8 “Stand-by Controller (STBC)” on page 322* for details.

**Stand-by wake-up**

To use an external interrupt as a wake-up factor from DEEPSTOP mode, its analog filter must be set to edge detection mode (FCLAnINTm = 0).
Refer to the section *“Port Filters Functional Description”* in the *“Port Functions”* chapter for details.

## 4.5 Interrupt Controller Control Registers

### (1) ICn - EI level interrupt control registers (n = 0 to 255)

These registers, each of which is for a channel n of EI level maskable interrupt INTn, are used to set a condition to control each channel.

**Caution** Do not access ICn registers of interrupt channels, not listed in above interrupt request tables.

**Access** These registers can be read/written in

- 16-bit units
  – via 16-bit registers ICn
- 8-bit or 1-bit units
  – bits 7 to 0 via 8-bit registers ICnL
  – bits 15 to 8 via 8-bit registers ICnH.

**Address** ICn: FFFF $6000_H$ + 2n
ICnL: FFFF $6000_H$ + 2n, ICnH: FFFF $6001_H$ + 2n

**Initial Value** $008F_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | RFn | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| MKn | 0 | 0 | 0 | 0 | P[2:0]n | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-17 ICn register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 12 | RFn | Interrupt request flag<br>RFn is set upon occurrence of an EI level maskable interrupt INTn.<br>RFn can also be set by the application program, which generates also an EI level maskable interrupt.<br>0: no interrupt request<br>1: interrupt request<br>RFn is reset if the interrupt is acknowledged, i.e. the interrupt service routine is started.<br>RFn can also be reset by the application program. |

**Table 4-17 ICn register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | MKn | Interrupt mask bit<br>When the MKn bit is set, the interrupt request set to the interrupt request flag RFn is masked, so that the interrupt request is not issued from that channel to the CPU. From a channel with the MKn bit set, interrupt pending status is not displayed by the ICSR.PMF bit. The MKn bit does not mask a signal input from an interrupt input itself and, therefore, the corresponding interrupt request flag is set even when the MKn bit is set. The setting of the corresponding bit of the interrupt mask register IMR is also reflected.<br>0: enables interrupt servicing<br>1: disables interrupt servicing |
| 2 to 0 | P[2:0]n | These bits specify 8 levels of interrupt priorities. The highest priority is 0 and the lowest is 7.<br>If two or more interrupt requests of EI level are generated at the same time, the interrupt source having the higher priority specified by these bits is selected and reported to the CPU. If the priority specified by the P[2:0]n bits is the same, the source having the lower channel number is selected by a fixed priority. |

**(2) IMRm - EI level interrupt mask registers (m = 0 to 15)**

These registers are a collection of the MKn bits of the ICn registers. Each bit of IMRm reflects the setting of the corresponding ICn.MKn bit. Setting IMRm is reflected in the corresponding MKn bit.

**Caution** MKn bits for interrupt channels, not listed in above interrupt request tables, must be set to 1.

**Access** These registers can be read/written in:

- 16-bit units
  – via 16-bit registers IMRm
- 8-bit or 1-bit units
  – bits 7 to 0 via 8-bit registers IMRmL
  – bits 15 to 8 via 8-bit registers IMRmH.

**Address**

IMR0: FFFF 6400$_H$ IMR1: FFFF 6402$_H$
IMR2: FFFF 6404$_H$ IMR3: FFFF 6406$_H$
IMR4: FFFF 6408$_H$ IMR5: FFFF 640A$_H$
IMR6: FFFF 640C$_H$ IMR7: FFFF 640E$_H$
IMR8: FFFF 6410$_H$ IMR9: FFFF 6412$_H$
IMR10: FFFF 6414$_H$ IMR11: FFFF 6416$_H$
IMR12: FFFF 6418$_H$ IMR13: FFFF 641A$_H$
IMR14: FFFF 641C$_H$ IMR15: FFFF 641E$_H$
IMRmL: address of IMRm IMRmH: address of IMRm + 1

**Initial Value** FFFF$_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| IMRmMK m x 16 + 15 | IMRmMK m x 16 + 14 | IMRmMK m x 16 + 13 | IMRmMK m x 16 + 12 | IMRmMK m x 16 + 11 | IMRmMK m x 16 + 10 | IMRmMK m x 16 + 9 | IMRmMK m x 16 + 8 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IMRmMK m x 16 + 7 | IMRmMK m x 16 + 6 | IMRmMK m x 16 + 5 | IMRmMK m x 16 + 4 | IMRmMK m x 16 + 3 | IMRmMK m x 16 + 2 | IMRmMK m x 16 + 1 | IMRmMK m x 16 + 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-18 IMRm registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | IMRmMK255 to IMRmMK0 | These bits mask an interrupt from channels n = 0 to 255 of EI level maskable interrupt INTn.<br>0: interrupt request issued<br>1: interrupt request not issued |

### (3) ISPR - In-service priority register

This register holds the interrupt priority of an EI level maskable interrupt INTn that is being processed by the CPU.

**ISPR[7:0] setting**

When the CPU enters an interrupt service routine, i.e. acknowledges the interrupt, with interrupt request priority ICn.P[2:0]n = 0 to 7, the bit ISPR[7:0] of this register, corresponding to this priority, is set.

**ISPR[7:0] clearing**

When the CPU has completed the interrupt service routine by the EIRET instruction, the ISPR[7:0] bit, that was set upon entering this interrupt service routine, is cleared.
Since the interrupt service routine with the highest priority is always completed first, the cleared bit is always the one, that corresponds to the highest priority.

**Nested interrupt services**

If multiple EI level maskable interrupts with different priorities ICn.P[2:0]n have been acknowledged by the CPU (lower prioritized interrupt service routines have been interrupted by higher prioritized), the corresponding ISPR[7:0] bits of all interrupts in service are sequentially set.

**ISPR[7:0] clearing by software**

Clearing this register is protected by a special sequence, involving the in-service priority clear register ISPC register.
All ISPR[7:0] bits can be cleared only all together by use of ISPC. For that purpose ISPC has to be written with $FFFF_H$, followed by writing $0000_H$ to ISPR.
Note that it is not possible to clear single ISPR[7:0] bits.
Refer also to *5 “ISPC - In-service priority clear register” on page 176* .

**Access**

This register can be accessed as follows:

- writing $0000_H$ via the 16-bit register ISPR for clearing all ISPR[7:0], after ISPC has been written with $FFFF_H$
- reading
  – in 16-bit units via the 16-bit register ISPR
  – ISPR[7:0] in 8-bit units via the 8-bit register ISPRL

**Address**

ISPR: $FFFF\ 6440_H$
ISPRL: $FFFF\ 6440_H$

**Initial Value**

$0000_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ISPR7 | ISPR6 | ISPR5 | ISPR4 | ISPR3 | ISPR2 | ISPR1 | ISPR0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-19 ISPR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | ISPR7 to ISPR0 | These bits indicate the priority of the interrupt being acknowledged.<br>0: Interrupt request of the priority corresponding to the bit position is not acknowledged.<br>1: Interrupt request of the priority corresponding to the bit position is being processed by the CPU. |

### (4) PMR - Priority mask register

This register specifies an interrupt priority by which an interrupt request flag of EI level maskable interrupt INTn is to be masked. It disables all at once the interrupt requests from the INTn channel for which the interrupt priority specified by this register is set.

The position of each bit of this register corresponds to an interrupt priority. For example, if 1 is set to bit 0, channel of interrupt priority 0 can be masked.

**Access** This register can be read/written in

- in 16-bit units via the 16-bit register PMR
- PMR[7:0] in 8-bit or 1-bit units via the 8-bit register PMRL.

**Address** PMR: FFFF 6448$_H$
PMRL: FFFF 6448$_H$

**Access** This register can be read/written in 16-bit or 8-bit or 1-bit units.

**Address** FFFF 6448$_H$

**Initial Value** 0000$_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PMR7 | PMR6 | PMR5 | PMR4 | PMR3 | PMR2 | PMR1 | PMR0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-20 PMR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | PMR7 to PMR0 | These bits specify an interrupt priority by which an interrupt request flag<br>0: Enables interrupt servicing of the priority corresponding to a specified bit position.<br>1: Disables interrupt servicing of the priority corresponding to a specified bit position. |

### (5) ISPC - In-service priority clear register

This register prepares to clear the in-service priority register ISPR.

For clearing ISPR proceed as follows:

1. write $FFFF_H$ to ISPC
2. write $0000_H$ to ISPR

As a result, the internal mode registers of the Interrupt Controller for interrupt servicing, which indicate that an interrupt request is being processed by the CPU, are cleared.

At the same time, all the processing modes of FE level NMI, FE level maskable interrupt FEINT, and EI level maskable interrupts INTn of the ICSR register are canceled.

After writing $FFFF_H$ to ISPC, $FFFF_H$ is also read back until ISPR has been cleared by writing ISPR = $0000_H$. Afterwards ISPC is automatically cleared to $0000_H$.
Before writing ISPR = $0000_H$, ISPC can be reset by writing $0000_H$, and thus disabling any ISPR clear attempt.

ISPC can only be written with $FFFF_H$ or $0000_H$. Writing any other value does not change the register content.

Refer also to *3 “ISPR - In-service priority register” on page 174* .

**Caution** Writing $FFFF_H$ to ISPC and $0000_H$ to ISPR in order to clear ISPR does not need to maintain any time relation to each other. In particular this means, that any kind of other processes may take place beween both write accesses.

**Access** This register can be read/written in 16-bit units.

**Address** FFFF $6450_H$

**Initial Value** $0000_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| ISPC[15:8] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ISPC[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 4-21 ISPC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | ISPC[15:0] | ISPR clear preparation<br>• writing $FFFF_H$: enables ISPR to be cleared by ISPR = $0000_H$<br>• writing $0000_H$: disables ISPR to be cleared<br>If ISPR has been cleared after ISPC. = $FFFF_H$, ISPC returns to $0000_H$ automatically. |

### (6) SCR - Selected channel hold register

This register holds the channel number n of the EI level maskable interrupt INTn, acknowledged by the CPU.

**Access** This register can be read only in

- in 16-bit units:
  – via 16-bit register SCR
- in 8-bit units:
  – SCR[7:0] via 8-bit register SCRL.

**Address** SCR: FFFF 6458$_H$
SCRL: FFFF 6458$_H$

**Access** This register can be read only in 16-bit or 8-bit units.

**Address** FFFF 6458$_H$

**Initial Value** 0000$_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SCR[7:0] | | | | | | | |
| R | R | R | R | R | R | R | R |

**Table 4-22 SCR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | SCR[7:0] | Holds the channel number n of the maskable interrupt that has been acknowledged by the CPU. The value of these bits is updated when an interrupt vector is reported to the CPU. It is overwritten when multiple interrupts of EI level maskable interrupt (INTn) are acknowledged. These bits are not updated when an FE level interrupt is acknowledged.<br>Writing to this register is ignored. |

### (7) ICSR - Interrupt controller status register

This register indicates the operation status of the Interrupt Controller. Especially the FNE, FIE and EIE of this register serve as a mode register of interrupt servicing.

**Access** This register can be read only in

- 16-bit units:
  – via 16-bit register ICSR
- 8-bit or 1-bit units:
  – bits 7 to 0 via 8-bit register ICSRL
  – bits 15 to 8 via 8-bit register ICSRL

**Address** ICSR: FFFF 645A$_H$
ICSRL: FFFF 645A$_H$, ICSRH: FFFF 645B$_H$

**Initial Value** 0000$_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | PMF |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | FNR | FIR | EIR | 0 | FNE | FIE | EIE |
| R | R | R | R | R | R | R | R |

**Table 4-23 ICSR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 8 | PMF | Indicates 1 if the request flag of a channel n of EI level interrupt INTn, that has the interrupt priority prohibited by the setting of PMR from being serviced, is set. |
| 6 | FNR | Indicates 1 if an FE level non-maskable interrupt FENMI has been issued to the CPU. |
| 5 | FIR | Indicates 1 if an FE level maskable interrupt FEINT has been issued to the CPU. |
| 4 | EIR | Indicates 1 if an EI level maskable interrupt INTn has been issued to the CPU. |
| 2 | FNE | Indicates 1 if the CPU is processing the FE level non-maskable interrupt FENMI. |
| 1 | FIE | Indicates 1 if the CPU is processing the FE level maskable interrupt FEINT. |
| 0 | EIE | Indicates 1 if the CPU is processing the EI level maskable interrupt INTn. |

### (8) FNC - FE level NMI status register

This register informs about the occurence of a FE level non-maskable interrupt FENMI.

**Access** This register can be read only in

- 16-bit units:
  – via 16-bit register FNC
- 8-bit or 1-bit units:
  – bits 15 to 8 via 8-bit register FNCH

**Address** FNC: FFFF 645C$_H$
FNCH: FFFF 645D$_H$

**Initial Value** 0000$_H$. This register is initialized by any reset.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | FNRF | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R |

**Table 4-24 FNC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 12 | FNRF | FENMI interrupt request flag<br>0: no interrupt request FENMI<br>1: FENMI interrupt request occured |

### (9) FIC - FE level INT status register

This register informs about the occurence of a FE level maskable interrupt FEINT.

**Access** This register can be read only in

- 16-bit units:
  – via 16-bit register FIC
- 8-bit or 1-bit units:
  – bits 15 to 8 via 8-bit register FICH

**Address** FiC: FFFF 645E$_H$
FICH: FFFF 645F$_H$

**Initial Value** 0000$_H$. This register is initialized by any reset

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | FIRF | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R |

**Table 4-25 FIC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 12 | FIRF | FEINT interrupt request flag<br>0: no FEINT interrupt request<br>1: FEINT interrupt request occured |

## 4.6 Interrupt Acknowledgment and Restoring

This section describes the operation during interrupt acknowledgment and restoring from interrupt servicing.

**CPU registers** This section frequently refers to registers of the CPU. For a detailed description of these registers refer to the “V850E2 32-bit Microcontroller Core Architecture” user manual.

### 4.6.1 FE level non-maskable interrupt FENMI

When an FENMI interrupt is requested, an FE level non-maskable interrupt is generated in the CPU. This FE level non-maskable interrupt is used when a fatal system error occurs.

**Caution** Upon acknowledgment of the FENMI interrupt, generation of the next FENMI, FEINT, or INTn interrupt is pended until the FERET instruction is executed (interrupt request is acknowledged and held.)
FENMI can be acknowledged even when the NP bit of the program status word CPU register PSW is set to 1. Therefore, if the FENMI interrupt occurs during the processing of an FEINT exception, PPI exception, or other FE level exceptions, the save address is lost and cannot be restored. After a FENMI interrupt is requested and the required processing has been completed, execute a system reset, etc. Return to the original processing is not possible.

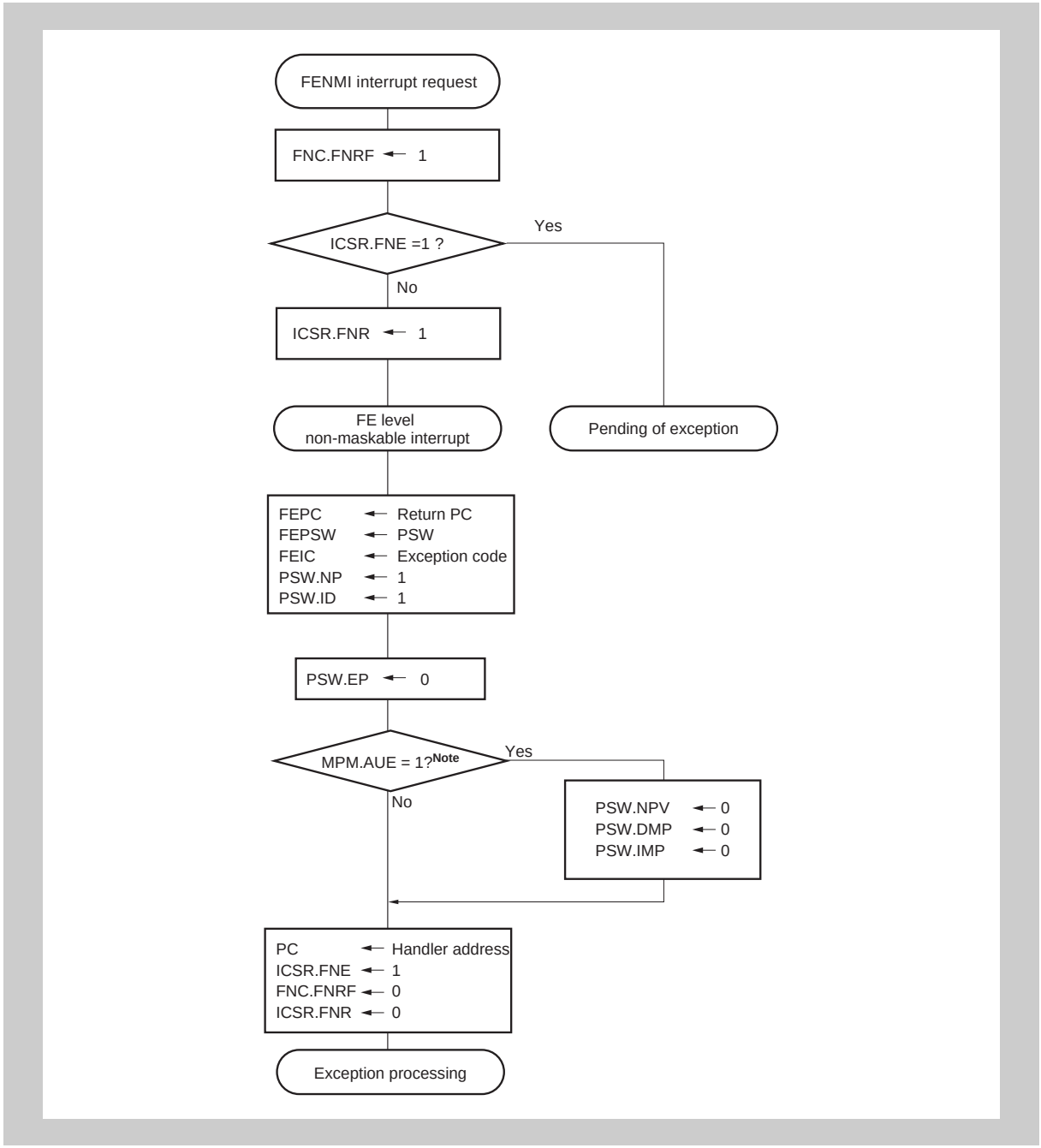



**Figure 4-1 Processing upon occurrence of FENMI interrupt request**

**Note** If a processor protection exception (MDP or MIP) occurs, the PSW.NPV, DMP, and IMP bits are always cleared, regardless of the status of MPM.AUE. MPM.AUE controls the execution level auto transition function, refer to the “V850E2 32-bit Microcontroller Core Architecture” (Document number R01US0037EJxxxx) for details.

**FENMI restore** An FE level non-maskable interrupt FENMI cannot be restored since such interrupt indicates a fatal system error.

Execute a system reset after exception processing.

### 4.6.2 FE level maskable interrupt FEINT

**Caution** A FE level maskable interrupt FEINT is recoverable.

Upon acknowledgment of the FEINT interrupt, generation of the next FEINT or INTn interrupt is kept pending until the FERET instruction is executed. Interrupt request is acknowledged and held.

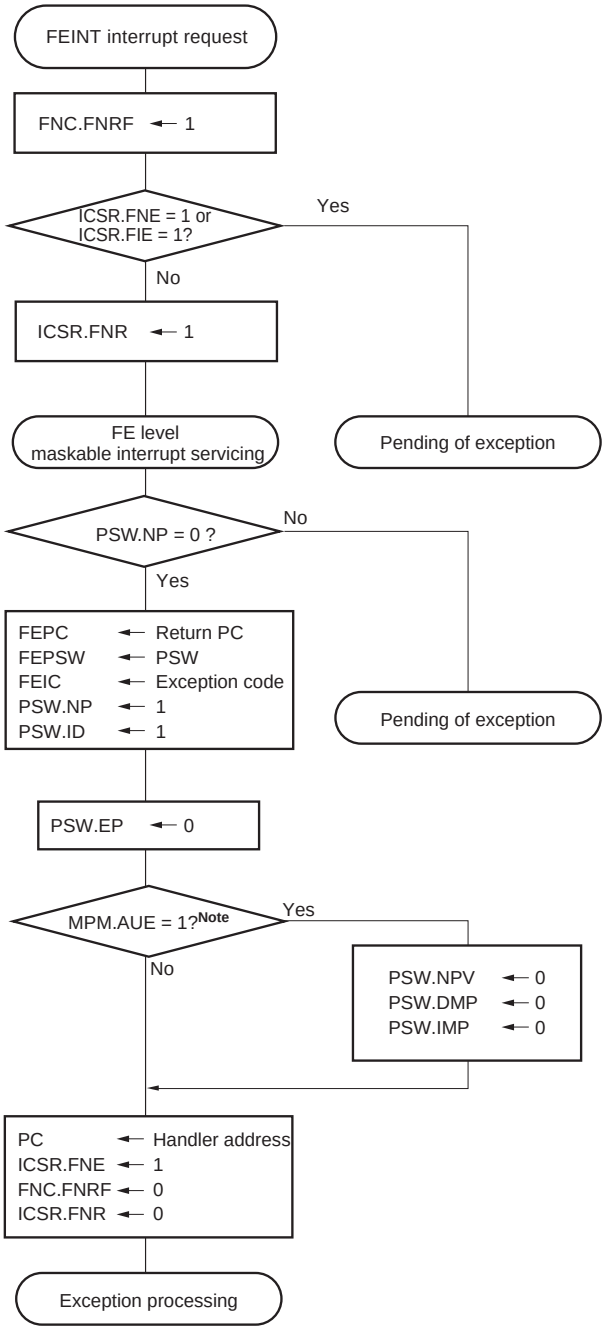



**Figure 4-2 Processing upon occurrence of FEINT interrupt request**

**Note** MPM.AUE controls the execution level auto transition function, refer to the “V850E2 32-bit Microcontroller Core Architecture” (Document number R01US0037EJxxxx) for details.

**FEINT restore** Restore from FE level maskable interrupt FEINT servicing is performed using the FERET instructions. Execution of the FERET instruction while the PSW.EP bit status is cleared launches restore processing from the FE level maskable interrupt interrupt FEINT. Completely restoring from interrupt servicing when the PSW.EP bit is "1" is not possible (clearing of the ICSR, ISPR, and other registers is not performed). For return from FE level maskable interrupts, execute the FERET instruction with the PSW.EP bit always cleared.

**Caution** In the V850E2 CPU, although RETI instructions are provided for backward compatibility with V850E1 and V850E2 architectures, but their use is, in principle, prohibited. Replace all RETI instructions other than existing programs that cannot be modified with EIRET or FERET instructions.

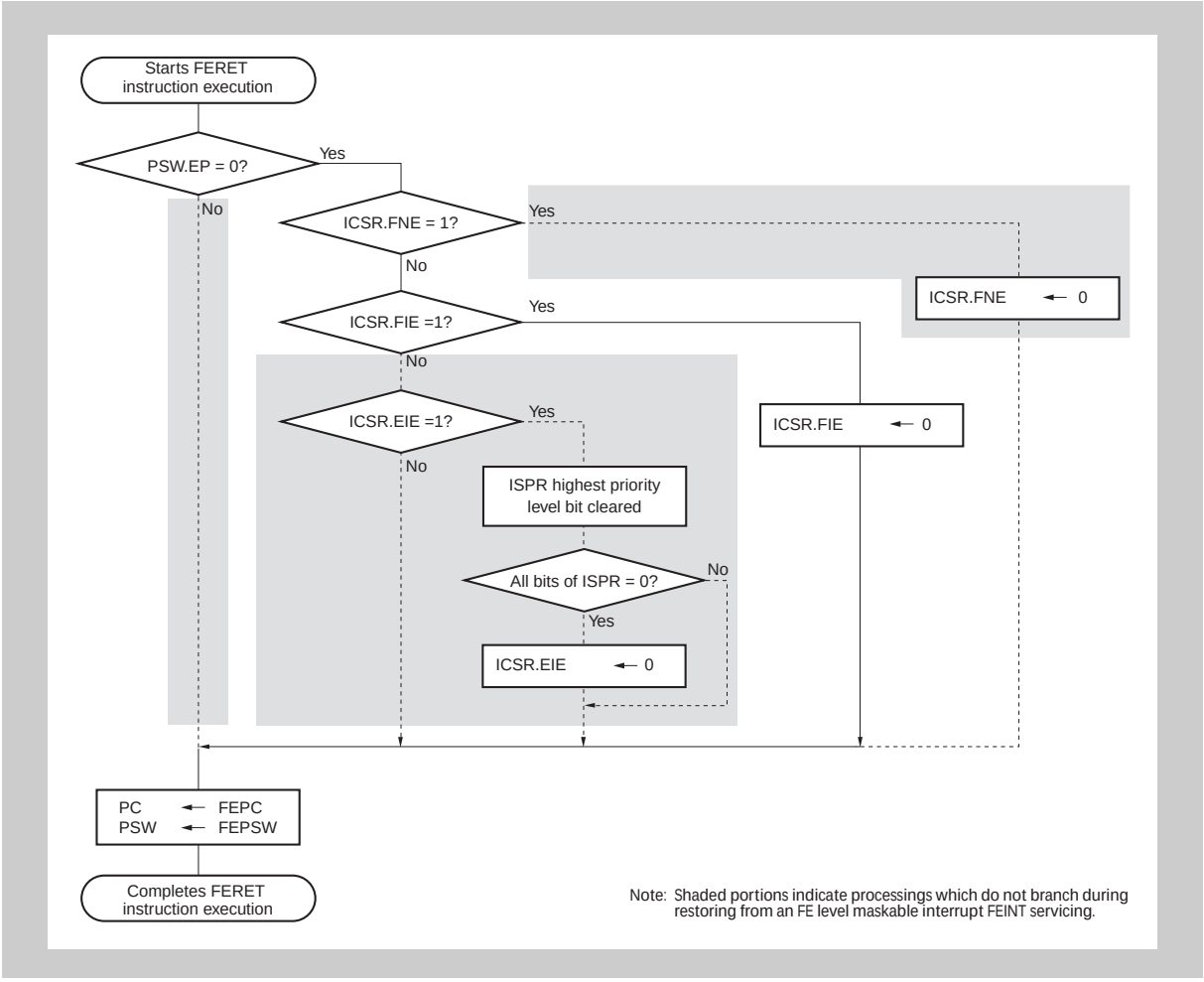



**Figure 4-3 Restore from FE level maskable interrupt (FEINT) servicing**

### 4.6.3 EI level maskable interrupt INTn

When an EI level maskable interrupt is requested, an INTn interrupt is requested to the CPU (the transition to the interrupt handler occurs from the setting of the IMR register of the Interrupt Controller). This interrupt is a recoverable EI level interrupt.

In the case of an INTn interrupt, its channel number n is set to the selected channel hold register SCR. As a result, the channel number can be easily known when wishing to share the same interrupt vectors among several channels.

**Caution** Upon acknowledgment of the EI level interrupt, the priority level of the currently acknowledged interrupt is registered to the in-service priority register ISPR. Then, until execution of the EIRET instruction, interrupt with a priority level lower than that of this ISPR register are not generated. Interrupt request is acknowledged and held.
Registration of the priority level of the currently acknowledged interrupt and deletion of the priority level of the interrupt during EIRET to/from the ISPR register are automatically performed by the hardware. Write to the ISPR register cannot be performed by software. Write operations are ignored.

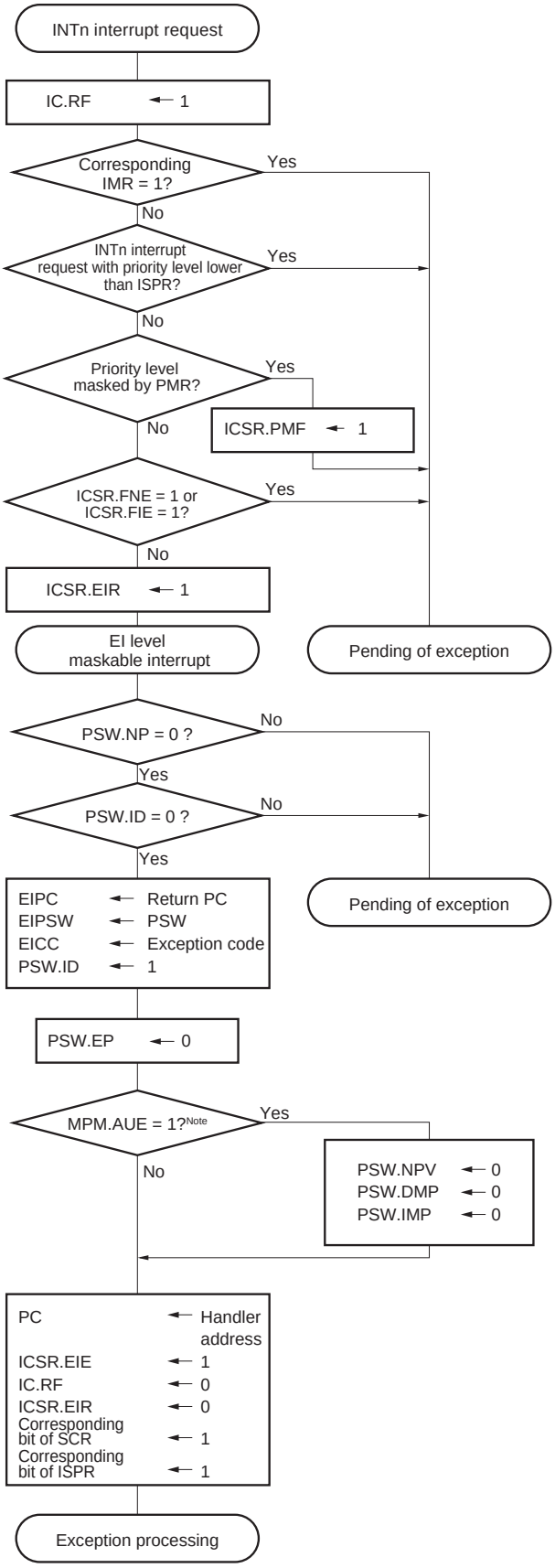



**Figure 4-4 Processing upon occurrence of INTn interrupt request**

**Note** MPM.AUE controls the execution level auto transition function, refer to the “V850E2 32-bit Microcontroller Core Architecture” (Document number R01US0037EJxxxx) for details.

**INTn restore** Restore from EI level maskable interrupt INTn is performed using the EIRET instruction. Execution of the EIRET instruction while the PSW.EP bit status is cleared launches restore processing from the interrupt. Completely restoring from interrupt servicing when the PSW.EP bit is "1" is not possible (clearing of the ICSR, ISPR, and other registers is not performed). For return from EI level maskable interrupt, execute the EIRET instruction with the PSW.EP bit always cleared.

**Caution** In the V850E2 CPU, although RETI instructions are provided for backward compatibility with V850E1 and V850E2 architectures, but their use is, in principle, prohibited. Replace all RETI instructions other than existing programs that cannot be modified with EIRET or FERET instructions.

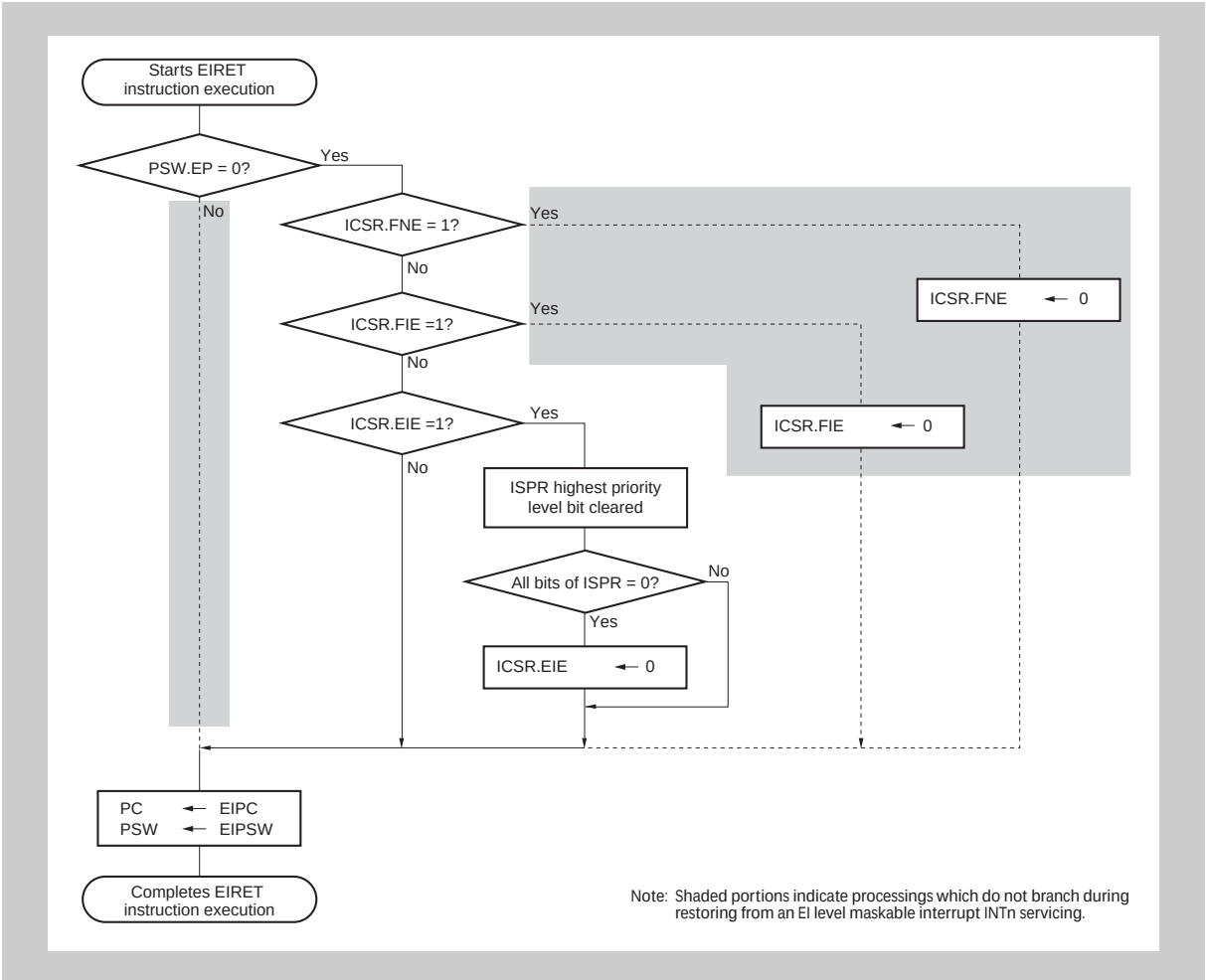



**Figure 4-5 Restore from EI level maskable interrupt INTn**

## 4.7 Interrupt Operation

### 4.7.1 Mask function of EI level maskable interrupt INTn

Interrupt masking can be specified for each respective interrupt channel of INTn. Interrupt masking is performed by the following register settings.

| ICn.MKn | Operation |
|---|---|
| 1 | Masks interrupt |
| 0 | Enables interrupt |

The ICn.MKn bits can also be read and written via the corresponding IMRmEIMKn bits of the EI level interrupt mask registers IMRm.

**Operation example**

3. When "1" is written to IMRm.IMRmEIMKn bits, interrupts are prohibited for the corresponding channels.
4. When the ICn.MKn bits are read, "1" is read.

**Caution** For MKn bit, the processing after interrupt hold is masked. Even if the MKn bit is set to 1, interrupt request acknowledgment and hold are performed. Therefore, even if software interrupts are requested for interrupts for which interrupt prohibit has been specified with MKn, no interrupt occurs. Moreover, when MKn bit is again set to 0 while an interrupt request is held, that interrupts occur at that timing. To delete an interrupt request that is already being held, clear the corresponding RFn bit.

### 4.7.2 Interrupt priority level judgment

When FENMI, FEINT, and INTn interrupts are input, priorities including other exceptions are determined, and the exception with the highest priority (including interrupts) is requested. Exceptions requested at the same time (including interrupts) are processed in a pre-allocated priority order (the default priority order). The priority orders of FENMI, FEINT, and INTn interrupts are as follows.

FENMI > FEINT > INTn

See *"V850E2/Fx4-G Interrupt Requests"* and the *"V850E2 32-bit Microcontroller Core Architecture User Manual"* for other exceptions.

For INTn interrupts, the priority level can be set independently for each interrupt source. The priority level is specified with ICn.P[2:0]n. Priority levels from 0 to 7 can be set. 0 is the highest priority level, and 7 the lowest.
In the case of multiple INTn interrupts with the same priority level, the interrupt with the lowest interrupt channel number, i.e. with the highest default priority, has priority.

**Table 4-26 Example of INTn interrupt priority level settings and priority levels**

| INTn[a] | ICn.P[2:0] setting[b] | Priority level during operation[c] |
|---|---|---|
| INT0 | 3 | 10 |
| INT1 | 4 | 11 |
| INT2 | 0 | 1 |
| INT3 | 0 | 2 |
| INT4 | 1 | 3 |
| INT5 | 2 | 6 |
| INT6 | 2 | 7 |
| INT7 | 1 | 4 |
| INT8 | 1 | 5 |
| INT9 | 2 | 8 |
| INT10 | 2 | 9 |

a) n = 0: highest default priority
b) 0: highest priority
c) 1: highest priority

During interrupt servicing, the Interrupt Controller also processes multiple interrupts acknowledging other interrupts. When multiple INTn interrupts are requested at the same time, the interrupt to be acknowledged is determined by the following procedure.

**(1) Comparison with the priority level as the interrupt currently being serviced**

Interrupts with the same or lower priority level as the interrupt currently being serviced are held pending.

The priority level of the interrupt currently being serviced is shown in the ISPR register.

Interrupts with a higher priority level than the interrupt currently being serviced proceed to the next priority judgment stage.

**(2) Masking through priority mask register (PMR)**

Only interrupts enabled by the PMR register proceed to the next priority judgment stage.

**(3) The requested interrupt source with the highest priority level is selected**

When interrupts are being simultaneously requested from multiple sources, the interrupt source from the highest priority level, determined by ICn.P[2:0]n, with the smallest interrupt channel number, i.e. highest default priority, is selected.

**(4) Interrupt hold by CPU**

Interrupt acknowledgment is pending according to the state of the NP and ID bits of the PSW register.

At this time, priority judgment

- among INTn interrupts, according to their assigned ICn.P[2:0] and their default priority (interrupt channel number n)
- and priority judgment among INTn, FEINT and FENMI interrupts

is performed even while interrupt acknowledgment is pending, and the interrupt with the highest priority is selected upon realization of the acknowledgment condition.

**Example** When a priority level ICn.P[2:0]n = 5 INTn interrupt has already been requested and interrupt generation is pending because the value of the PSW.ID bit is "1", a subsequent priority level ICn.P[2:0]n = 3 INTn interrupt is requested. Then, if the PSW.ID bit is cleared, the priority level 3 INTn interrupt is generated.

Multiple interrupt servicing in which an interrupt is acknowledged while another interrupt is being serviced is shown below.

When an interrupt request is acknowledged, the PSW.ID flag is automatically set to 1. Therefore, when multiple interrupts are to be used, clear the ID flag to 0 beforehand (for example, by placing the EI instruction in the interrupt service program) to set the interrupt enable mode.

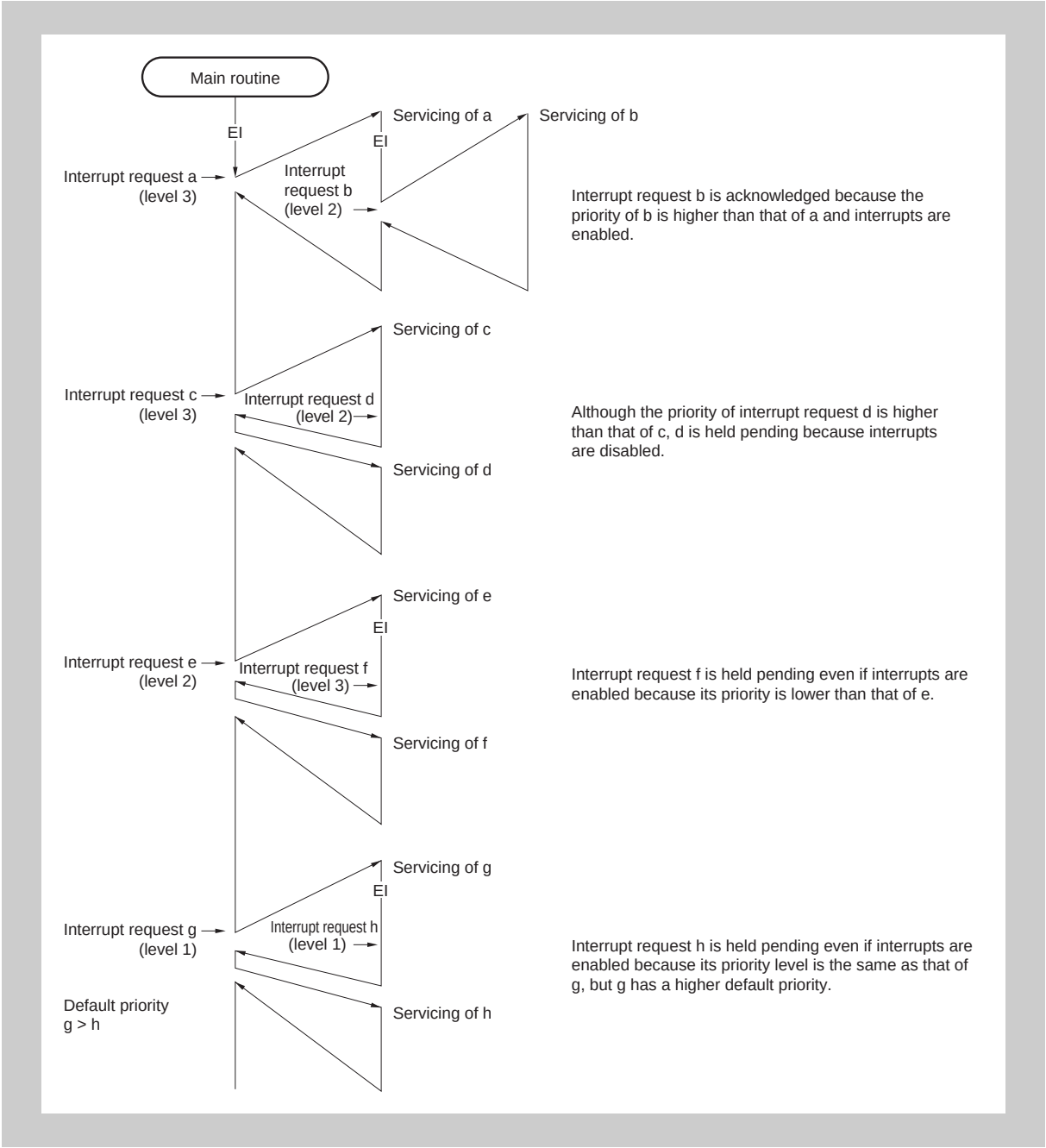



**Figure 4-6 Example of processing in which an interrupt request is issued while another interrupt is being serviced (1/2)**

**Notes**

1. "a" to "h": in the figure are the temporary names of interrupt request signals shown for the sake of explanation.
2. The default priority in the figure indicates the relative priority between two interrupt request signals.

**Caution** To perform multiple interrupt servicing, the values of the EIPC and EIPSW registers must be saved before executing the EI instruction. When returning from multiple interrupt servicing, restore the values of EIPC and EIPSW after executing the DI instruction.

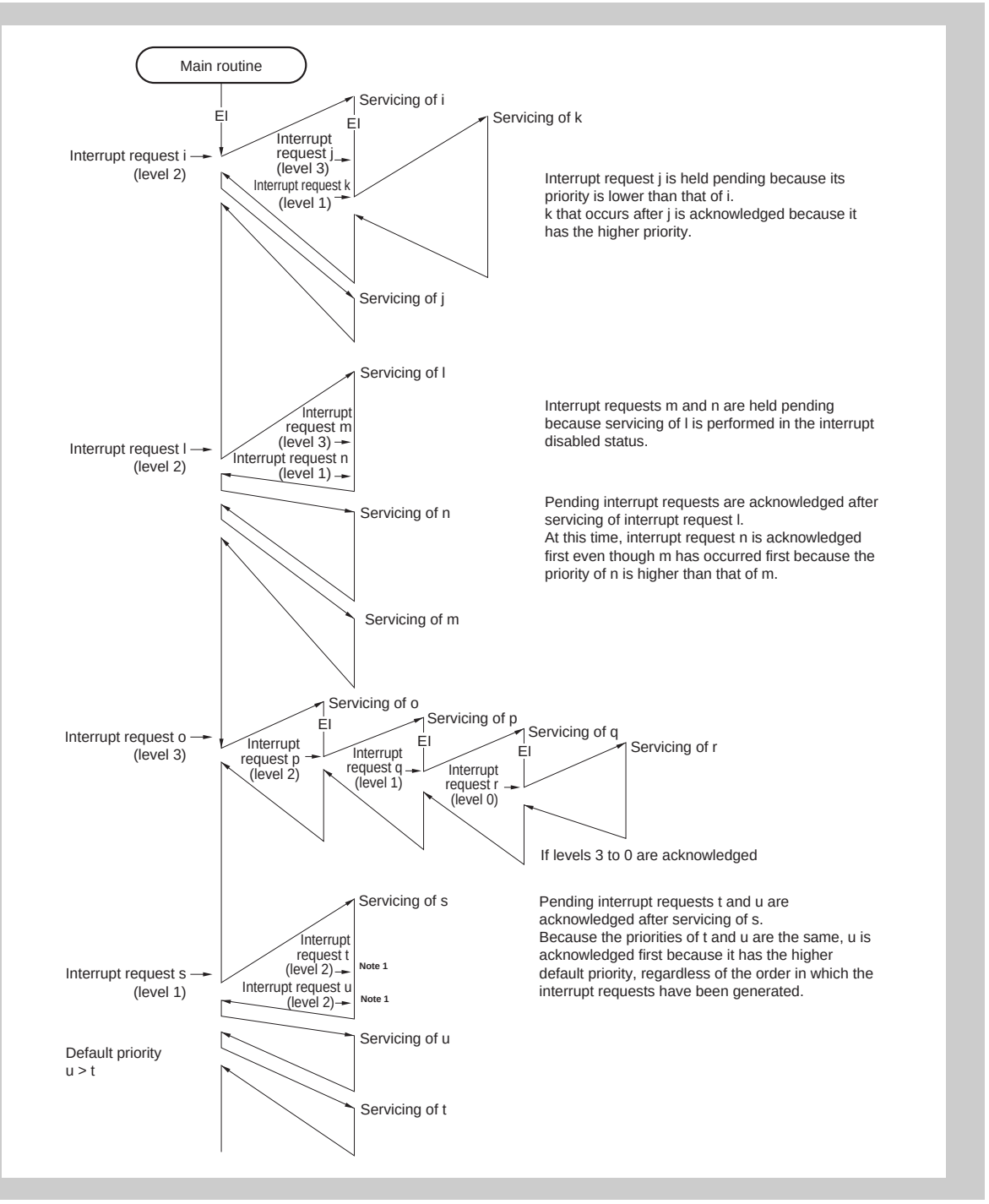



**Figure 4-7 Example of processing in which an interrupt request signal is issued while another interrupt is being serviced (2/2)**

**Notes 1.** Interrupt request “t” has lower default priority than interrupt request “u”.

**2.** “i” to “u” in the figure are the temporary names of interrupt request signals shown for the sake of explanation.

**Caution** To perform multiple interrupt servicing, the values of the EIPC and EIPSW registers must be saved before executing the EI instruction. When returning from multiple interrupt servicing, restore the values of EIPC and EIPSW after executing the DI instruction.

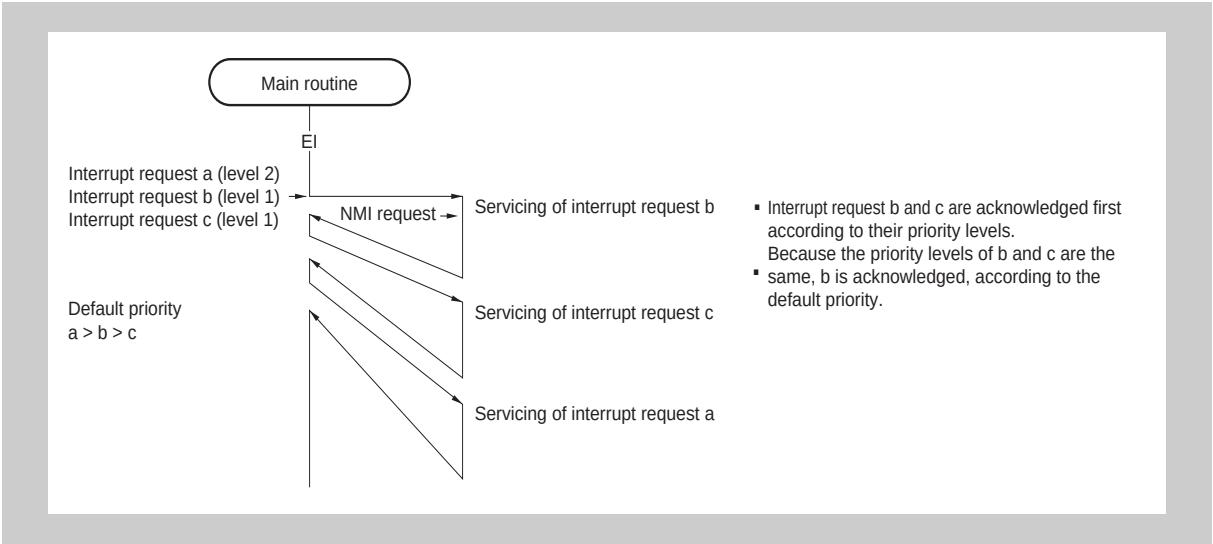



**Figure 4-8 Example of servicing simultaneously generated interrupt requests**

**Notes**

1. “a” to “c” in the figure are the temporary names of interrupt request signals shown for the sake of explanation.
2. The default priority in the figure indicates the relative priority between two interrupt request signals.

**Caution** To perform multiple interrupt servicing, the values of the EIPC and EIPSW registers must be saved before executing the EI instruction. When returning from multiple interrupt servicing, restore the values of EIPC and EIPSW after executing the DI instruction.

### 4.7.3 Priority mask function

The priority mask function prohibits in batch INTn interrupts of the specified interrupt priority level.

The interrupt masking priority level is specified with the PMR register. Masking and acknowledgment can be set for each priority level.

The following operations are possible using this function:

- Temporary prohibition of interrupts that have a priority level that is lower than a given priority level
- Temporary prohibition of interrupts that have a given priority level

| PMR.PMRm | Operation |
|---|---|
| 0 | Acknowledges requests from priority level m interrupt source. |
| 1 | Masks requests from priority level m interrupt source. |

**Note** m = 0 to 7

The PMR register prohibits interrupt occurrence. Interrupt request is acknowledged and held even while the interrupt occurrence is prohibited.

The presence of INTn interrupts held pending with this function can be checked with the next section.

### 4.7.4 Pending interrupt report function

The state of the currently pending interrupt can be checked with the pending interrupt report function.

This function allows checking of the following states:

- When interrupts that are masked only by the priority mask function (PMR) exist
  The ICSR.PMF bit is set to 1.
  The ICSR.PMF bit is not set to 1 only when interrupts that are priority masked through ISPR register or interrupts masked through MKn bit exist. Thus, the existence of priority requests pending through the priority mask function can be checked while interrupts are prohibited through priority masking.
- When EI level maskable interrupt request is not output to the CPU
  The ICSR.EIR bit is set to 1.
  By looking at the ICSR.EIR bit in the interval during which PSW.ID = 1, it is possible to check whether an INTn interrupt request exists.
- When FE level maskable interrupt request is not output to the CPU
  ICSR.FIR bit is set to 1.
  By looking at the ICSR.FIR bit in the interval during which PSW.NP = 1, it is possible to check whether a FEINT interrupt request exists.

### 4.7.5 In-service priority clear function

This function initializes the internal status of the Interrupt Controller. It operates when the ISPC register is accessed. The following operations are possible using this function.

- Clear all contents of ISPR register
- Clear ICSR.EIE, FIE, and FNE bits

All the bits of ISPR register can be cleared to 0 by writing "1" to all the bits of this register and then writing "0" to all the bits of ISPR register. Moreover, the ICSR.EIE, FIE, and FNE bits, which all indicates the state in which an interrupt request is being processed in the CPU, are all be cleared.

The value of this register is automatically cleared to 0 by writing 0 to all the bits of ISPR. The values of the bits of ISPR remain unchanged in the case of write access that is not performed simultaneously to all the bits.

## 4.8 Exception Handler Address Switching Function

Interrupt handler addresses can be switched by software.

For details, refer to the chapter *“Exception Handler Address Switching Functio*n” in the “V850E2 32-bit Microcontroller Core Architecture” user manual.

# Chapter 5 DMA Controller (DMAC)

This chapter contains a generic description of the DMA Controller (DMAC).

The first section describes all V850E2/Fx4-G specific properties.

## 5.1 V850E2/Fx4-G DMA Features

**DMA channels**

This microcontroller has following number of DMA channels.

**Table 5-1 Instances of FCN**

| DMA channels | Fx4-G |
|---|---|
| DMA channels | n = 0 to 7 |

**Channel index n**

Throughout this chapter, the index n is used to indentify an individual DMA channel.

**DMA address map**

Refer to the section *"V850E2/Fx4-G CPU Address Map"* in chapter *"CPU System Functions"* for details.

**Clock supply**

Since the DMA Controller is part of the CPU Subsystem, it is supplied with the same clock CKSCLC_000 as the CPU Subsystem.
Refer to the section *"CPU Subsystem"* in chapter *"CPU System Functions"* for details.

**DMAC H/W reset**

The DMA Controller and its registers are initialized by the following reset signal:

**Table 5-2 DMAC reset signal**

| DMAC | Reset signal |
|---|---|
| DMAC | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

### (1) DMA trigger factors

The assignment of a DMA trigger source to the DMA channel n is done by the DTFNRn.IFC[5:0].

The following table lists all DMA triggers and how to select them by setting the DMA trigger factor register DTFRn.

**Table 5-3 DMA trigger factors selection**

| DTFRn.IFCn[6:0] | DMA trigger interrupt |
|---|---|
| 0 | No DMA |
| 1 | INTP0 |
| 2 | INTP1 |
| 3 | INTP2 |
| 4 | INTP3 |
| 5 | INTP4 |
| 6 | INTP5 |
| 7 | INTP6 |
| 8 | INTP7 |
| 9 | INTP8 |
| 10 | INTP9 |
| 11 | INTP10 |
| 12 | INTTAUB0I8 |
| 13 | INTTAUB0I9 |
| 14 | INTTAUB0I10 |
| 15 | INTTAUB0I11 |
| 16 | INTTAUB0I12 |
| 17 | INTTAUB0I13 |
| 18 | INTTAUB0I14 |
| 19 | INTTAUB0I15 |
| 20 | INTFCN0REC |
| 21 | INTFCN1REC |
| 22 | INTFCN2TRX |
| 23 | INTFCN2REC |
| 24 | INTFCN3TRX |
| 25 | INTFCN3REC |
| 26 | INTFCN4TRX |
| 27 | INTFCN4REC |
| 28 | INTFCN0TRX |
| 29 | INTFCN1TRX |
| 30 | INTADAA0I0 |
| 31 | INTADAA0I1 |

| DTFRn.IFCn[6:0] | DMA trigger interrupt |
|---|---|
| 32 | INTADAA0I2 |
| 33 | INTADAA0LLT |
| 34 | INTCSIG0IR |
| 35 | INTCSIG0IC |
| 36 | INTDMA0 |
| 37 | INTDMA1 |
| 38 | INTDMA2 |
| 39 | INTDMA3 |
| 40 | INTDMA4 |
| 41 | INTDMA5 |
| 42 | reserved |
| 43 | INTIICB0IA |
| 44 | INTTAUJ0I0 |
| 45 | INTTAUJ0I1 |
| 46 | INTTAUJ0I2 |
| 47 | INTTAUJ0I3 |
| 48 | INTCSIG4IR |
| 49 | INTCSIG4IC |
| 50 | INTFCN5TRX |
| 51 | INTFCN5REC |
| 52 | INTLMA2IT |
| 53 | INTFCN0ERR |
| 54 | INTFCN1ERR |
| 55 | INTFCN2ERR |
| 56 | INTCSIG7IR |
| 57 | INTCSIG7IC |
| 58 | INTLMA10IR |
| 59 | INTLMA10IT |
| 60 | INTLMA11IR |
| 61 | INTLMA11IT |
| 62 | reserved |
| 63 | reserved |

## 5.2 Definition of Terms

The terms used in this document are defined as follows.

**Table 5-4 Definition of terms**

| Term | Function |
|---|---|
| DMA transfer | Period from the start of the first DMA cycle to assertion of INTDMAn |
| DMA cycle | Period of transferring one unit of data (since a read cycle has been started and until a write cycle is completed). |
| Hardware DMA transfer request | 64 hardware DMA transfer request (INTIN[63:0]) |
| Software DMA transfer request | DMA transfer request by internal register (DTSnSR bit) |
| DMA transfer request | Hardware DMA transfer request or software DMA transfer request |
| Single transfer | One DMA cycle is executed per transfer request. |
| Block transfer | The number of transfers set in the transfer count setting register (DTCn) is executed per software DMA transfer request.<br>Since the bus is released after each transfer, the CPU is able to take over the bus and interrupt the DMA transfers.<br>If a higher-priority transfer request occurs during execution of a single step transfer, the single step transfer is aborted while the higher-priority transfer request is executed. |

## 5.3 General

Direct memory access (DMA) function is used to access data without going via the CPU.

The DMA Controller incorporates following units:

- DMA trigger factor register (DTFR)
- DMA controller (DMAC)
- DMA data transfer unit (DMAT)

**Note** The number of DMA channels, expressed by the index n, is defined in the first section of this chapter under the key words *“DMA channels”* and *“Channel index n”*.

### 5.3.1 DMA Controller (DMAC) function

- Registers to store transfer information (transfer address, transfer size, etc.) and registers to control DMAC are included.
- When a DMA transfer request is accepted, a transfer request is generated, according to the contained transfer information.
- Hardware DMA transfer requests, DMA acknowledge signals, and DMA transfer completion interrupts are input and output.
- Write back information is written back to registers.

### 5.3.2 DMA trigger factor register (DTFR) function

The DMA transfer factors are selected from among interrupt signals.

**Table 5-5 Transfer target spaces**

| Transfer destination / Transfer source | Peripheral I/O | External memory | Data RAM | Internal Code flash | Internal Data-Flash |
|---|---|---|---|---|---|
| Peripheral I/O | √ | √ | √ | x | x |
| External memory | √ | √ | √ | x | x |
| Data RAM | √ | √ | √ | x | x |
| Internal Code flash | √ | √ | √ | x | x |
| Internal Data-Flash | √ | √ | √ | x | x |

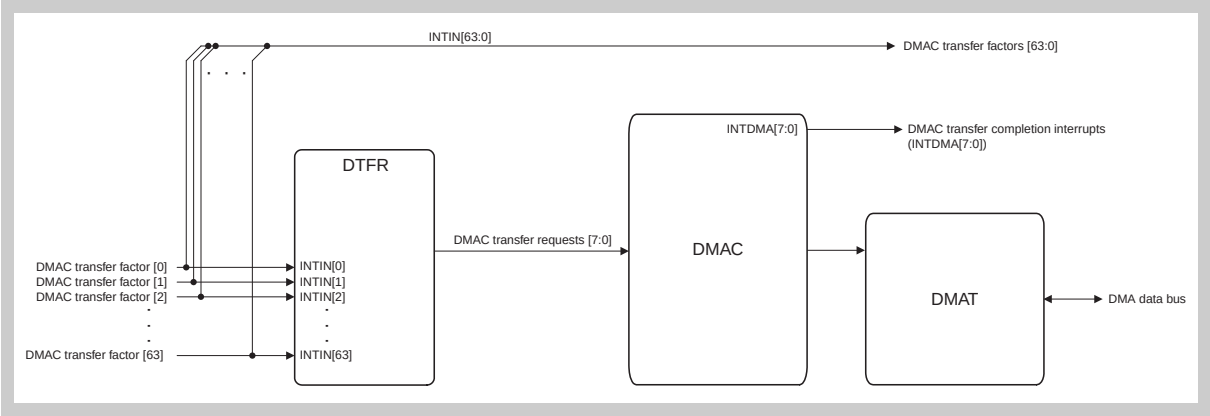



**Figure 5-1 DMA trigger and interrupt signals connections**

### 5.3.3 DMA access memory map

Refer to the section *"V850E2/Fx4-G CPU Address Map"* in chapter *"CPU System Functions"* for details.

### 5.3.4 Prioritization of channels

The following describes how the prioritization of the DMA subsystem's various transfer channels is determined.

The priority is set as

CH0 > CH1 > CH2 > CH3 > CH4 > CH5 > CH6 > CH7,

in which CH0 has the highest priority.

### 5.3.5 Stand-by function

A stop acknowledge signal is returned following a stop request when transfer of the DMA transfer unit is completed.

The transfer unit for DMA is one DMA cycle (from when read starts until write ends).

## 5.4 DMAC Function

### 5.4.1 Features

**Transfer data size**

- 8 bits
- 16 bits
- 32 bits

**Transfer data**

Fixed to little endian

Misaligned data not supported

**Maximum transfer count**

32,768 ($2^{15}$) times

**Channel priority control**

Fixed priority (highest priority (CH0) → lowest priority (CH7))

**Subject to transfer**

- Code flash (as source only)
- Data RAM
- Data-Flash (as source only)
- Peripheral I/O area

**Transfer type**

2-cycle transfer (dual address transfer)

The address at both the transfer source and destination is accessed. Two bus cycles (read cycle + write cycle) are required to execute a single transfer. Because the bus is not locked between the read cycle and write cycle, a CPU bus cycle may take place.

**Transfer modes**

- Single transfer mode (when hardware DMA transfer request is generated)

  When a hardware DMA transfer request is generated, the bus mastership is acquired, and the bus is always released after transfer has been executed once. If another hardware DMA transfer request is generated after that, transfer is executed once again. This operation is repeated until transfer has been executed the number of times specified by the transfer count register (DTC).

- Block transfer mode (when software DMA transfer request is generated)

  When a software DMA transfer request is generated, the bus mastership is acquired, and the bus is released each time transfer has been executed once. Once a software DMA transfer request has been acknowledged, this operation is repeated until transfer has been executed the number of times specified by the transfer count register (DTC).

**Transfer address control**

- Incremental
- Fixed

**Transfer error support**

When the data from the transfer source contains an error, or if an error occurs at the transfer destination, DMA transfer is aborted, and a SYSERR exception is output for the CPU.

**DMA transfer request**

A hardware DMA transfer request or a software DMA transfer request can be selected for each channel.
The software DMA transfer request can be set by software (by setting the DTS register). This register also has a status bit (DTS register) that indicates that a hardware DMA transfer request has been generated.

**Transfer completion interrupt output function**

This function outputs a transfer completion interrupt signal (INTDMA7 to INTDMA0) when DMA transfer of each channel has been completed the number of times specified by the transfer count register (DTC).

**Stand-by support**

When a stop mode request is generated, DMA transfer is aborted momentarily and the stop mode is entered.

**DMA transfer abort function**

This function supports aborting DMA transfer by software.

### 5.4.2 DMAC setting registers

Table 5-6 DMAC setting registers (1/2)

| Address | Symbol | Function register name | R/W | Operable bits | | | | Initial value |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 8 | 16 | 32 | |
| FFFF7300H | DTRC | DMA transfer request control register | R/W | √ | √ | | | 00H |
| FFFF7314H | DSA0 | DMA source address register CH0 | | | | | √ | 00000000H |
| FFFF7314H | DSA0L | DMA source address register LCH0 | | | | √ | | 0000H |
| FFFF7316H | DSA0H | DMA source address register HCH0 | | | | √ | | 0000H |
| FFFF7324H | DDA0 | DMA destination address register CH0 | | | | | √ | 00000000H |
| FFFF7324H | DDA0L | DMA destination address register LCH0 | | | | √ | | 0000H |
| FFFF7326H | DDA0H | DMA destination address register HCH0 | | | | √ | | 0000H |
| FFFF7332H | DTC0 | DMA transfer count register CH0 | | | | √ | | 0000H |
| FFFF7338H | DTCT0 | DMA transfer control register CH0 | | | | √ | | 0000H |
| FFFF733AH | DTS0 | DMA transfer status register CH0 | | √ | √ | | | 00H |
| FFFF7344H | DSA1 | DMA source address register CH1 | | | | | √ | 00000000H |
| FFFF7344H | DSA1L | DMA source address register LCH1 | | | | √ | | 0000H |
| FFFF7346H | DSA1H | DMA source address register HCH1 | | | | √ | | 0000H |
| FFFF7354H | DDA1 | DMA destination address register CH1 | | | | | √ | 00000000H |
| FFFF7354H | DDA1L | DMA destination address register LCH1 | | | | √ | | 0000H |
| FFFF7356H | DDA1H | DMA destination address register HCH1 | | | | √ | | 0000H |
| FFFF7362H | DTC1 | DMA transfer count register CH1 | | | | √ | | 0000H |
| FFFF7368H | DTCT1 | DMA transfer control register CH1 | | | | √ | | 0000H |
| FFFF736AH | DTS1 | DMA transfer status register CH1 | | √ | √ | | | 00H |
| FFFF7374H | DSA2 | DMA source address register CH2 | | | | | √ | 00000000H |
| FFFF7374H | DSA2L | DMA source address register LCH2 | | | | √ | | 0000H |
| FFFF7376H | DSA2H | DMA source address register HCH2 | | | | √ | | 0000H |
| FFFF7384H | DDA2 | DMA destination address register CH2 | | | | | √ | 00000000H |
| FFFF7384H | DDA2L | DMA destination address register LCH2 | | | | √ | | 0000H |
| FFFF7386H | DDA2H | DMA destination address register HCH2 | | | | √ | | 0000H |
| FFFF7392H | DTC2 | DMA transfer count register CH2 | | | | √ | | 0000H |
| FFFF7398H | DTCT2 | DMA transfer control register CH2 | | | | √ | | 0000H |
| FFFF739AH | DTS2 | DMA transfer status register CH2 | | √ | √ | | | 00H |
| FFFF73A4H | DSA3 | DMA source address register CH3 | | | | | √ | 00000000H |
| FFFF73A4H | DSA3L | DMA source address register LCH3 | | | | √ | | 0000H |
| FFFF73A6H | DSA3H | DMA source address register HCH3 | | | | √ | | 0000H |
| FFFF73B4H | DDA3 | DMA destination address register CH3 | | | | | √ | 00000000H |
| FFFF73B4H | DDA3L | DMA destination address register LCH3 | | | | √ | | 0000H |
| FFFF73B6H | DDA3H | DMA destination address register HCH3 | | | | √ | | 0000H |
| FFFF73C2H | DTC3 | DMA transfer count register CH3 | | | | √ | | 0000H |
| FFFF73C8H | DTCT3 | DMA transfer control register CH3 | | | | √ | | 0000H |
| FFFF73CAH | DTS3 | DMA transfer status register CH3 | | √ | √ | | | 00H |
| FFFF73D4H | DSA4 | DMA source address register CH4 | | | | | √ | 00000000H |

**Table 5-6 DMAC setting registers (2/2)**

| Address | Symbol | Function register name | R/W | Operable bits | | | | Initial value |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 8 | 16 | 32 | |
| FFFF73D4H | DSA4L | DMA source address register LCH4 | | | | √ | | 0000H |
| FFFF73D6H | DSA4H | DMA source address register HCH4 | | | | √ | | 0000H |
| FFFF73E4H | DDA4 | DMA destination address register CH4 | | | | | √ | 00000000H |
| FFFF73E4H | DDA4L | DMA destination address register LCH4 | | | | √ | | 0000H |
| FFFF73E6H | DDA4H | DMA destination address register HCH4 | | | | √ | | 0000H |
| FFFF73F2H | DTC4 | DMA transfer count register CH4 | | | | √ | | 0000H |
| FFFF73F8H | DTCT4 | DMA transfer control register CH4 | | | | √ | | 0000H |
| FFFF73FAH | DTS4 | DMA transfer status register CH4 | | √ | √ | | | 00H |
| FFFF7404H | DSA5 | DMA source address register CH5 | | | | | √ | 00000000H |
| FFFF7404H | DSA5L | DMA source address register LCH5 | | | | √ | | 0000H |
| FFFF7406H | DSA5H | DMA source address register HCH5 | | | | √ | | 0000H |
| FFFF7414H | DDA5 | DMA destination address register CH5 | | | | | √ | 00000000H |
| FFFF7414H | DDA5L | DMA destination address register LCH5 | | | | √ | | 0000H |
| FFFF7416H | DDA5H | DMA destination address register HCH5 | | | | √ | | 0000H |
| FFFF7422H | DTC5 | DMA transfer count register CH5 | | | | √ | | 0000H |
| FFFF7428H | DTCT5 | DMA transfer control register CH5 | | | | √ | | 0000H |
| FFFF742AH | DTS5 | DMA transfer status register CH5 | | √ | √ | | | 00H |
| FFFF7434H | DSA6 | DMA source address register CH6 | | | | | √ | 00000000H |
| FFFF7434H | DSA6L | DMA source address register LCH6 | | | | √ | | 0000H |
| FFFF7436H | DSA6H | DMA source address register HCH6 | | | | √ | | 0000H |
| FFFF7444H | DDA6 | DMA destination address register CH6 | | | | | √ | 00000000H |
| FFFF7444H | DDA6L | DMA destination address register LCH6 | | | | √ | | 0000H |
| FFFF7446H | DDA6H | DMA destination address register HCH6 | | | | √ | | 0000H |
| FFFF7452H | DTC6 | DMA transfer count register CH6 | | | | √ | | 0000H |
| FFFF7458H | DTCT6 | DMA transfer control register CH6 | | | | √ | | 0000H |
| FFFF745AH | DTS6 | DMA transfer status register CH6 | | √ | √ | | | 00H |
| FFFF7464H | DSA7 | DMA source address register CH7 | | | | | √ | 00000000H |
| FFFF7464H | DSA7L | DMA source address register LCH7 | | | | √ | | 0000H |
| FFFF7466H | DSA7H | DMA source address register HCH7 | | | | √ | | 0000H |
| FFFF7474H | DDA7 | DMA destination address register CH7 | | | | | √ | 00000000H |
| FFFF7474H | DDA7L | DMA destination address register LCH7 | | | | √ | | 0000H |
| FFFF7476H | DDA7H | DMA destination address register HCH7 | | | | √ | | 0000H |
| FFFF7482H | DTC7 | DMA transfer count register CH7 | | | | √ | | 0000H |
| FFFF7488H | DTCT7 | DMA transfer control register CH7 | | | | √ | | 0000H |
| FFFF748AH | DTS7 | DMA transfer status register CH7 | | √ | √ | | | 00H |

**Caution** If an unmapped address is accessed, a write access is ignored and “0” is returned in response to a read access.

### 5.4.3 Enabling or disabling writing control registers

The following control registers cannot be written while DMA transfer is enabled. All these registers can always be read, however.

**Table 5-7 Enabling/disabling writing control registers**

| | |
|---|---|
| Always writable | DTRC, DTSn |
| Writing prohibited while DMA transfer is enabled (DTE = 1) (Operation is not guaranteed if these registers are written.) | DSA, DSAnL, DSAnH, DDA, DDAnL, DDAnH, DTCn, DTCTn |

## 5.5 DMA Control Registers

### 5.5.1 DTRC – DMA transfer request control register

**Access** This register can be read or written in 8-bit units.

**Address** FFFF 7300$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ERR | 0 | 0 | 0 | 0 | 0 | 0 | ADS |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-8 DTRC0 registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | ERR | DMA transfer error status<br>This bit indicates that an error response has been received from the transfer target during DMA transfer.<br>If an error response is received, the ERR and ADS bits are set and a system error exception SYSERR is generated. To clear this bit, write “0” to it.<br>0: No DMA transfer error<br>1: DMA transfer error |
| 0 | ADS | DMA transfer aborted<br>This bit indicates that DMA transfer has been aborted by a transfer stop request. In addition, the current DMA transfer can be aborted if the user writes “1” to this bit.<br>0: DMA transfer not aborted<br>1: DMA transfer aborted/DMA transfer abort request |

### 5.5.2 DSAnL – DMA source address register L

These registers hold the lower 16 bits of the transfer source address. The higher 16 bits of the transfer source address are accessible via the DSAnH register.

**32-bit access** The entire 26-bit address is also accessible through the 32-bit register DSAn. DSAn has the same address as the DSAnL register.

If the entire address shall be read, while the DMA transfer of channel is enables (DTSn.DTSnDTE = 1), proceed as follows in order to acquire the correct address:

- Read DSAn twice consecutively and compare the upper 10 address bits SA[25:16] of both read accesses.
- In case the upper 10 address bits SA[25:16] are identical between both read accesses the address SA[25:0] of the second read access represents the correct address.
- In case the upper 10 address bits SA[25:16] are different between both read accesses the address SA[25:0] of the first read access represents the correct address.

**Access** This register can be read or written in 16-bit units.

**Address** DSA7L: FFFF 7464$_H$, DSA6L: FFFF 7434$_H$, DSA5L: FFFF 7404$_H$, DSA4L: FFFF 73D4$_H$, DSA3L: FFFF 73A4$_H$, DSA2L: FFFF 7374$_H$, DSA1L: FFFF 7344$_H$, DSA0L: FFFF 7314$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| SA[15:8] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SA[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-9 DSAnL registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | SA[15:0] | DMA source address lower 16 bits<br>If this register is referenced during DMA transfer, the address from which data is to be transferred next can be read.<br>When referencing this register, it is recommended to access this register together with DSAnH via the 32-bit register DSAn.<br>When the entire DMA transfer has been completed, the values of these bits return to their initial values, i.e. when DMA transfer was started. |

**Cautions**

1. Writing this register is prohibited while DMA transfer of channel n is enabled (DTSn.DTSnDTE = 1). If it is written, correct DMA operation is not guaranteed.
2. Set an address by accessing in 32-bit units while DMA transfer of channel n is disabled (DTSn.DTSnDTE = 0) in order to avoid data being transferred from an address that has not been completely set.
3. DMA transfer of misaligned data is not supported. The lower 2 bits of an address corresponding to the transfer data size are as follows (x indicates any bit).

   The operation is not guaranteed if a setting other than the following is made.

| Data size | SA1 | SA0 |
|---|---|---|
| 8 bits | x | x |
| 16 bits | x | 0 |
| 32 bits | 0 | 0 |

### 5.5.3 DSAnH – DMA source address register H

These registers hold the higher 10 bits of the transfer source address.
The lower 16 bits of the transfer source address are accessible via the DSAnL register.

**Access** This register can be read or written in 16-bit units.

**Address** DSA7H: FFFF $7466_H$, DSA6H: FFFF $7436_H$, DSA5H: FFFF $7406_H$, DSA4H: FFFF $73D6_H$, DSA3H: FFFF $73A6_H$, DSA2H: FFFF $7376_H$, DSA1H: FFFF $7346_H$, DSA0H: FFFF $7316_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | SA[25:24] | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SA[23:16] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-10 DSAnH registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 9 to 0 | SA[25:16] | DMA source address<br>If this register is referenced during DMA transfer, the address from which data is to be transferred next can be read.<br>When referencing this register, it is recommended to access this register together with DSAnL in 32-bit units.<br>When the entire DMA transfer has been completed, the values of these bits return to their initial values, i.e. when DMA transfer was started. |

**Cautions**

1. Writing this register is prohibited while DMA transfer of channel n is enabled (DTSn.DTSnDTE = 1). If it is written, correct DMA operation is not guaranteed.
2. Set an address by accessing in 32-bit units while DMA transfer of channel n is disabled (DTSn.DTSnDTE = 0) in order to avoid data being transferred from an address that has not been completely set.

### 5.5.4 DDAnL – DMA destination address register L

These registers hold the lower 16 bits of the transfer destination address. The higher 16 bits of the transfer destination address are accessible via the DDAnH register.

**32-bit access**

The entire 29-bit address is also accessible through the 32-bit register DDAn. DDAn has the same address as the DDAnL register.

If the entire address shall be read, while the DMA transfer of channel is enables (DTSn.DTSnDTE = 1), proceed as follows in order to acquire the correct address:

- Read DDAn twice consecutively and compare the upper 10 address bits DA[25:16] of both read accesses.
- In case the upper 10 address bits DA[25:16] are identical between both read accesses the address DA[25:0] of the second read access represents the correct address.
- In case the upper 10 address bits DA[25:16] are different between both read accesses the address DA[25:0] of the first read access represents the correct address.

**Access** This register can be read or written in 16-bit units.

**Address** DDA7L: FFFF 7474$_H$, DDA6L: FFFF 7444$_H$, DDA5L: FFFF 7414$_H$, DDA4L: FFFF 73E4$_H$, DDA3L: FFFF 73B4$_H$, DDA2L: FFFF 7384$_H$, DDA1L: FFFF 7354$_H$, DDA0L: FFFF 7324$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| DA[15:8] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DA[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-11 DDAnL registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | DA[15:0]DA0 | DMA destination address<br>If this register is referenced during DMA transfer, the address to which data is to be transferred next can be read.<br>When referencing this register, it is recommended to access this register together with DDAnH in 32-bit units.<br>When the entire DMA transfer has been completed, the values of these bits return to their initial values, i.e. when DMA transfer was started. |

**Cautions**

1. Writing these bits is prohibited while DMA transfer of channel n is enabled (DTSn.DTSnDTE = 1). If they are written, the operation is not guaranteed.
2. Set an address by accessing in 32-bit units while DMA transfer of channel n is disabled (DTSn.DTSnDTE = 0) in order to avoid data being transferred from an address that has not been completely set.
3. If an error occurs in the transfer target in the read cycle of DMA transfer, the write cycle is not executed but the destination address is updated.
4. DMA transfer of misaligned data is not supported. The lower 2 bits of an address corresponding to the transfer data size are as follows (x indicates any bit).

   The operation is not guaranteed if a setting other than the following is made.

| Data size | DA1 | DA0 |
|---|---|---|
| 8 bits | x | x |
| 16 bits | x | 0 |
| 32 bits | 0 | 0 |

### 5.5.5 DDAnH – DMA destination address register H

These registers hold the higher 10 bits of the transfer destination address. The lower 16 bits of the transfer destination address are accessible via the DDAnL register.

**Note** The entire 29-bit transfer destination address is accessible via the 32-bit register DDAn. Refer to the description of the DDAnL for details about the 32-bit access.

**Access** This register can be read or written in 16-bit units.

**Address** DDA7H: FFFF $7476_H$, DDA6H: FFFF $7446_H$, DDA5H: FFFF $7416_H$, DDA4H: FFFF $73E6_H$, DDA3H: FFFF $73B6_H$, DDA2H: FFFF $7386_H$, DDA1H: FFFF $7356_H$, DDA0H: FFFF $7326_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | DA[25:24] | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DA[23:16] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-12 DDAnH registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 9 to 0 | DA[25:16] | DMA destination address<br>If this register is referenced during DMA transfer, the address to which data is to be transferred next can be read.<br>When referencing this register, it is recommended to access this register together with DDAnL in 32-bit units.<br>When the entire DMA transfer has been completed, the values of these bits return to their initial values, i.e. when DMA transfer was started. |

**Cautions**

1. Writing these bits is prohibited while DMA transfer of channel n is enabled (DTSn.DTSnDTE = 1). If they are written, the operation is not guaranteed.
2. Set an address by accessing in 32-bit units while DMA transfer of channel n is disabled (DTSn.DTSnDTE = 0) in order to avoid data being transferred from an address that has not been completely set.
3. If an error occurs in the transfer target in the read cycle of DMA transfer, the write cycle is not executed but the destination address is updated.

### 5.5.6 DTCn – DMA transfer count register

**Access** This register can be read or written in 16-bit units.

**Address** DTC7: FFFF $7482_H$, DTC6: FFFF $7452_H$, DTC5: FFFF $7422_H$, DTC4: FFFF $73F2_H$, DTC3: FFFF $73C2_H$, DTC2: FFFF $7392_H$, DTC1: FFFF $7362_H$, DTC0: FFFF $7332_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | DTC[14:8] | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DTC[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-13 DTCn registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 14 to 0 | DTC[14:0] | DMA transfer count<br>These bits specify the number of times of DMA transfers (DMA transfer count) for channel n.<br>When this register is referenced during DMA transfer, the remaining number of times DMA transfer to be executed can be read.<br>When the entire DMA transfer has been completed, the values of these bits return to their initial values, i.e. when DMA transfer was started. |

| DTC[14:0] | The operation |
|---|---|
| $0000_H$ | Transfer executed 32,768 times or until completion of transfer |
| $0001_H$ | Transfer executed once or transfer to be executed once |
| : | : |
| $7FFF_H$ | Transfer executed 32,767 times or 32,767 times of transfer to be executed |

**Cautions**

1. Writing these bits is prohibited while DMA transfer of channel n is enabled (DTSn.DTSnDTE = 1). If they are written, the operation is not guaranteed.
2. If an error occurs in the transfer target in the read cycle of DMA transfer, the write cycle is not executed but the destination address is updated.

### 5.5.7 DTCTn – DMA transfer control register

**Access** This register can be read or written in 16-bit units.

**Address** DTCT7: FFFF 7488$_H$, DTCT6: FFFF 7458$_H$, DTCT5: FFFF 7428$_H$, DTCT4: FFFF 73F8$_H$, DTCT3: FFFF 73C8$_H$, DTCT2: FFFF 7398$_H$, DTCT1: FFFF 7368$_H$, DTCT0: FFFF 7338$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | DTCTnDS[1:0] | | DTCTnLE | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DTCTnSACM[1:0] | | DTCTnDACM[1:0] | | 0 | 0[a] | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

a) The default value “0” of bit 2 must not be changed.

**Caution** The default value “0” of bit 2 must not be changed.

**Table 5-14 DTCTn registers contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 14 to 13 | DTCTn DS[1:0] | DMA transfer data size<br>These bits specify the DMA transfer data size of channel n. |
| 12 | DTCTn LE | This bit determines whether a new DMA transfer request is accepted, though a current DMA transfer on channel n is not completed (DTSn.DTSnTC = 0).<br>If a new DMA transfer request is accepted (DTCTnLE = 1), channel n remains enabled (DTSnDTE = 1) in order to start the next DMA transfer.<br>0: transfer request is not accepted during ongoing transfer, DTSnDTE is cleared upon completion of ongoing transfer<br>1: transfer request is accepted during ongoing transfer, DTSnDTE is not cleared upon completion of ongoing transfer<br>**Note:** This bit is effective only in single transfer mode. |
| 7 to 6 | DTCTn SACM[1:0] | DMA transfer source address counting direction<br>These bits specify the direction in which the transfer source address of channel n is to be counted. |

| DTCTn DS1 | DTCTn DS0 | Transfer data size |
|---|---|---|
| 0 | 0 | 8 bits |
| 0 | 1 | 16 bits |
| 1 | 0 | 32 bits |
| 1 | 1 | Setting prohibited |

| SACM1 | SACM0 | Counting direction |
|---|---|---|
| 0 | 0 | Incremented |
| 0 | 1 | Setting prohibited |
| 1 | 0 | Fixed |
| 1 | 1 | Setting prohibited |

**Table 5-14 DTCTn registers contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 5 to 4 | DTCTn DACM[1:0] | DMA transfer destination address counting direction<br>These bits specify the direction in which the transfer destination address of channel n is to be counted. |

| DACM1 | DACM0 | Counting direction |
|---|---|---|
| 0 | 0 | Incremented |
| 0 | 1 | Setting prohibited |
| 1 | 0 | Fixed |
| 1 | 1 | Setting prohibited |

**Cautions**

1. Writing these bits is prohibited while DMA transfer of channel n is enabled (DTSn.DTSnDTE = 1). If they are written, the operation is not guaranteed.
2. The operation cannot be guaranteed if the SACM[1:0] and DACM[1:0] bits are set to a prohibited status.

### 5.5.8 DTSn – DMA transfer status register

**Access** This register can be read or written in 8- or 1-bit units.

**Address** DTS7: FFFF 748A$_H$, DTS6: FFFF 745A$_H$, DTS5: FFFF 742A$_H$, DTS4: FFFF 73FA$_H$, DTS3: FFFF 73CA$_H$, DTS2: FFFF 739A$_H$, DTS1: FFFF 736A$_H$, DTS0: FFFF 733A$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DTSnTC | DTSnDT | 0 | 0 | DTSnER | DTSnDR | DTSnSR | DTSnDTE |
| R/W | R/W | R | R | R | R | R/W | R/W |

**Table 5-15 DTSn registers contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | DTSnTC | DMA transfer end status<br>This bit indicates that DMA transfer has been completed. Write “0” to this bit to clear it after reading “1” from it. It is recommended to write this bit using bit manipulation such as CLR1.<br>0: DMA transfer not completed<br>1: DMA transfer completed |
| 6 | DTSnDT | DT DMA transfer status<br>This bit indicates that a DMA transfer request has been acknowledged and that DMA transfer is in progress. It is not set (to “1”) when only a DMA transfer request is issued. This bit is cleared (to “0”) when DMA transfer has been completed. If the DTE bit is “0”, this bit can be cleared by the user. (It can also be written at the same time as the DTE bit.)<br>0: DMA transfer request acknowledged<br>1: DMA transfer in progress |
| 3 | DTSnER | DMA transfer error flag<br>This bit indicates that a DMA transfer error has occurred in channel n. It is cleared (to “0”) when the ERR bit of the DTRC register is cleared. Note that this bit is read-only.<br>0: No DMA transfer error<br>1: DMA transfer error |
| 2 | DTSnDR | Hardware DMA transfer request flag<br>This bit indicates that channel n has a hardware DMA transfer request. It is cleared (to “0”) when the hardware DMA transfer request is deasserted. This bit operates regardless of the status of the DTE bit. It is not set (to “1”) by a software DMA transfer request, or by a hardware DMA transfer request when a software DMA transfer request is selected by the DMA transfer request select register. Note that this bit is read-only.<br>0: No hardware DMA transfer request<br>1: Hardware DMA transfer request |
| 1 | DTSnSR | Software DMA transfer request<br>This bit selects a software DMA transfer request. If a software DMA transfer request is selected by the DMA transfer request select register, writing “1” to this bit and the DTE bit starts DMA transfer. This bit is automatically cleared (to “0”) when DMA transfer has been completed. Writing “0” to this bit aborts DMA transfer.<br>0: No software DMA transfer request<br>1: Software DMA transfer request |

**Table 5-15 DTSn registers contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | DTSnDTE | DMA transfer enable<br>This bit enables or disables DMA transfer. DMA transfer is executed if “1” is written to this bit and a DMA transfer request is issued. This bit is automatically cleared (to “0”) if the MLE bit is “0” when DMA transfer has been completed. DMA transfer is aborted if “0” is written to this bit during DMA transfer.<br>0: Disables DMA transfer<br>1: Enables DMA transfer |

## 5.6 DMAC Function Details

### 5.6.1 DMAC transfer setting flow

*Figure 5-2 "DMAC transfer setting flow"* shows the flow for setting DMAC transfer.

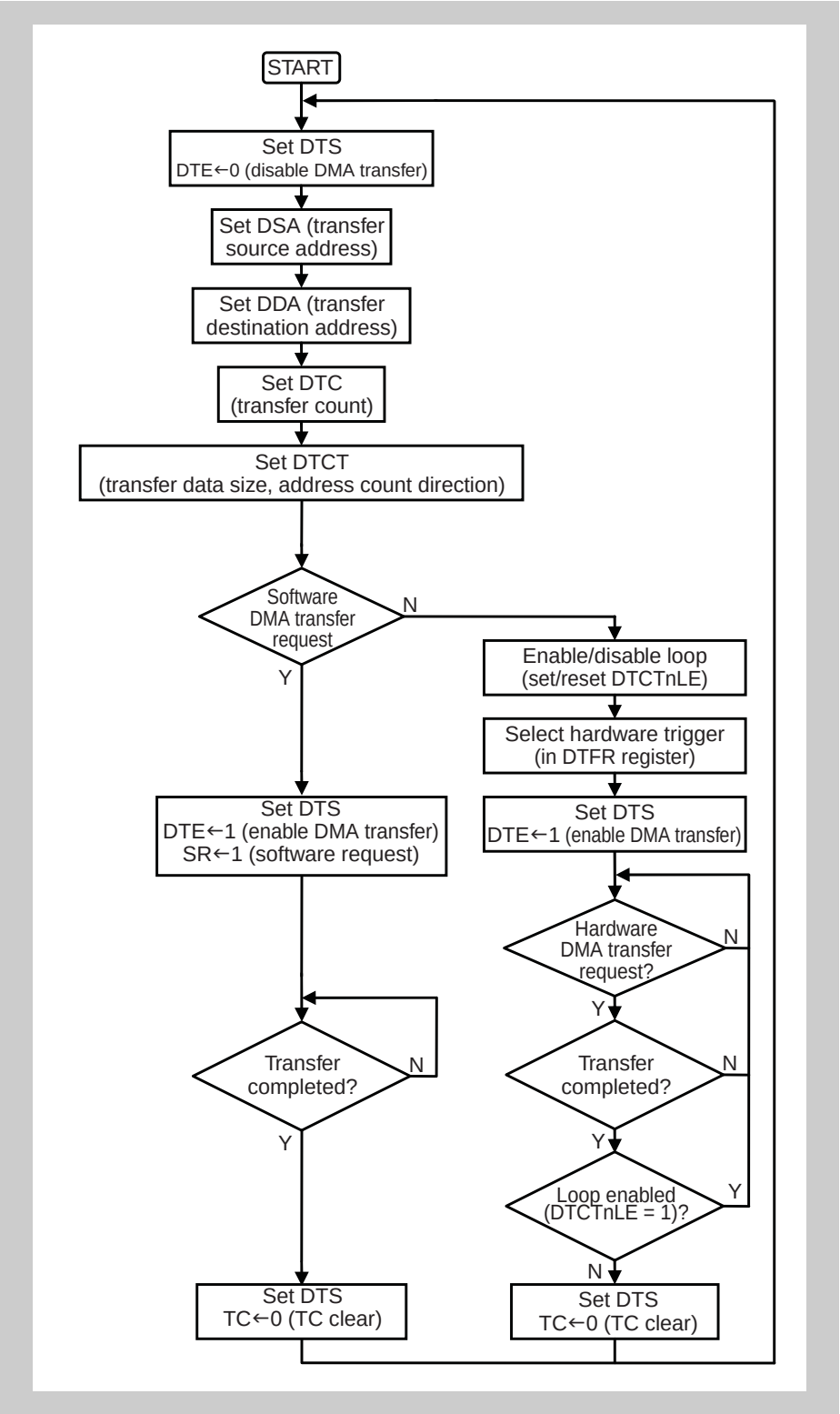



**Figure 5-2 DMAC transfer setting flow**

### 5.6.2 DMAC transfer modes

A single-transfer mode and a block transfer mode are supported as transfer modes.

In either mode, transfer is executed in two cycles (dual address transfer) and therefore, a read cycle and a write cycle are generated each time transfer is executed.

Note that the bus is not locked. Consequently, a CPU cycle may take place between the read and write cycles.

#### (1) Single transfer mode (when hardware DMA transfer request is generated)

When a hardware DMA transfer request is acknowledged, data of the transfer data size (8, 16, or 32 bits) is transferred. Each time transfer has been executed, the bus is released and the DMA controller waits for a DMA transfer request. At this time, the acknowledge, signal that indicates that the hardware DMA transfer request has been acknowledged, is also output.

Each time a hardware DMA transfer request has been acknowledged, transfer is executed once. This operation is repeated the number of times specified by the DMA transfer count register n (DTCn).

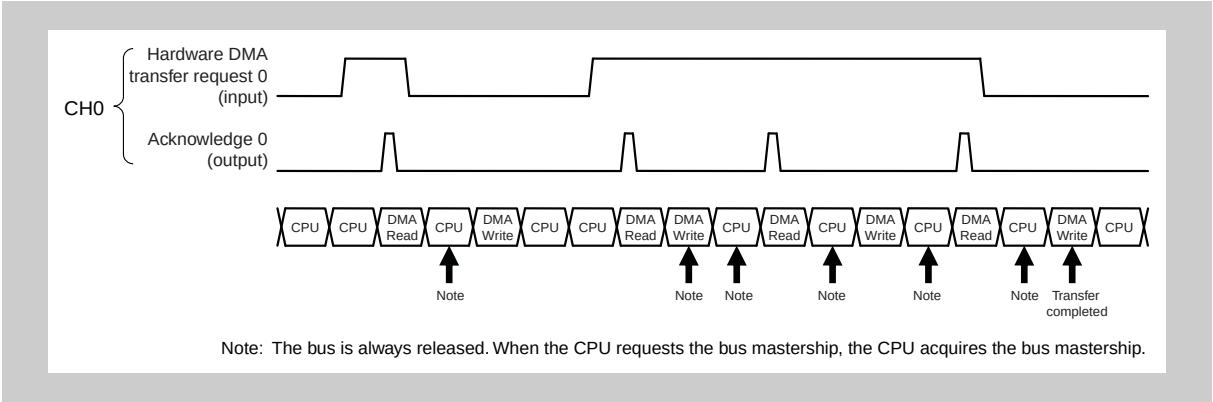



**Figure 5-3 Example of single transfer**

**(2) Block transfer mode (when software DMA transfer request is generated)**

When a software DMA transfer request is acknowledged, data of the transfer data size (8, 16, or 32 bits) is transferred. Each time transfer has been executed, the bus is released. At this time, the acknowledge signal, that indicates that a hardware DMA transfer request has been acknowledged, is not output.

Once a software DMA transfer request has been acknowledged, this operation is repeated the number of times specified by the DMA transfer count register n (DTCn). Because the priority is identified each time transfer is executed, the DMA cycle of a channel having the higher priority may interrupt a lower priority channel.

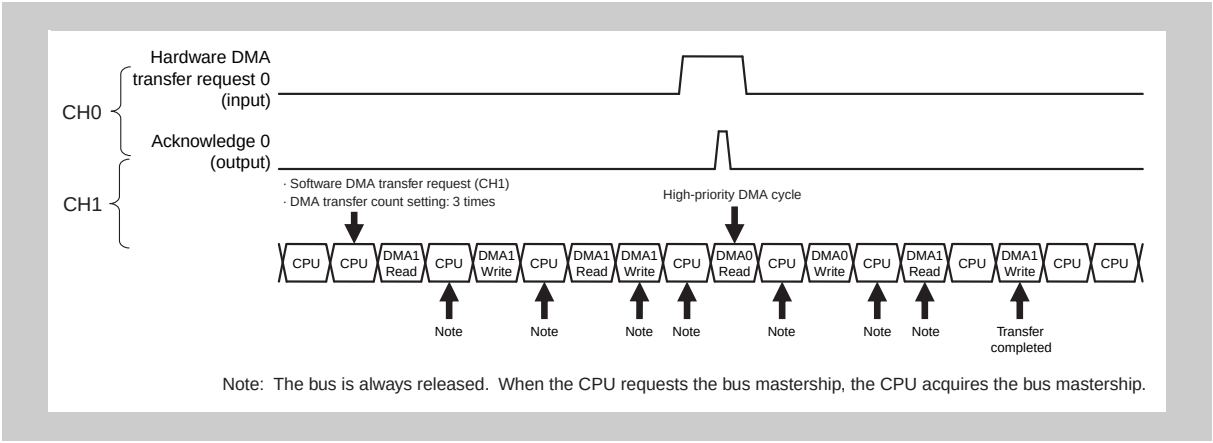



**Figure 5-4 Example of block transfer (DMA channel priority: CH0 (high) > CH1 (low))**

### 5.6.3 DMAC channel priority control

The priority of each channel is fixed and is as follows.

CH0 > CH1 > CH2 > CH3 > CH4 > CH5 > CH6 > CH7

If another DMA transfer request with a high priority is generated, the DMA transfer request with the higher priority always takes precedence.

When a software DMA transfer request is generated, the bus is also released each time a DMA cycle has been completed.

If a DMA transfer request with a high priority is generated, therefore, the DMA transfer request with the higher priority always takes precedence.

An example where another DMA transfer request with a high priority is generated when DMA transfer is executed is shown below.

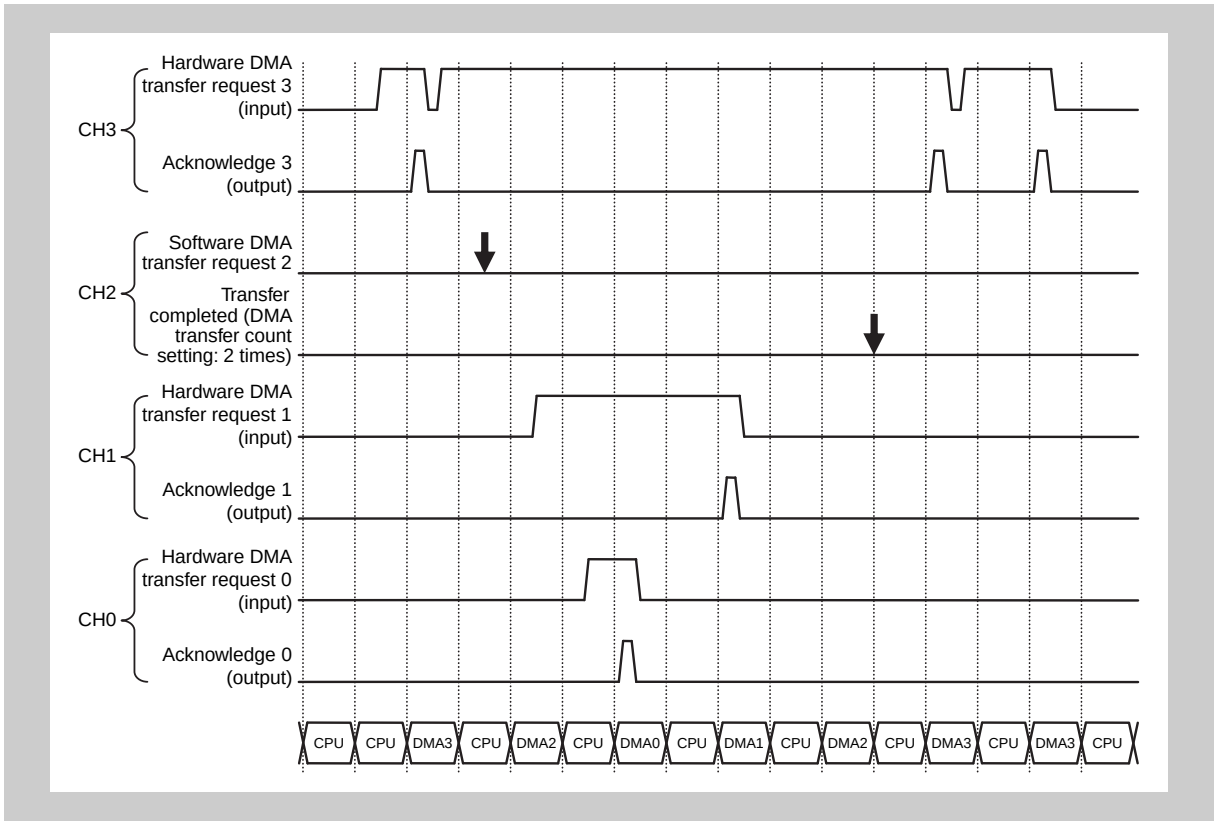



**Figure 5-5 Example of priority control**

### 5.6.4 Valid DMA transfer request conditions

Whether a DMA transfer request of channel n is acknowledged depends on the setting of the ERR and ADS bits of the DMA transfer request control register (DTRC), and the DTSnTC and DTSnDTE bits of the DMA transfer status register (DTSn).

*Table 5-16 "Valid DMA transfer request conditions of channel n"* shows the relationship between the setting of each of the above bits and whether a DMA transfer request is acknowledged.

**Table 5-16 Valid DMA transfer request conditions of channel n**

| Situation | DTSn. DTSnDTE | DTSn. DTSnTC | DTCTn DTCTnLE | DTRC.ERR | DTRC.ADS | DMA transfer request |
|---|---|---|---|---|---|---|
| When DMA transfer is disabled | 0 | X | X | X | X | Invalid |
| When DMA transfer error occurs | X | X | X | 1 | X | Invalid |
| When DMA transfer is aborted | X | X | X | X | 1 | Invalid |
| When DMA transfer is completed, loop disabled | 0 | 1 | 0 | X | X | Invalid |
| When DMA transfer is completed, loop enabled | 1 | 1 | 1 | 0 | 0 | Valid |
| When DMA transfer is enabled | 1 | 0 | 0 | 0 | 0 | Valid |

### 5.6.5 Aborting/resuming DMA transfer

**(1) Aborting or resuming DMA transfer for all channels through software**

By setting the DMA transfer abort bit (ADS) of the DMA transfer request control register (DTRC), the next DMA transfer and those that follow can be aborted.

During a DMA cycle, the next DMA transfer is aborted after the ongoing DMA cycle has been completed.
Note that the DMA transfer enable bit (DTSnDTE) and the software DMA transfer request bit (DTSnSR) of the DMA transfer status register (DTSn) are not cleared.

To resume the aborted DMA transfer, clear the ADS bit.
If a DMA transfer is requested at that point, the transfer of the channel having the highest priority at that time is executed.

To end DMA transfer, clear the DMA transfer request with the DTSnDTE bit cleared.

**(2) Aborting or resuming DMA transfer by using DMA transfer enable bit (DTE)**

By clearing the DMA transfer enable bit (DTSnDTE) of the DMA transfer status register (DTSn), the next DMA transfer and those that follow can be aborted.

During a DMA cycle, the next DMA transfer is aborted after the ongoing DMA cycle is completed. Note that the software DMA transfer request bit (DTSnSR) of DTSn is not cleared.

To resume the aborted DMA transfer, set the DTSnDTE bit.
If another channel is not executing DMA transfer at that point, the priority is identified as usual.
If another channel is executing DMA transfer, the priority is identified after that transfer has been completed.

To end DMA transfer, clear the DMA transfer request with the DTSnDTE bit cleared.

**(3) Aborting or resuming DMA transfer by using software DMA transfer request bit (DTSnSR)**

By clearing the software DMA transfer request bit (DTSnSR) of the DMA transfer status register, the next DMA transfer and those that follow can be aborted (DTS).

During a DMA cycle, the next DMA transfer is aborted after the ongoing DMA cycle has been completed.

To resume the aborted DMA transfer, set the DTSnSR bit.
If another channel is not executing DMA transfer at that point, the priority is identified as usual.
If another channel is executing DMA transfer, the priority is identified after that transfer has been completed.

### 5.6.6 Error response support

#### (1) Aborting DMA transfer by error response

When an error occurs at the DMA transfer source or destination, the DMA transfer abort bit (ADS) of the DMA transfer request control register (DTRC) to abort subsequent DMA transfers is set.

At the same time, the DMA transfer error status bit (ERR) is set and a SYSERR exception is generated towards the CPU.
The user can evaluate in which channel the error has occurred, by using the DMA transfer error flag (DTSnER) of the DMA transfer status register (DTSn), when the user has confirmed that ERR has been set.

In this case, note that, if an error response is acknowledged in the read cycle, the write cycle is not executed, but the transfer address and the transfer count are updated.

#### (2) Canceling transfer abort by error response

DMA transfer abort can be canceled by clearing the DMA transfer abort bit (ADS) and DMA transfer error status bit (ERR) of the DMA transfer request control register (DTRC).

Clear the DMA transfer enable bit (DTSnDTE) of the DMA transfer status register (DTSn) in advance, so that DMA transfer is not resumed after its abort has been canceled.

In the case of a software DMA transfer request, also clear the software DMA transfer request bit (DTSnSR).

### 5.6.7 Stand-by support

When a stop request is generated, DMA transfer stops until completion of the two DMA cycles currently being executed.

Unlike DMA transfer abort caused by software, this does not affect the DMA control register.

DMA transfer resumes upon cancellation of the stop request, and if a DMA request is already retained, that DMA transfer starts.

## 5.7 DTFR Function

The DMA trigger factor register (DTFR) selects DMA trigger factors from among interrupt signals, and requests DMAC for DMA transfer.
DTFRn registers are included for selecting the signals to be used for DMA transfer requests from among the 64 input trigger signals.

### 5.7.1 Features

**Number of transfer factors**
DMA transfer requests (for 8 channels) are selected from among 64 trigger signals.

**DMAC interface**
The DMA transfer request signal n is output.
The DMA transfer request signal n is cleared by an acknowledge signal from DMA.

**CPU interface**
The last transfer signal from DMA is output as a CPU interrupt signal.

**Clearing of transfer request**
A function that clears transfer request signals sent to DMA through register access is provided.

**Confirmation of transfer request**
A function that checks transfer request signals sent to DMA through register access is provided.

# 5.8 DTFR Control Registers

## 5.8.1 DTFRn – DMA trigger factor register

**Access** This register can be read or written in 16-bit units.

**Address** DTFR0: FFFF 7B00$_H$, DTFR1: FFFF 7B02$_H$, DTFR2: FFFF 7B04$_H$, DTFR3: FFFF 7B06$_H$, DTFR4: FFFF 7B08$_H$, DTFR5: FFFF 7B0A$_H$, DTFR6: FFFF 7B0C$_H$, DTFR7: FFFF 7B0E$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| REQEN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | IFCn[5:0] | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-17 DTFRn registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | REQEN | This bit enables or disables operation of the DMA source selector of channel n.<br>1: Enables operation of source selector<br>0: Stops operation of source selector. Does not issue DMA transfer request (DMARQ).<br>The settings of IFC6 to IFC0 are valid. Requests are always sampled. |
| 5 to 0 | IFCn[5:0] | These bits select the transfer source.<br>The set values are shown in the table in the first section of this chapter. |

**Caution** Stopping DMA channel n by DTFRn.REQEN = 0 does not clear any pending DMA request for that channel. In order to clear also a pending DMA request, set also DRQCLR.RQCRn = 1.

### 5.8.2 DRQCLR – DMA request clear register

**Access** This register can be read or written in 16-bit units.

**Address** FFFF 7B40$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RQCR7 | RQCR6 | RQCR5 | RQCR4 | RQCR3 | RQCR2 | RQCR1 | RQCR0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 5-18 DRQCLR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | RQCR7 | Setting “1” to this bit clears transfer request held in channel 7 to “0”. |
| 6 | RQCR6 | Setting “1” to this bit clears transfer request held in channel 6 to “0”. |
| 5 | RQCR5 | Setting “1” to this bit clears transfer request held in channel 5 to “0”. |
| 4 | RQCR4 | Setting “1” to this bit clears transfer request held in channel 4 to “0”. |
| 3 | RQCR3 | Setting “1” to this bit clears transfer request held in channel 3 to “0”. |
| 2 | RQCR2 | Setting “1” to this bit clears transfer request held in channel 2 to “0”. |
| 1 | RQCR1 | Setting “1” to this bit clears transfer request held in channel 1 to “0”. |
| 0 | RQCR0 | Setting “1” to this bit clears transfer request held in channel 0 to “0”. |

**Note** Writing “0” to bits 7 to 0 is ignored.

### 5.8.3 DRQSTR – DMA request check register

**Access** This register is read-only, in 16-bit units.

**Address** FFFF 7B44$_H$

**Initial Value** 0000$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RQST7 | RQST6 | RQST5 | RQST4 | RQST3 | RQST2 | RQST1 | RQST0 |
| R | R | R | R | R | R | R | R |

**Table 5-19 DRQSTR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | RQST7 | 1: Request issued (DMA transfer request signal 7 is “1”),<br>0: No request (DMA transfer request signal 7 is “0”) |
| 6 | RQST6 | 1: Request issued (DMA transfer request signal 6 is “1”),<br>0: No request (DMA transfer request signal 6 is “0”) |
| 5 | RQST5 | 1: Request issued (DMA transfer request signal 5 is “1”),<br>0: No request (DMA transfer request signal 5 is “0”) |
| 4 | RQST4 | 1: Request issued (DMA transfer request signal 4 is “1”),<br>0: No request (DMA transfer request signal 4 is “0”) |
| 3 | RQST3 | 1: Request issued (DMA transfer request signal 3 is “1”),<br>0: No request (DMA transfer request signal 3 is “0”) |
| 2 | RQST2 | 1: Request issued (DMA transfer request signal 2 is “1”),<br>0: No request (DMA transfer request signal 2 is “0”) |
| 1 | RQST1 | 1: Request issued (DMA transfer request signal 1 is “1”),<br>0: No request (DMA transfer request signal 1 is “0”) |
| 0 | RQST0 | 1: Request issued (DMA transfer request signal 0 is “1”),<br>0: No request (DMA transfer request signal 0 is “0”) |

# Chapter 6 Flash Memory

The following V850E2/Fx4-G devices are equipped with internal flash memory as follows:

| Series name | Product name | Code flash | Data flash |
|---|---|---|---|
| **FE4-G:** | | | |
| FF4-G-256K | μPD70F4177 | 256 KB | 32 KB |
| FF4-G-512K | μPD70F4178 | 512 KB | |
| **FG4-G:** | | | |
| FG4-G-256K | μPD70F4179 | 256 KB | 32 KB |
| FG4-G-512K | μPD70F4180 | 512 KB | |

The code flash memory is attached to the dedicated instruction fetch bus of the V850E2 CPU core. It is used for non-volatile storage of program code and constant data.

The data flash memory is accessible via the memory interface bus. It holds nonvolatile user's data, which are subject to be altered during normal program operation.

The flash memory can be written in different ways:

- mounted on the target board by connecting a dedicated flash programmer to the target system (Serial-Programming)
- by the microcontroller's application software (Self-Programming)

Additionally the flash memory is equipped with a configuration area to hold various configuration settings. Via these options start-up configurations can be set for e.g. the Watchdog Timer. The flash configuration options can be written by use of an external flash programmer and in Self-Programming mode. They are not accessible via the normal CPU address space.

## 6.1 Code Flash Memory Overview

### 6.1.1 Code flash memory features

- Erase/write with single power supply
- Two programming modes:
  - Serial-Programming with flash programmer using dedicated serial interfaces
  - Flash memory programming by Self-Programming

### 6.1.2 Code flash memory map

The microcontroller's internal code flash memory area is divided into blocks of 32 KB blocks and can be programmed/erased in block units.

Following tables list the block structures and address assignments for all V850E2/Fx4-G devices with code flash memory.

**Table 6-1 Code flash memory configuration**

| | | | |
|---|---|---|---|
| | Block 15 (32 KB) | **007 FFFF$_H$** | **Address** |
| | | **...** | |
| | | **007 8000$_H$** | |
| | ... | | |
| | | **...** | |
| | | | |
| | Block 12 (32 KB) | **0006 7FFF$_H$** | |
| | | **...** | |
| | | **006 0000$_H$** | |
| | Block 11 (32 KB) | **0005 FFFF$_H$** | |
| | | **...** | |
| | | **0005 8000$_H$** | |
| | ... | | |
| | | **...** | |
| | | | |
| | Block 8 (32 KB) | **0004 7FFF$_H$** | |
| | | **...** | |
| | | **0004 0000$_H$** | |
| Block 7 (32 KB) | Block 7 (32 KB) | **0003 FFFF$_H$** | |
| | | **...** | |
| | | **0003 8000$_H$** | |
| ... | ... | | |
| | | **...** | |
| | | | |
| Block 1 (32 KB) | Block 1 (32 KB) | **0000 FFFF$_H$** | |
| | | **...** | |
| | | **0000 8000$_H$** | |
| Block 0 (32 KB) | Block 0 (32 KB) | **0000 7FFF$_H$** | |
| | | **...** | |
| | | **0000 0000$_H$** | |
| 256 KB | 512 KB | **Code flash size** | |
| 32/64/96/128 KB | | **Boot swap cluster sizes** | |
| Fx4-G-256K:<br>• µPD70F4177<br>• µPD70F4179 | Fx4-G-512K:<br>• µPD70F4178<br>• µPD70F4180 | **Products** | |

### 6.1.3 Data flash memory map

The data flash memory is organized in blocks of 32 byte size.

Following tables list the block structures and address assignments for all V850E2/Fx4-G devices with data flash memory.

**Table 6-2 V850E2/Fx4-G data flash memory**

| Product | Data flash size | Number of 32 byte blocks | Address range |
|---|---|---|---|
| • Fx4-G | 32 KB | 1024 | FE00 $0000_H$ - FE00 $7FFF_H$ |

## 6.2 Code Flash Memory functional Outline

**Serial-Programming** The internal flash memory of the microcontroller can be written by using the erase and write functions of a flash programmer, also while the microcontroller has already been mounted on the target system.

**Self-Programming** The Self-Programming facility allows rewriting of the flash memory by the user program. It is ideal for program updates after production and shipment, since no additional programming equipment is required. During Self-Programming some software services as well as interrupt serving can still be in operation, e.g to sustain communication with other devices.

While the Self-Programming mode can be initiated from the normal operation mode the external flash programmer mode is entered immediately after release of a system reset.
Refer to *"Flash memory programming control"* in the section *"Flash Programming with Flash Programmer"* in this chapter for details on how to enter normal operation or Serial-Programming mode.

**Configuration area** The flash memory contains a configuration area, used to store the settings of security and protection functions, initial settings for some modules.

**Boot swap** A boot swap function makes safe re-programming of the flash memory possible and is used to maintain an operable software version, even if re-programming fails for any reason, e.g. in a power fail situation.
For further information concerning boot swapping refer to *"Secure Self-Programming (boot cluster swapping)"* below in this chapter.

**Protection** A set of protection flags can be specified during flash memory programming to prohibit access to the flash memory in different ways, such as read-out, write and erase protections for the external programmer interface. By these means the flash memory can be protected against read-out and rewrite of the flash memory content by unauthorized persons.
For further information concerning data protection refer to the chapter *"Code Protection and Security"*.

**Table 6-3 Flash memory write methods**

| Environment | Interface | Outline | Operation Mode |
|---|---|---|---|
| Serial-Programming | Dedicated serial interface | Flash memory programming is done by an external flash programmer.<br>The device is mounted on the target system. The communication between the device and the flash progammer is using a dedicated serial interface.<br>For details refer to the section *"Flash Programming with Flash Programmer"* in this chapter. | Flash memory programming mode |
| Self-Programming | Self-Programming library | Flash memory can be rewritten by executing a user program that has been written to the flash memory in advance by means of Serial-Programming. The Self-Programming library provides all necessary functions to be called by the application software.<br>For details refer to the section *"Code Flash Self-Programming"* in this chapter. | Normal operation mode |

*Table 6-4 "Basic functions for flash memory modifications" on page 231* summarizes the functions used to modify flash memory content.

**Table 6-4 Basic functions for flash memory modifications**

| Function | Functional outline | Support (√: supported, ×: not supported) | |
|---|---|---|---|
| | | Serial-Programming | Self-Programming |
| Block erasure | The contents of specified memory blocks are erased. | √ | √ |
| Write | Writing to specified addresses, and a verify check to see if write level is secured are performed. | √ | √ |
| Verify | Data read from the flash memory is compared with data transferred from the flash programmer. | √ | x[a] |
| Checksum | Microcontroller internally calculated CRC checksum over the entire flash memory content is compared with the checksum calculated by the serial programmer | √ | x |
| Blank check | The erasure status of the entire memory is checked. | √ | √ |
| Protection settings | Following functions can be prohibited:<br>• block erase<br>• write<br>• read<br>• rewriting of the boot cluster<br>• flash shield | √ | √[b] |

a) Can be carried out by the user's program.

b) Except protection against rewriting of the boot cluster all other protections have no effect in Self-Programming mode.
Protection settings can be activated in Self-Programming mode. Already activate protection settings can not be deactivated.

The following table lists the available flash memory protection functions.

For details refer to the chapter *"Code Protection and Security"*.

**Table 6-5 Protection functions**

| Function | Functional outline | Applicable (√: applies, ×: doesn't apply) | |
|---|---|---|---|
| | | Serial-Programming | Self-Programming |
| Block erase command prohibit | Erasure of single blocks impossible. | √ | × |
| Program command prohibit | Erasure and rewrite of single blocks impossible. | √ | × |
| Read command prohibit | Read-out of any flash content impossible. | √ | × |
| Rewriting boot area prohibit | Erasure (by block or chip erase) or writing of the boot cluster impossible. | √ | √ |
| Flash shield | Write/erase protection outside a defined window. | √ | √ |

### 6.2.1 Code flash memory erasure and rewrite

**Erasure** Each 32 KB flash memory block can be erased separately.

**Rewrite** In Self- and Serial-Programming mode it is possible to rewrite the flash memory in smaller units than one block. Once a complete block has been erased it can be rewritten in units of 256 byte. Each unit can be rewritten only once after erasure of the complete block.

## 6.3 Data Flash Memory

The V850E2/Fx4-G Series products contain data flash in addition to the code flash.

### 6.3.1 Data flash memory features

The data flash has the following features:

- data flash memory in 32 byte blocks
- write access in 16-bit steps
- erase in 32 byte blocks
- write, erase operations to the data flash while application code can be executed from code flash

### 6.3.2 Data flash reading and writing

The data flash can be read and written by using an external flash programming tool or the data flash access library.

Programming during normal operation is achieved by using the data flash access software library. For details refer to FDL (Data Flash Access Library).

**Data flash read** The data flash can only be read 16 bit units on halfword aligned addresses.

Due to the data flash cell structure (complementary read) the read value of an erased data flash is undefined. Use the blank check function of the provided FDL (Data Flash Access Library) for checking if the read value belongs to an erased flash.

**Note** Once a Data Flash location is written (even with value $FFFF_H$) it can not be overwrite without a previous erase.

## 6.4 Flash Programming with Flash Programmer

A dedicated flash programmer can be used for external writing of the flash memory in Serial-Programming mode.

**Serial-Programming**

During Serial-Programming the target microcontroller remains mounted on its board. The board is equipped with a connector, that connects the flash programmer to the target microcontroller.

The microcontroller must be fully functional and operated in Serial-Programming mode, in particular

- All external power supplies must be active.
- The external resonator must be connected to the X1/X2 pins.

All other necessary microcontroller configurations are conducted by the on-chip firmware, that is processed in Serial-Programming mode.

**Caution** Connecting the flash programmer to the on-board microcontroller may yield conflicts with other sginals. Pay attention to the hints given in section *"Potential conflicts with on-board signal connections" on page 239*.

### 6.4.1 Programming environment

The recommended environment to write data to the flash memory of the microcontroller is shown below.

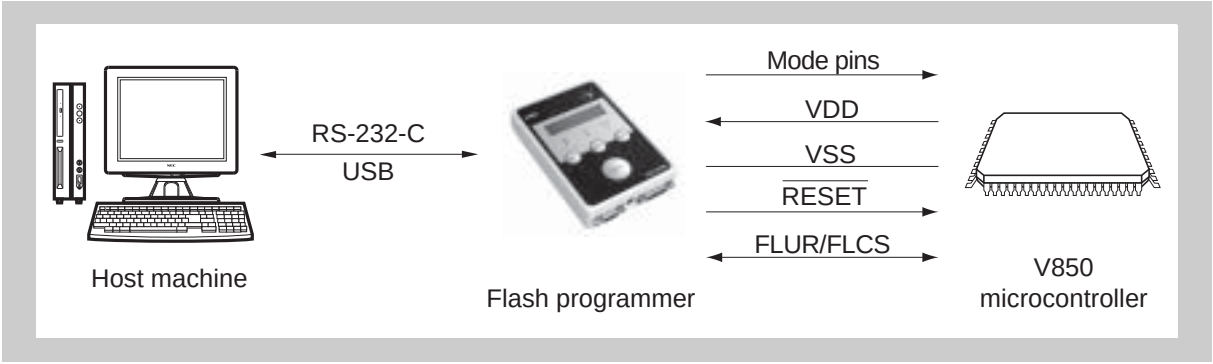



**Figure 6-1 Environment to write program to flash memory**

A host machine is required for configure the flash programmer.
The used flash programmer may also feature a stand-alone mode, so a host machine may not be necessary.

Following dedicated microcontroller serial interfaces can be used as the interface between the flash programmer and the microcontroller:

- single-wire asynchronous serial interface FLUR0
- clocked serial interface FLCS0

**Mode pins** The mode pins are used to set the microcontroller into flash programming mode.
Refer to the section *"Operation modes"* in the chapter *"CPU System Functions"* for further information.

**Note** In normal operation mode, i.e. not in flash programming mode, the serial interfaces FLUR0 and FLCS0 are not available. The used ports in flash programming mode are specified in the following sections.
These are automatically configured for communicating with the flash programmer in flash programming mode.

### 6.4.2 Communication modes

#### (1) Asynchronous flash programming interface FLUR0

The single-wire asynchronous Serial-Programming interface FLUR0 uses following port for connecting to the flash programmer:

- JP0_0: reception/transmission data FLUR0RTX

The external flash programmer offers various choices of available baudrates.

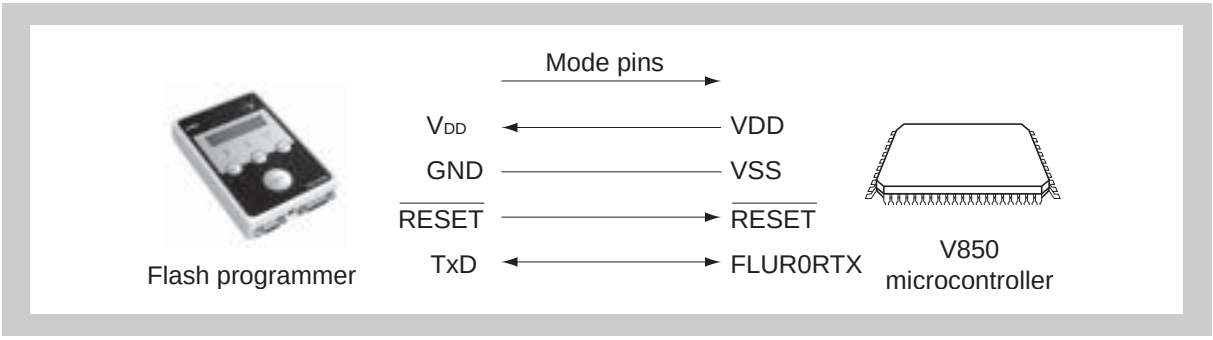



**Figure 6-2 Communication with flash programmer via FLUR0**

#### (2) Synchronous flash programming interface FLCS0

The synchronous Serial-Programming interface FLCS0 uses following ports for connecting to the flash programmer:

- JP0_0: serial data input FLCS0SI
- JP0_1: serial data output FLCS0SO
- JP0_2: serial data clock input FLCS0SCI

The external flash programmer offers various choices of available clock rates.

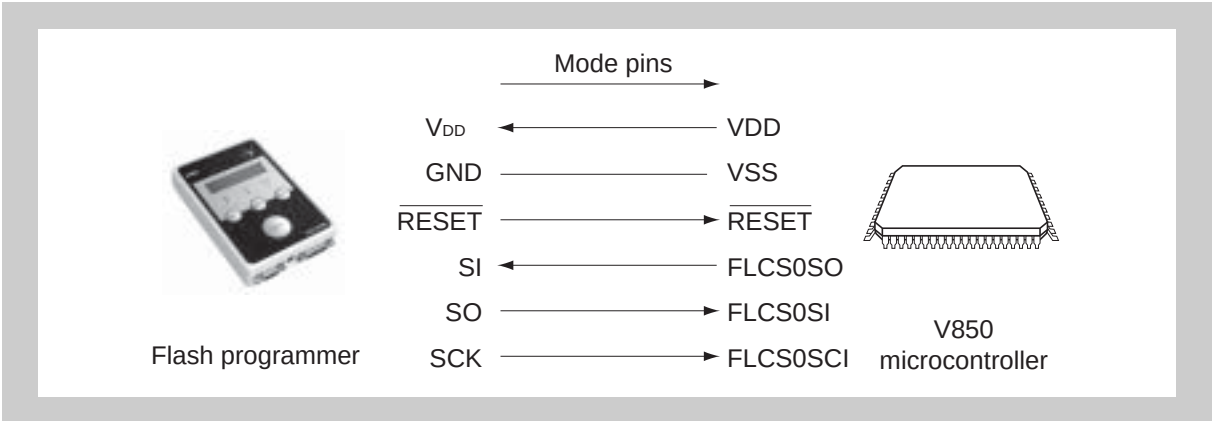



**Figure 6-3 Communication with flash programmer via FLCS0**

The flash programmer outputs the serial data clock SCK and the microcontroller operates as a slave.

### 6.4.3 Pin connection with flash programmer PG-FP5

A connector must be mounted on the target system to connect the flash programmer for Serial-Programming. In addition, functions to switch between the normal operation mode and flash memory programming mode and to control the microcontroller's $\overline{\text{RESET}}$ pin must be provided on the board.

When the microcontroller flash memory programming mode is set, all the pins not used for flash memory programming are in the same status as immediately after reset.

**Mode pins** The mode pins are used to set the microcontroller into flash programming mode.
Refer to the section *"Operation modes"* in the chapter *"CPU System Functions"* for further information.

When the microcontroller flash memory programming mode is set, all the pins not used for flash memory programming are in the same status as immediately after reset.

If the PG-FP5 is used as the flash programmer, connect the PG-FP5 target interface connector to the microcontroller as follows:

**Table 6-6 Connection of microcontroller flash programmer PG-FP5**

| Flash programmer FG-FP5 connection pin | | | Microcontroller signal (port) name | | | |
|---|---|---|---|---|---|---|
| Signal name | I/O | Function | FLUR0 | | FLCS0 | |
| | | | Signal | Port | Signal | Port |
| SO/TxD | O | • FLUR0: receive/transmit data<br>• FLCS0: transmit data | FLUR0RTX | JP0_0 | FLCS0SI | JP0_0 |
| SI/RxD | I | Receive data | leave open | | FLCS0SO | JP0_1 |
| SCK | O | Transfer clock | leave open | | FLCS0SCI | JP0_2 |
| CLK | O | Clock to microcontroller | leave open | | leave open | |
| | | | leave open | | leave open | |
| $\overline{\text{RESET}}$ | O | Reset signal | $\overline{\text{RESET}}$ | – | $\overline{\text{RESET}}$ | – |
| FLMD0 | I | Mode selection | FLMD0 | – | FLMD0 | – |
| FLMD1 | I | Mode selection | FLMD1[a] | P0_1 | FLMD1[a] | P0_1 |
| H/S | I | Handshake signal | leave open | | leave open | |
| $V_{DD}$ | I | Microcontroller supply voltage monitoring | Power supply of JP0 port group buffers[b] | | Power supply of JP0 port group buffers[b] | |
| $V_{DD2}$ | – | Supply voltage | leave open | | leave open | |
| $V_{PP}$ | – | Flash programming voltage | leave open | | leave open | |

Table 6-6 Connection of microcontroller flash programmer PG-FP5

| Flash programmer FG-FP5 connection pin | | | Microcontroller signal (port) name | | | |
|---|---|---|---|---|---|---|
| Signal name | I/O | Function | FLUR0 | | FLCS0 | |
| | | | Signal | Port | Signal | Port |
| GND | – | Ground | VSS | | VSS | |
| VDE | – | Reserved | leave open | | leave open | |
| RFU-1 | – | Reserved | leave open | | leave open | |

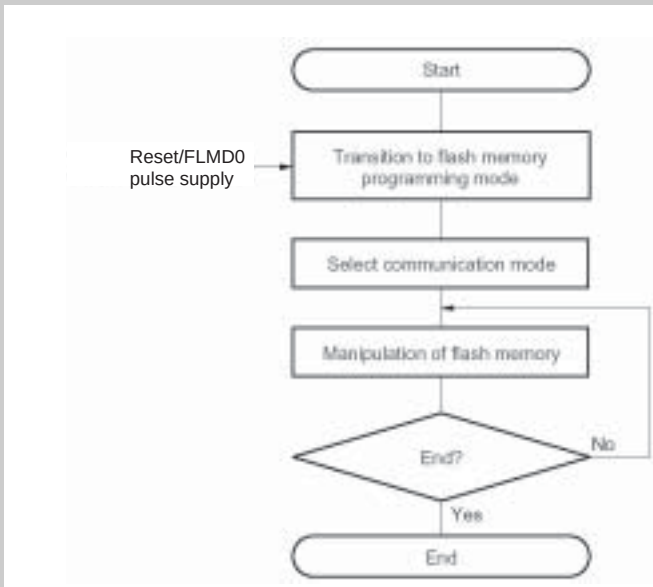
a) If FLMD1 is fixed to low level on the target board, the programmer's FLMD1 signal can be left unconnected.
b) Refer to the chapter *"Power Supply"* to obtain the correct microcontroller's power supply pins for the JP0 port group.

Refer also to the section *"Operation Modes"* in chapter *"CPU System Functions"* for more details about flash programming mode setting.

For details concerning the PF-FP5 programmer, refer to the PG-FP5 User's Manual, document number R20UT0008EJxxxx (xxxx denotes the current version number).

### 6.4.4 Flash memory programming control

The procedure to program the flash memory is illustrated below.




Note: A reset pulse is required to initiate the selection of the flash programming mode.

Figure 6-4 Flash memory programming procedure

### (1) Operation mode control

To rewrite the contents of the flash memory by using the flash programmer, set the microcontroller in the flash memory programming mode.

To enter the Serial-Programming mode, i.e. on-board programming by an external flash programmer, the FLMD0 pin has to be supplied with VDD and FLMD1 with VSS level at $\overline{\text{RESET}}$ release.

In the normal operation mode, VSS is input to the FLMD0 pin. A pull-down resistor at the FLMD0 pin ensures normal operation mode if no flash programmer is connected.

**Note** Refer to the sections *"Operation modes"* and " Mode pins and JP0 connections" in the *"CPU System Functions"* chapter for details about operation modes settings.

An example of connection of the FLMD0 and FLMD1 pins is shown below. FLMD1 can be connected to ground via a resistor. Alternatively the FLMD1 pin may also be connected directly to the FLMD1 signal of the flash programmer.

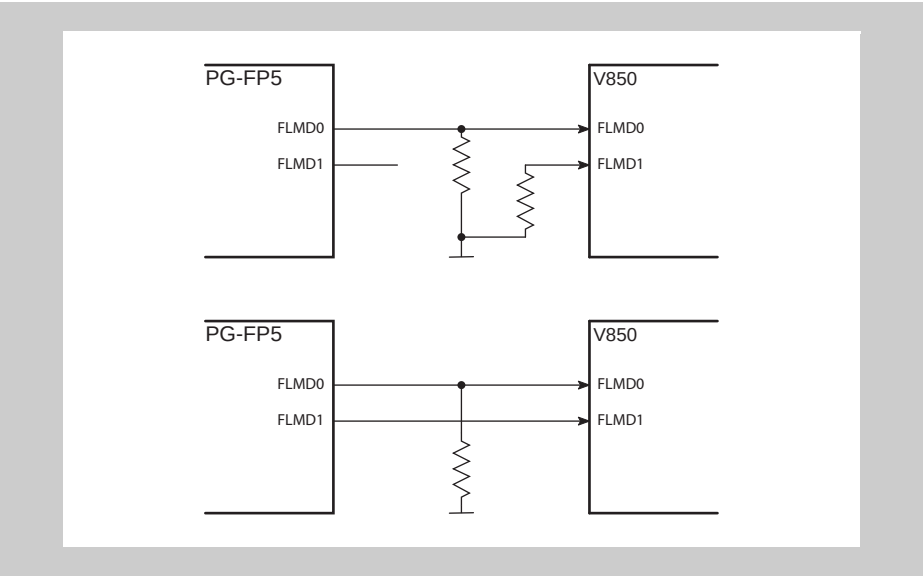



**Figure 6-5 Example of connection to flash programmer PG-FP5**

Once started in normal operation mode (FLMD0 = 0), FLMD0 pin is used for enabling Self-Programming. Refer also to *6.5 "Code Flash Self-Programming" on page 244* .

### (2) Potential conflicts with on-board signal connections

**Serial I/O signals**

If other devices are connected to the serial interface pins in use for flash memory programming in Serial-Programming mode take care that the concerned signals do not conflict with the signals of the flash programmer and the microcontroller. Output pins of the other devices must be isolated or set in high impedance state. Ensure that the other devices do not malfunction because of flash programmer signals.

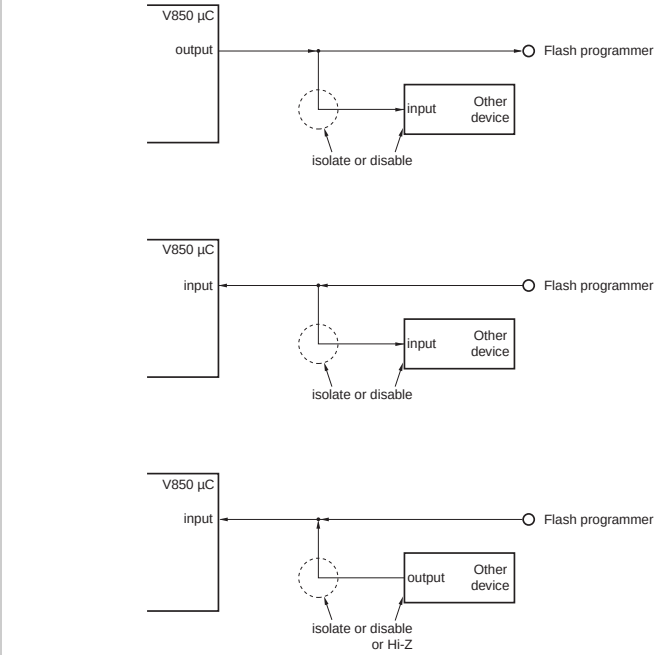



**Figure 6-6 Potential conflicts with serial interfaces signals**

**$\overline{\text{RESET}}$**

Pay attention in particular if the flash programmer's $\overline{\text{RESET}}$ signal is connected also to an on-board reset generation circuit. The reset output of the reset generator may ruin the flash programming process and may need to be isolated or disabled.

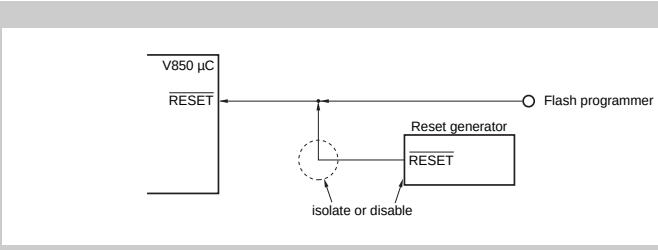



**Figure 6-7 Potential conflict with $\overline{\text{RESET}}$**

**Ports** The V850 port pins adopts following status during Serial-Programming:

Ports used for programming are configured as FLUR0 respectively FLCS0 pins.

All other pins remain in their default state after reset release.

In case the default state after reset of the pins not used for programming is in port or high -impedance output port, pay attention to other devices connected to these pins. If these devices require defined levels at the pins, the ports may have to be connected to VDD or VSS via a resistors.

**Oscillators** Connect all oscillator pins in the same way as in the normal operation mode.

**$\overline{\text{DCUTRST}}$** During flash memory programming, input a low level to $\overline{\text{DCUTRST}}$ (port JP0_4) or leave it open. Do not input a high level.

**Power supply** Supply the same power to all power supply pins, including reference voltages, power regulator pins, etc., as in the normal operation mode.

### (3) Selection of the communication mode

The communication interface is chosen by applying a specified number of pulses to the FLMD0 pin after reset release. Note that this is handled by the flash programmer.

*Figure 6-8 “Selection of communication mode” on page 241* gives an example how the FLCS0 is established for the communication between the flash programmer and the microcontroller.

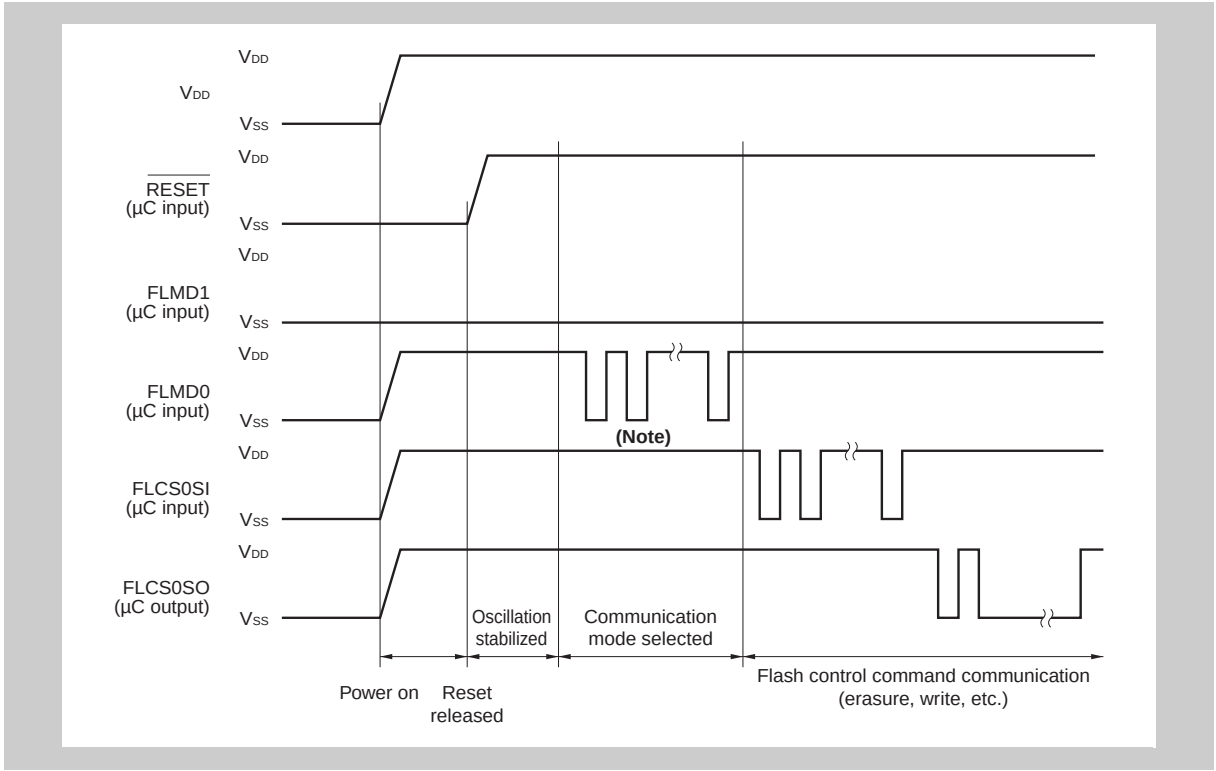



**Figure 6-8 Selection of communication mode**

**Note** The number of clocks to be inserted differs depending on the chosen communication mode. For details, refer to *Table 6-7 “FLMD0 pulses for communication mode setting” on page 241* .

**Table 6-7 FLMD0 pulses for communication mode setting**

| FLMD0 pulses | Communication mode | Remarks |
|---|---|---|
| 0 | FLUR0 | Communication rate: 9600 bps (after reset), LSB first |
| 8 | FLCS0 | Microcontroller performs slave operation, MSB first |
| Other | – | Setting prohibited |

When FLUR0 has been selected after reception of the FLMD0 pulses with 9600 bps, the flash programmer changes the baudrate according to the user’s choice via the flash programmer’s user interface.

### (4) Communication commands

The flash programmer sends commands to the microcontroller. Depending on the commands, the microcontroller returns status information or the requested data.

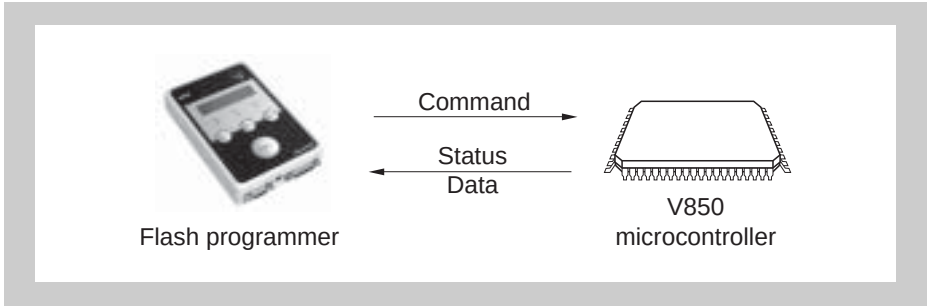



**Figure 6-9 Communication commands exchange**

The following table lists the flash memory control commands of the microcontroller. All these commands are issued by the flash programmer, and the microcontroller performs the corresponding processing.

**Table 6-8 Flash memory control commands (1/2)**

| Classification | Command name | Support | | Function |
|---|---|---|---|---|
| | | FLCS0 | FLUR0 | |
| Blank check | Block blank check | √ | √ | Checks erasure status of entire memory. |
| Erase | Block erase | √ | √ | Erases the memory contents of specified blocks. Note that the configuration area remains untouched. |
| Write | Write | √ | √ | Writes data by specifying write address and number of bytes to be written, and executes verify check. |
| | Chip write | √ | √ | Writes all flash memory contents |
| Read | Read | √ | √ | Reads data by specifying write address and number of bytes to be read. |
| Verify | Verify | √ | √ | Compares input data with all memory contents. |
| ID | Set ID code | √ | √ | Set the On-chip Debug ID to the registers OCDIDL, OCDIDM, OCDIDH |
| | Get ID code | √ | √ | Reads the On-chip Debug ID from the registers OCDIDL, OCDIDM, OCDIDH |
| CRC check | CRC check | √ | √ | Calculates a checksum over a specified number of flash blocks of the code flash or the data flash. Note that this checksum does not cover the configuration area. |
| | Chip CRC check | √ | √ | Calculates a checksum over the entire flash memory, including code flash, data flash and configuration area. |

**Table 6-8 Flash memory control commands (2/2)**

| Classification | Command name | Support | | Function |
|---|---|---|---|---|
| | | FLCS0 | FLUR0 | |
| Flash configuration option | Set flash configuration option | √ | √ | Set the flash configuration options to register OPBT0 |
| | Get flash configuration option | √ | √ | Reads the flash configuration options from register OPBT0 |
| Protection | Protection setting | √ | √ | Sets protection against chip erasure, block erasure, and writing. |
| | Get protection settings | √ | √ | Reads the protection settings. |
| System setting and control | Reset | √ | √ | Escapes from each status. |
| | Oscillation frequency setting | √ | √ | Sets oscillation frequency. |
| | Baudrate setting | – | √ | Sets baudrate when UART is used. |
| | Silicon signature | √ | √ | Reads silicon signature information. |

## 6.5 Code Flash Self-Programming

This V850 microcontroller supports a flash macro service that allows the user program to rewrite the internal flash memory by itself.

By using this flash macro service and a Self-Programming library (SPL), provided by Renesas, the user's program is able to rewrite the flash memory with data, transferred in advance to the internal RAM or the external memory.

Thus the user program can be upgraded and constant data can be rewritten in the field.

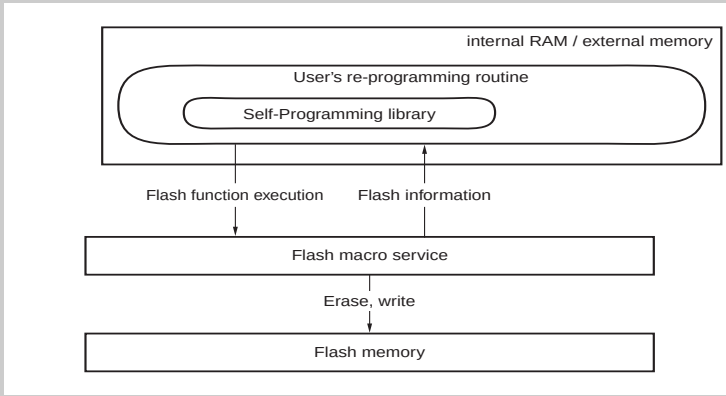



**Figure 6-10 Concept of Self-Programming**

During Self-Programming access to the flash memory is not possible. Thus program execution is only possible by instruction fetching from internal RAM or external memory.

Consequently the instructions of user re-programming software routines, which shall remain in operation during the Self-Programming procedure, must be copied from the flash memory to the internal RAM or external memory prior to activating the Self-Programming. Since interrupt processing by using the interrupt vectors in the flash memory is also impossible during Self-Programming, interrupt acknowledges need to be re-routed to interrupt vectors in the internal RAM (refer to *"Interrupt handling during flash Self-Programming" on page 251*).

It is recommended to refer to the User's Manual "<tbd>" for comprehensive information concerning flash Self-Programming. This document explains also the functions of the Self-Programming library.

### 6.5.1 Self-Programming enable

The Self-Programming function can be started out of the normal user mode of the microcontroller.

Self-Programming must be in particular enabled in order to avoid unintended re-programming of the flash. Two ways to enable Self-Programming are provided:

- by setting the external FLMD0 pin to high level
  This requires some external components or wiring, e.g. connecting an output port to FLMD0.
- by setting the internal register bit FLMDCNT.FLMDPUP
  This way does not need any special external components or wiring.

The following register is used to enable Self-Programming internally by software.

**Note** The FLMD0 pin must be connected to ground via a 82 kΩ resistor. Refer to the section *"Mode pins and JP0 connections"* in the *"CPU System Functions"* chapter and the Data Sheet for further details.

#### (1) FLMDCNT – FLMD control register

This register controls an internal pull-up respectively pull-down resistor, connected to the FLMD0 pin, and thus enables respectively disables Self-Programming.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register FLMDPCMD.
Refer to the section *"Write protected Registers"* in chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 8-bit units.

**Address** FF43 $8000_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | FLMDPUP |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 6-9 FLMDCNT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | FLMDPUP | FLMD0 pull-up/pull-down control<br>0: pull-down resistor active at FLMD0 (Self-Programming mode disabled)<br>1: pull-up resistor active at FLMD0 (Self-Programming mode enabled) |

### 6.5.2 Self-Programming library functions

Code flash memory Self-Programming by the user's program is supported by the Self-Programming library.

This library provides a set of C function calls to carry out basic functions like

- erase and rewrite of the flash memory
- boot cluster swapping, including definition of boot clusters
- setting of protection flags
- obtain various information concerning the code flash memory

Detailed information how to use the library functions is given in the User's Manual "<tbd>".

### 6.5.3 Self-Programming internal RAM occupancy

During Self-Programming the upper 4 KB of the internal RAM (address FEDF $F000_H$ - FEDF $FFFF_H$) are occupied by the Self-Programming Library. Thus the content of this 4 KB RAM is altered during Self-Programming and may have to be recovered by the user's program.

**Note** Additional RAM may be necessary for intermediate storage of user data and code to be copied from flash memory to RAM during Self-Programming.

### 6.5.4 Secure Self-Programming (boot cluster swapping)

The V850 flash microcontrollers support a mechanism to swap a cluster of code flash memory blocks, starting from address 0000 $0000_H$, with another cluster of the same size, located immediately above the first one.

**Boot swap cluster**

A group of boot blocks to be swapped. The cluster of blocks starting at address 0000 $0000_H$ is named active boot swap cluster, since it contains the entry point of the user's program at the default reset vector 0000 $0000_H$.

**Boot swap flag**

Which of the two clusters is the active boot cluster is controlled by the boot swap flag, that can be defined during flash programming via the Self-Programming library.
The boot swap flag is stored in the flash memory configuration area.

*Figure 6-11 "Boot swap cluster swapping function" on page 247* shows an example of the boot block swapping function with a cluster size of 4 flash memory blocks. After inverting the boot_flag - it becomes not(boot_flag) - blocks 4 to 7 become the active boot cluster. Thus after next reset release the user's program starts from the new boot swap cluster.

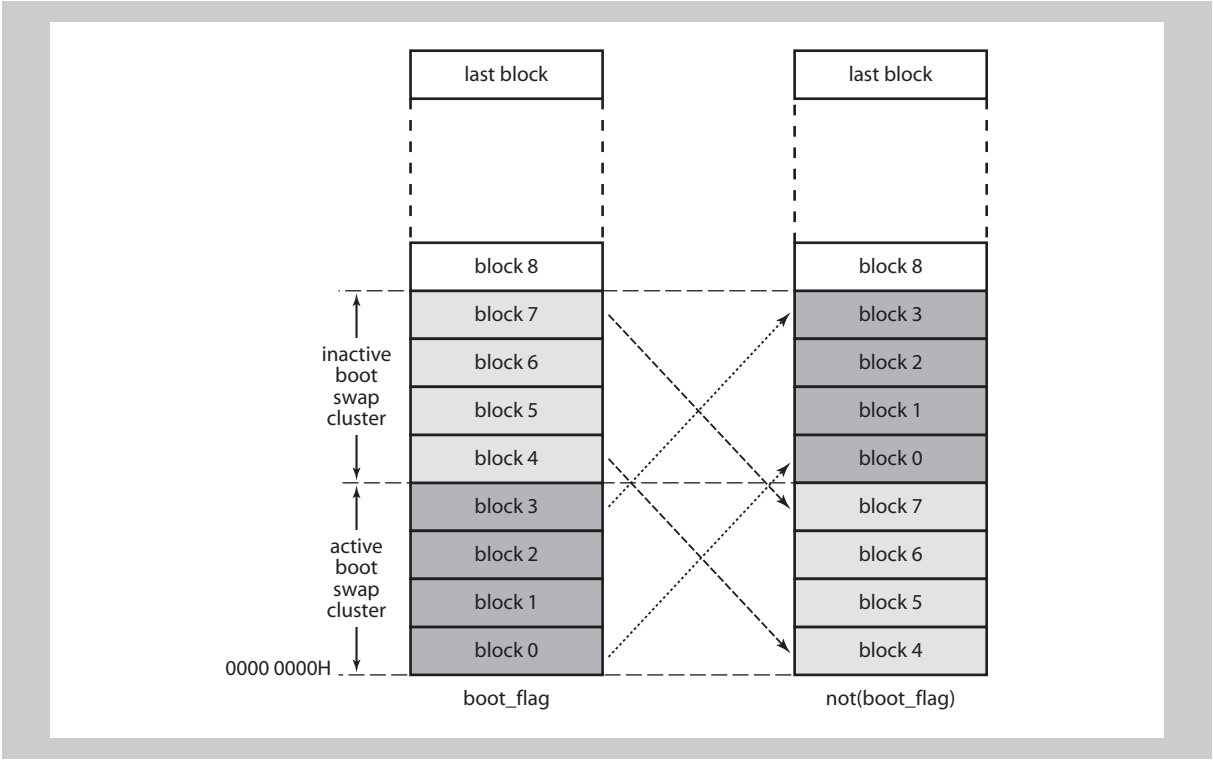



**Figure 6-11 Boot swap cluster swapping function**

**Secure Self-Programming**

The boot cluster swapping function enables secure Self-Programming. In case the boot code shall be rewritten, the new code can be written to the inactive boot cluster, while the boot_flag remains in its previous state.
If rewriting of the boot cluster has been completed successfully, the boot_flag can be inverted, making the new boot code active.
If rewriting of the new boot code fails for any reason, e.g. power fail or unintended reset, the old boot code still remains active and rewriting can be started again.

**Boot cluster**

The boot code size itself may be smaller than the boot swap cluster size.

The number of flash memory blocks, which are part of the boot code, are named boot cluster. The number of boot blocks, which are member of the cluster, can be defined during Self-Programming via the Self-Programming library.

The boot cluster size determines the boot swap cluster size. This is automatically evaluated from the number of boot blocks, defined during Self-Programming.

*Table 6-10 “Relation between boot block and boot swap cluster” on page 249* shows the relation between the number of boot blocks, the boot cluster size and the boot swap cluster.

**Number of boot blocks BTBLS**

The number of boot blocks (BTBLS) has to be defined by the user during self-programming. It determines the blocks, which are subject to the boot cluster protection, that allows to protect the boot blocks from any erase or write process.

**Boot block protection**

To prohibit rewriting of the boot blocks, the boot cluster protection flag can be set during flash memory programming. When this flag is set, the blocks of the active boot cluster can neither be erased nor written. Boot cluster swapping is impossible as well.
Note that only the blocks of the active boot cluster are protected. In the example according to *Figure 6-12 “Boot cluster swapping function” on page 250* , for instance, blocks 0 and 1 would be prohibited, while blocks 2 and 3 could still be erased and written.

**Caution**

Once the boot cluster protection has been activated, it can never be deactivated again.

For further information concerning flash memory protection flags refer to the chapter *“Code Protection and Security”*.

**Table 6-10 Relation between boot block and boot swap cluster**

| Number of boot blocks BTBLS | Boot cluster size | Boot swap | Boot cluster protection | |
|---|---|---|---|---|
| | | Active boot swap cluster ↔inactive boot swap cluster | | |
| $00_H$ to $03_H$ | Setting inhibited | | | |
| $04_H$ to $07_H$ | 32 KB | $0000\ 0000_H$ to $0000\ 7FFF_H$ ↔$0000\ 8000_H$ to $0000\ FFFF_H$ | Size: | 32 KB |
| | | | Address: | $0000\ 0000_H$ to $0000\ 7FFF_H$ |
| $08_H$ to $0F_H$ | 64 KB | $0000\ 0000_H$ to $0000\ FFFF_H$ ↔$0001\ 0000_H$ to $0001\ FFFF_H$ | Size: | 64 KB |
| | | | Address: | $0000\ 0000_H$ to $0000\ FFFF_H$ |
| $10_H$ to $17_H$ | 128 KB | $0000\ 0000_H$ to $0001\ FFFF_H$ ↔$0002\ 0000_H$ to $0003\ FFFF_H$ | Size: | 96 KB |
| | | | Address: | $0000\ 0000_H$ to $0001\ 7FFF_H$ |
| $18_H$ to $1F_H$ | 128 KB | $0000\ 0000_H$ to $0001\ FFFF_H$ ↔$0002\ 0000_H$ to $0003\ FFFF_H$ | Size: | 128 KB |
| | | | Address: | $0000\ 0000_H$ to $0001\ FFFF_H$ |
| $20_H$ to $3F_H$ | 256 KB | $0000\ 0000_H$ to $0003\ FFFF_H$ ↔$0004\ 0000_H$ to $0007\ FFFF_H$ | Size: | 128 KB + (BTBLS - $1F_H$) * 32 KB |
| | | | Address: | $0000\ 0000_H$ to (128 KB + (BTBLS - $1F_H$) * 32 KB) |
| $40_H$ to $FF_H$ | Setting inhibited | | | |

**Maximum boot swap cluster** The maximum boot cluster size is 256 KB. Thus code flash above 512 KB can not be subject to boot cluster swapping.

*Figure 6-12 “Boot cluster swapping function” on page 250* illustrates an example with following settings:

- number of boot blocks: 2 (boot cluster contains 2 blocks), thus the active boot cluster comprises
  - if boot_flag: blocks 0 and 1
  - if not(boot_flag): blocks 4 and 5
- active boot swap clusters comprises
  - if boot_flag: blocks 0 to 3
  - if not(boot_flag): blocks 4 to 7

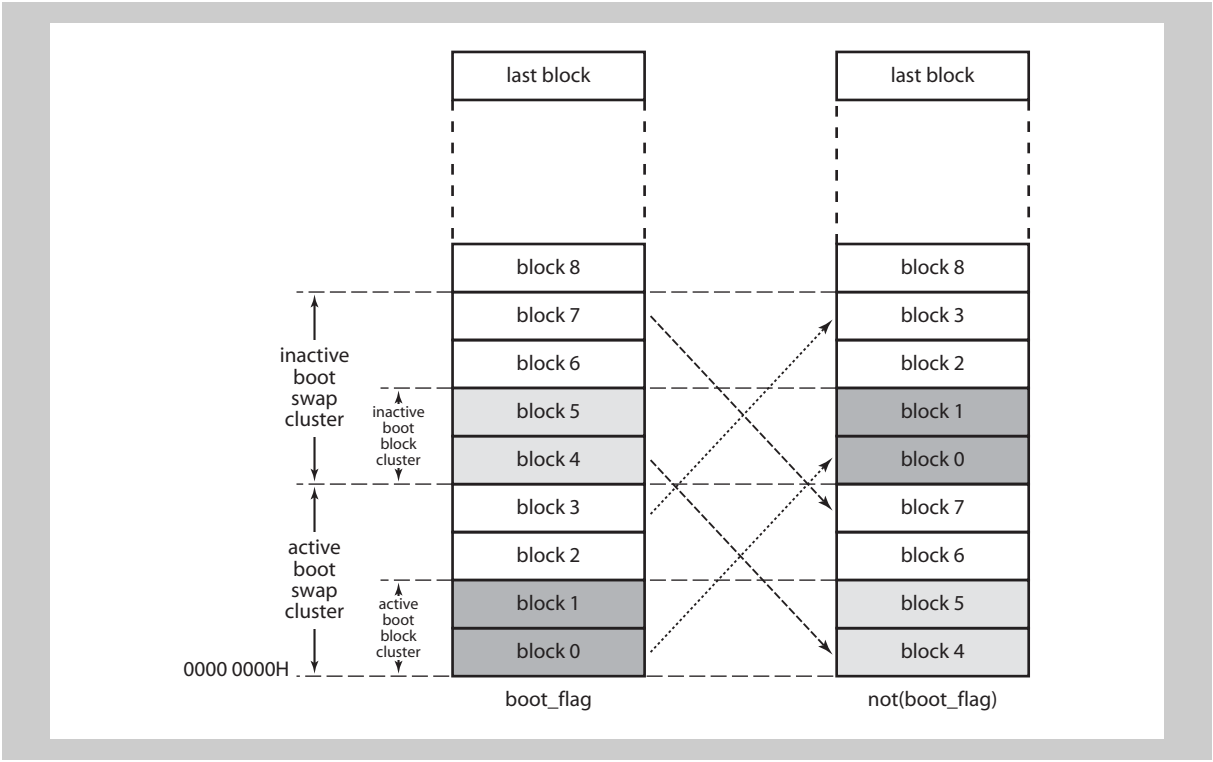



**Figure 6-12 Boot cluster swapping function**

### 6.5.5 Interrupt handling during flash Self-Programming

This microcontroller provides functions to maintain interrupt servicing during the Self-Programming procedure.

Since neither the interrupt vector table nor the interrupt handler routines, which are normally located in the flash memory, are accessible while Self-Programming is active, interrupt acknowledges have to be re-routed to non-flash memory, i.e. to the internal RAM.

Therefore two prerequisites are necessary to enable interrupt servicing during Self-Programming:

- The concerned interrupt handler routine needs to be copied to non-flash memory, e.g. to the internal RAM. The user has to initiate this copy process.
- The concerned interrupt acknowledge has to be re-routed to that handler. Re-routing to the handler is done by use of the CPU registers SW_CFG/ SW_BASE respectively EH_CFG/EH_BASE. Refer to the “Microcontroller Core Architecture User’s Manual” for further details about these CPU registers.

Re-routing of the interrupt vectors offer two options:

- All interrupt can be mapped to the single interrupt vector of interrupt channel 0.
- The base address of the interrupt vector table can be mapped to a different address. In this case the offsets of the interrupt channels are added to the new base address and the correct interrupt vector is obtained upon interrupt acknowledgement.

It is recommended to refer to the User’s Manual “<tbd>”.

## 6.6 Flash configuration options

The flash memory contains a configuration area that holds user specified data for various purposes.

These flash configuration options become effective upon a release of an external RESET, a Power-On-Clear reset POCRES or debugger reset DBRES, thus determining initial settings of various modules.

**Caution** If the flash memory is programmed during a debug session with the on-chip debugger and any flash configuration options have been changed, a target reset command has to be issued in order to make the new option settings effective.

All flash configuration options can be read in all operation modes.

Modification of flash configuration options depend partly on the operation mode, some can not be modified at all.
The following table summarizes all flash configuration options and about the possibility to modify them in the different operation modes.

**Table 6-11 Flash configuration options and setting**

| Function | Flash configuration option | Modification possible in mode | | | |
|---|---|---|---|---|---|
| | | Normal | Flash Serial-Programming | Flash Self-Programming | Debug |
| JTAG port group JP0 control | OPBT0.OPBT0[31] | no | yes | yes | yes |
| VAC enable/ disable of WDTA1 | OPBT0.OPBT0[26] | no | yes | yes | yes |
| Automatic or S/W start of WDTA1 | OPBT0.OPBT0[24] | no | yes | yes | yes |
| Enable/disable of WDTA1 | OPBT0.OPBT0[23] | no | yes | yes | yes |
| VAC enable/ disable of WDTA0 | OPBT0.OPBT0[22] | no | yes | yes | yes |
| Automatic or S/W start of WDTA0 | OPBT0.OPBT0[20] | no | yes | yes | yes |
| Enable/disable of WDTA0 | OPBT0.OPBT0[19] | no | yes | yes | yes |
| Initial value of WDTAn count clock | OPBT0.OPBT0[18:16] | no | yes | yes | yes |

### 6.6.1 OPBT0 - Flash configuration option register 0

**Access** In normal operation mode this register can be read in 32-bit units. Writing to this register is only possible in flash programming and self-programming mode.

**Address** FF47 000C$_H$

**Initial Value** User defined

| 31 | 30 | ... | 0 |
|---|---|---|---|
| OPBT0 [31] | OPBT0 [30] | ... | OPBT0 [1] | OPBT0 [0] |
| R | R | ... | R | R |

**Table 6-12 OPBT0 register contents**

| Bit position | Bit name | Connected to | | Function |
|---|---|---|---|---|
| | | Module | Signal | |
| 31 | OPBT0[31] | JTAG port group JP0 | OPJTAG | Controls the functions of the JTAG port group JP0:<br>0: JP0 used for general purpose/alternative functions ports<br>1: JP0 used as JTAG ports |
| 30 to 27 | OPBT0[30:27] | – | – | Reserved, set to “0” |
| 26 | OPBT0[26] | WDTA1 | OPWDVAC | Enables/disables the Variable Activation Code function (VAC) of WDTA1<br>0: VAC is disabled<br>1: VAC is enabled |
| 25 | OPBT0[25] | – | – | Reserved, set to “0” |
| 24 | OPBT0[24] | WDTA1 | OPWDRUN | Specifies the start mode of WDTA1:<br>0: Software trigger start mode<br>1: Automatic start mode |
| 23 | OPBT0[23] | | OPWDEN | Enables/disables WDTA1:<br>0: WDTA1 is disabled<br>1: WDTA1 is enabled |
| 22 | OPBT0[22] | WDTA0 | OPWDVAC | Enables/disables the Variable Activation Code function (VAC) of WDTA0<br>0: VAC is disabled<br>1: VAC is enabled |
| 21 | OPBT0[21] | – | – | Reserved, set to “0” |
| 20 | OPBT0[20] | WDTA0 | OPWDRUN | Specifies the start mode of WDTA0:<br>0: Software trigger start mode<br>1: Automatic start mode |
| 19 | OPBT0[19] | | OPWDEN | Enables/disables WDTA0:<br>0: WDTA0 is disabled<br>1: WDTA0 is enabled |
| 18 to 16 | OPBT0[18:16] | WDTA0 WDTA1 | OPWDOVF [2:0] | Specifies the reset value of the count clock WDTA0 and WDTA1 control bits WDTAnMD.WDTAnOVF[2:0]. |
| 15 to 0 | OPBT0[15:0] | – | – | Reserved, set to “0” |

# Chapter 7 Clock Controller

This chapter describes the Clock Controller functions of the V850E2/Fx4-G microcontrollers.

The clock signals, their control registers, etc. follow a defined naming convention, that reflects their membership to a certain power domain and clock domain.

**Power domain index m**

The index m is used to define the power area:

- m = 0: denotes the Isolated-Area-0 (Iso0)
- m = A: denotes the Always-On-Area (AWO)

**Clock domain index n**

The index n is used to define the clock domain.

- $n_0$ denotes the clock domain indices of Isolated-Area-0 clock domains
- $n_A$ denotes the clock domain indices of Always-On-Area clock domains

| Power domain | Clock domains |
|---|---|
| Isolated-Area-0 | $n_0$ = 00, 05, 06, 07, 11, 12, 16, 28, 29, 32, 33, 34 |
| Always-On-Area | $n_A$ = 02, 03, 05, 07 |

Examples:
The clock selector registers CKCS_0n select the clocks for the Isolated-Area-0, which are named CKSCLK_0n.
The clock signal CKSCLK_A06 is the clock, supplied to the clock domain 06 (n = 06) on Always-On-Area (m = A). This clock is selected via the clock selector register CKSC_A06.

## 7.1 Clock Controller Overview

**Features summary**

The Clock Controller has the following functions:

- three oscillators:
  - Low Speed Internal Oscillator with a nominal frequency of 240 kHz
  - High Speed Internal Oscillator with a nominal frequency of 8 MHz
  - Main Oscillator 4 MHz to 20 MHz
- one PLL circuit: PLL0
- separate clock selector for each clock domain with individual stand-by control
- three Clock Monitors (CLMA0, CLMA2, CLMA3)
  - CLMA0: supervision of MainOsc
  - CLMA2: supervision of High Speed IntOsc
  - CLMA3: supervision of PLL0

**Note** For the specification of the clock generators frequencies, their tolerance and other parameters, refer to the document Data Sheet .

**Clock sources overview** The following table summarizes all clock sources and their typical respectivelly maximum frequencies:

**Table 7-1 V850E2/Fx4-G clock sources overview**

| Source | Input | Frequency |
|---|---|---|
| MainOsc | external crystal | nom. 4, 5, 6, 8, 10, 12, 16, 20 MHz |
| PLL0 | MainOsc | • V850E2/FF4-G: max. 64 MHz<br>• V850E2/FG4-G: max. 64 MHz |
| Low Speed IntOsc | – | nom. 240 kHz |
| High Speed IntOsc | – | nom. 8 MHz |

**Clock Monitors outputs** The Clock Monitors are generating following output signals to indicate abnormal clock frequencies of the supervised clock:

- $\overline{\text{CLMAnRES}}$: the reset ouputs can generate a reset
- INTCLMAn: the interrupt can generate
  - an interrupt, if connected to the Interrupt Controller
  - a wake-up from stand-by mode, if connected to the Stand-by Controller and if enabled as wake-up factor (refer to the chapter *"Stand-by Controller"* for details).

**Table 7-2 CLMAn output signals**

| CLMA | $\overline{\text{CLMAnRES}}$ connected to | INTCLMAn connected to |
|---|---|---|
| CLMA0 | Reset Controller | Interrupt Controller<br>Stand-by Controller |
| CLMA2 | Reset Controller | Stand-by Controller |
| CLMA3 | Reset Controller | Stand-by Controller |

The following figure shows the main components of the Clock Controller.

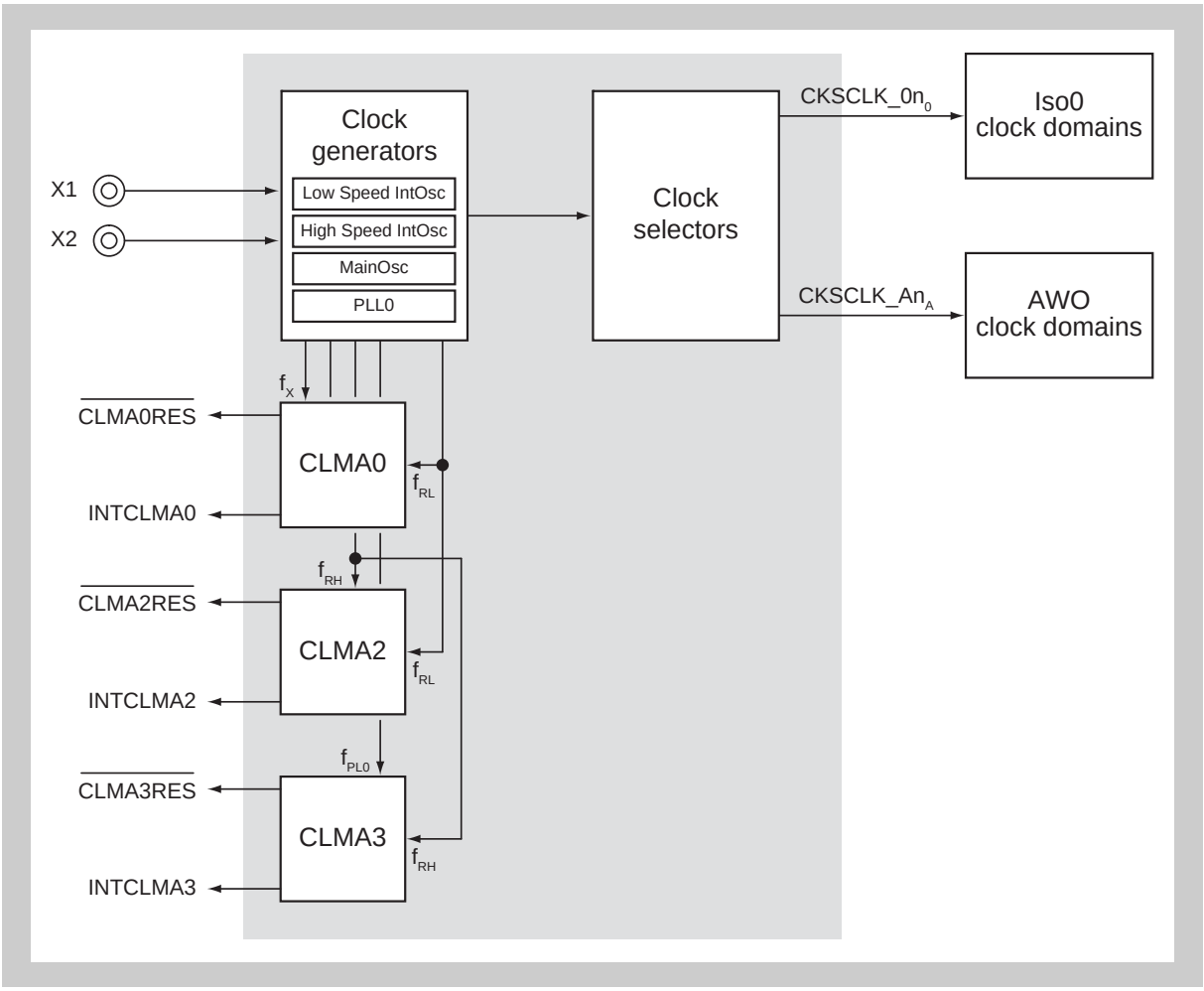



**Figure 7-1 Clock Controller overview**

## 7.2 General Description of Clock Generation and Control

The Clock Controller generates a set of clock signals for each of the three power domains

- CKSCLK_0n for clocks on the Isolated-Area-0 (Iso0)
- CKSCLK_An for clocks on the Always-On-Area (AWO).

Each clock signal CKSCLK_mn supplies the clock for the clock domain “mn”.

Common to all clock domains of a certain power area is, that they can be stopped all together in STOP stand-by mode, that requests the clocks of this domain to be stopped.

The following diagram outlines the basic structure of the Clock Controller.

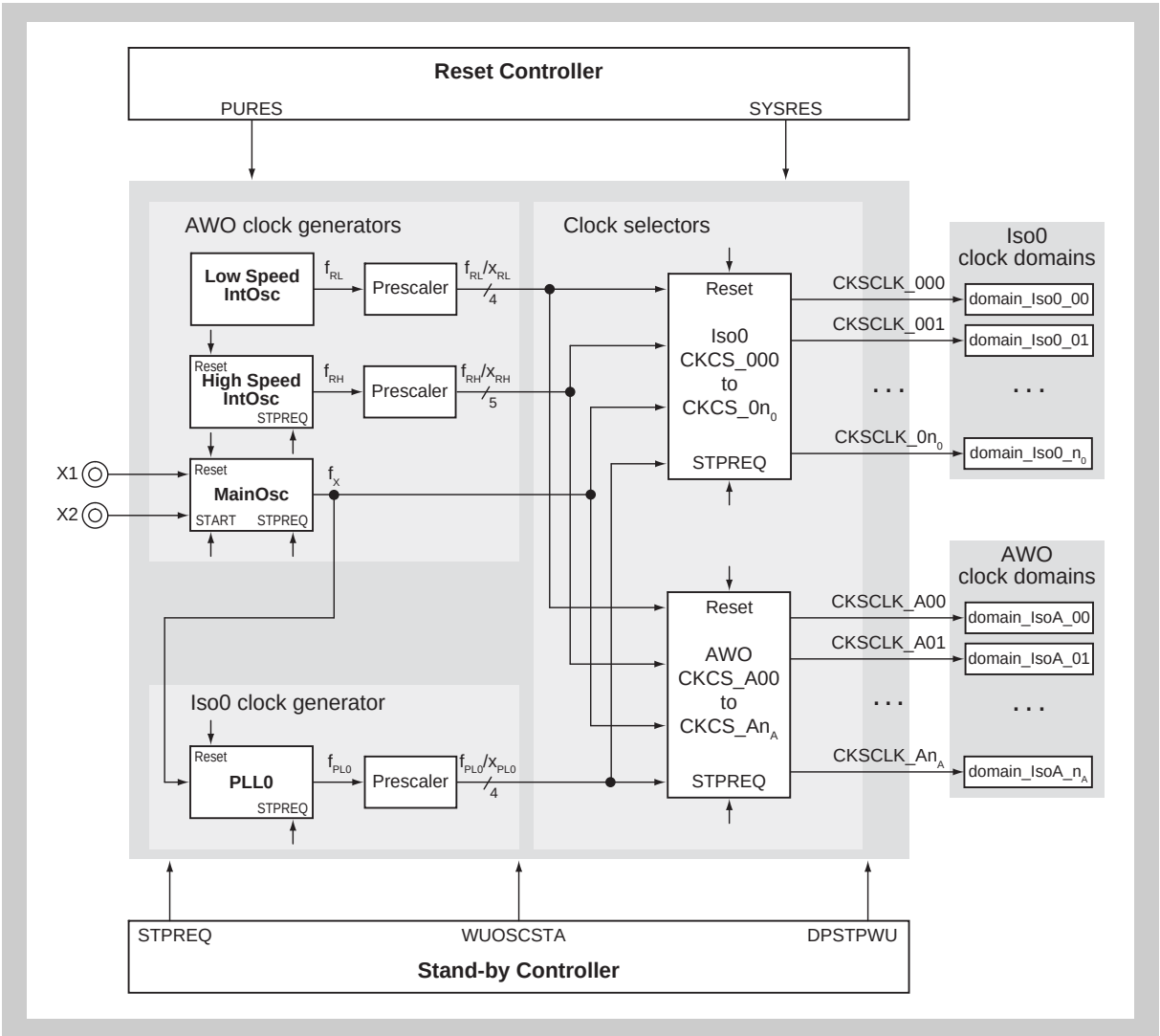



**Figure 7-2 Clock Controller structure**

The Clock Controller provides

- a set of different clock generators, that generate clocks in different frequency ranges
- a set of clock selectors, that allow to select a clock, or fractions thereof, from the clock generators to supply all clock domains on the three power domains.

The clock generators as well as the selectors are controlled by Reset and Stand-by Controller signals:

- Reset Controller signals
  - PURES: power-up reset signal (Power-On-Clear POCRES or Debugger DBRES reset), which is asserted during power-up of the microcontroller
  - SYSRES: system reset signal, which is asserted by any internal and external reset source, including PURES

  For detailed information about the Reset Controller signals, refer to the chapter *"Reset Controller"*.
- Stand-by Controller signals
  - STPREQ: Isolated Area-0 stop request signal
    STPREQ is asserted, when the microcontroller enters an Isolated-Area-0 stand-by mode and is used to stop the clock generators and the Isolated-Area-0 and Always-On-Area domain clocks via the clock selectors $CKSC\_0n_0$ and $CKSC\_An_A$.
  - DPSTPWU: Isolated Area-0 DEEPSTOP wake-up signal
    DPSTPWU is asserted, when the microcontroller wakes up from DEEPSTOP mode and is used to reset the Isolated-Area-0 clock selectors.
  - WUOSCSTA: Wake-up oscillator start signal
    WUOSCSTA can optionally be asserted, when the MainOsc shall start operation upon wake-up from a stand-by mode. The generation of WUOSCSTA is controlled by a register of the Stand-by Controller.

For detailed information about the Stand-by Controller signals, refer to the chapter *"Stand-by Controller (STBC)"*.

Common to all clock generators and clock domains of a certain power domain is, that they can be stopped all together in STOP stand-by mode.
The Stand-by Controller issues the signal STPREQ stop request to the generators and selectors. However all clock generators and most of the domain clock selectors allow to mask the stop request in order to continue clock output to the concerned clock domain also in stand-by mode.

Since the Isolated-Area-0 clock selectors reside on the Isolated-Area-0 power domain, and thus their power supply are switched off in DEEPSTOP mode, the clock selectors are reset upon DEEPSTOP wake-up. So they have to be completely re-initialized afterwards.

The following figure shows the reset and stop request signals wiring of the clock generators and selectors.

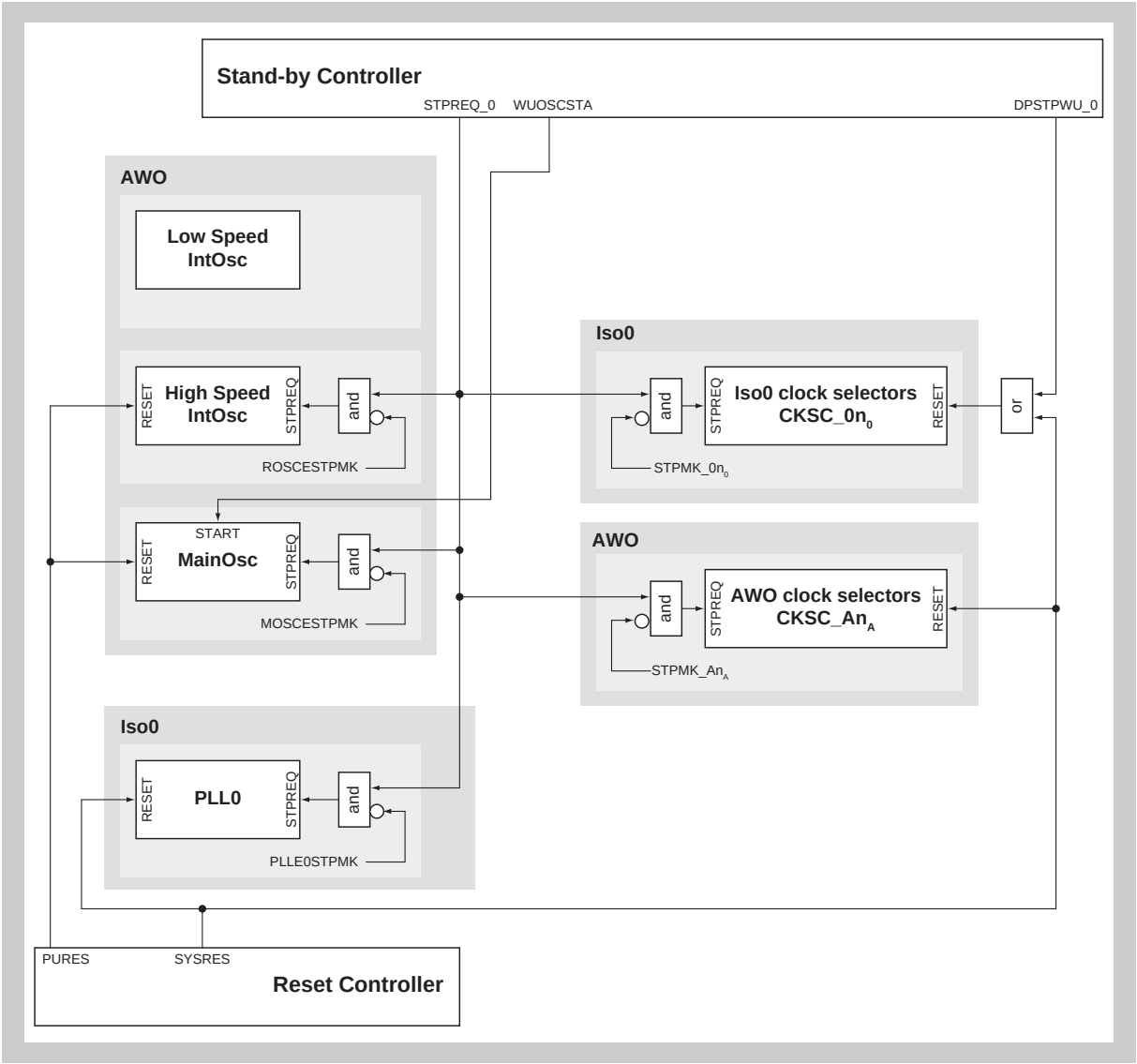



**Figure 7-3 Clock Controller stand-by mode and reset signals**

The Clock Controller consists of two main parts.

### 7.2.1 Clock generators

Four clock generators are provided:

- Low Speed Internal Oscillator (Low Speed IntOsc)
  This oscillator starts operation after power up and can not be stopped, hence it is always operating. It generates the clock $f_{RL}$ with a nominal frequency of 240 KHz.
  It does not require any external components.
- High Speed Internal Oscillator (High Speed IntOsc)
  The High Speed IntOsc generates a clock with the nominal frequency of 8 MHz without any external components.
- Main Oscillator (MainOsc)
  The MainOsc output clock $f_X$ is the main system root clock, as $f_X$ is input to the PLLs. The MainOsc operates with an external resonator (X1, X2).
- PLL0
  The PLL0 circuit generate the high speed operation clock $f_{PL0}$ for normal operation of the microcontroller.

The Low Speed IntOsc, High Speed IntOsc and MainOsc oscillators are located on the Always-On-Area, while the PLL0 resides on the Isolated-Area-0. Thus in DEEPSTOP stand-by mode PLL0 is switched off.

It is optionally possible to individually disable the generators (except the Low Speed IntOsc) in STOP stand-by mode.

The output clocks of the clock generators, except of the MainOsc $f_X$, are input to prescalers, which provide different fractions of these clocks.
Note that the prescalers for all clocks have different division factors.

#### (1) Automatic start of the MainOsc after stand-by wake-up

The MainOsc can be automatically started upon stand-by mode wake-up, independent of their status before stand-by.
Refer to the *"Wake-up"* section in the chapter *"Stand-by Controller (STBC)"* for details about this feature.

#### (2) Clock generators reset

The MainOsc and High Speed IntOsc clock generators are only reset by the power-up reset PURES (Power-On-Clear reset POCRES or Debugger reset DBRES).
Thus any internal reset neither stops these clock generators nor changes their configuration. Consequently the microcontroller start-up time after release from an internal reset is minimum, because the clock generator's stabilization times have not to be regarded.

The PLL0 is reset by all internal and external reset sources via SYSRES.

**(3) STOP stand-by mode request**

The Stand-by Controller signal STPREQ is asserted upon entering Isolated-Area-0 STOP mode and is applied to the stop requests STPREQ of the MainOsc, High Speed IntOsc and PLL0.

STPREQ can be individually masked by the respective clock generator control register:

- MainOsc: MOSCE.MOSCESTPMK = 0:
  STPRQ is not masked, thus $f_X$ is stopped, if STPRQ is asserted
- High Speed IntOsc: ROSCE.ROSCESTPMK = 0:
  STPRQ is not masked, thus $f_{RH}$ is stopped, if STPRQ is asserted
- PLL0: PLLE0.PLLE0STPMK = 0:
  STPRQ is not masked, thus $f_{PL0}$ is stopped, if STPRQ is asserted

If the stop request is not masked, i.e. the clock generator is stopped during stand-by mode, it automatically restarts operation after wake-up from stand-by mode, provided it was operating before stand-by mode entry.
If the clock generator was stopped before stand-by mode, it remains stopped. However the MainOsc can be started after wake-up, even if they were not operating before stand-by. Refer to the section *1 “Automatic start of the MainOsc after stand-by wake-up” on page 260* above.

If the stop request is masked, the clock generator’s status is not changed during stand-by mode.

**(4) DEEPSTOP stand-by mode**

A DEEPSTOP stand-by mode asserts also the STPREQ_0 signal and impacts the clock generators in the same way like STOP stand-by mode.

Since the power supplies of the Isolated-Area-0 is switched off in DEEPSTOP stand-by mode, the PLL0 clock $f_{PL0}$ stops operation.
After release from DEEPSTOP mode the PLL0 control registers take on their reset state and the PLL0 circuit has to be completely re-initialized.

**Notes**

1. Refer to *7.3 “Clock Generators” on page 263* for a detailed description of the clock generators.
2. For the specification of the clock generators frequencies, their tolerance and other parameters, refer to the document Data Sheet.

## 7.2.2 Clock selectors

The clocks, generated by the clock generators, are input to the clock selectors CKSC_mn. A separate clock selector register CKSC_mn is provided for each clock domain.
Note that not all available clocks from the clock generators are input to each clock selector.

The two sets of clock selector registers are dedicated to the clock domains on one of the two power domains:

- $n_0$ registers CKSC_000 to CKSC_0$n_0$ for clock domains on the Isolated-Area-0.

- $n_A$ registers CKSC_A00 to CKSC_An$_A$ for clock domains on the Always-On-Area

By use of a clock selector register CKSC_mn one of its input clocks is selected as the clock CKSCLK_mn.

The clock selectors CKSCLK_mn, selecting the clock domain clocks for power domain m, are also located on the same power domain.

**(1) Clock selectors reset**

The clock selectors are reset by the SYSRES signal, which is asserted by any microcontroller internal or external reset. Hence after any reset the supply of all clock domains are set to their default configuration.

Additionally the clock selectors located on the Isolated-Area-0 power domain are reset if Isolated-Area-0 is woken up from DEEPSTOP mode.

**(2) STOP stand-by mode request**

The stop request for the Always-On-Area and Isolated-Area-0 is issued by the Stand-by Controller by asserting any of the stop request signals STPREQ upon entering STOP mode.

The stop request, supplied to all clock selectors, can be individually masked by the clock selector register CKSC_mn:

- CKSC_mn.STPMK_mn = 0:
  STPREQ is not masked, thus CLSCLK_mn is stopped, if STPREQ is asserted
- CKSC_mn.STPMK_mn = 1:
  STPREQ is masked, thus CLSCLK_mn remains in operation, even if STPREQ is asserted

**Note** Not all clock selectors CKSC_mn provide the stop mask function. These clock domains remain in operation during STOP mode. Refer to the section *"Clock Selection"* below in this chapter for details about each clock selector.

**(3) DEEPSTOP stand-by mode**

In a DEEPSTOP stand-by mode the power supply of the Isolated-Area-0 is switched off.
Since the clock selectors CKSC_0n$_0$ reside on the Isolated-Area-0 power domain, they are switched off as well in DEEPSTOP mode.
Upon wake-up they receive the reset signal DPSTPWU, and are in their default - i.e. reset - state after DEEPSTOP wake-up. Consequently the CKSC_0n$_0$ clock selectors have to be completely re-initialized.

**Note** Refer to *7.6 "Clock Selection" on page 1* for a detailed description of the clock selectors.

# 7.3 Clock Generators

## 7.3.1 Main Oscillator (MainOsc) clock generator

The Main Oscillator generates the clock $f_X$, which is supplied to the clock domain clock selectors CKSC_mn. $f_X$ is also used as the PLL input clock PLLCLKIN.

The diagram below shows the basic structure and signals of the MainOsc clock generator.

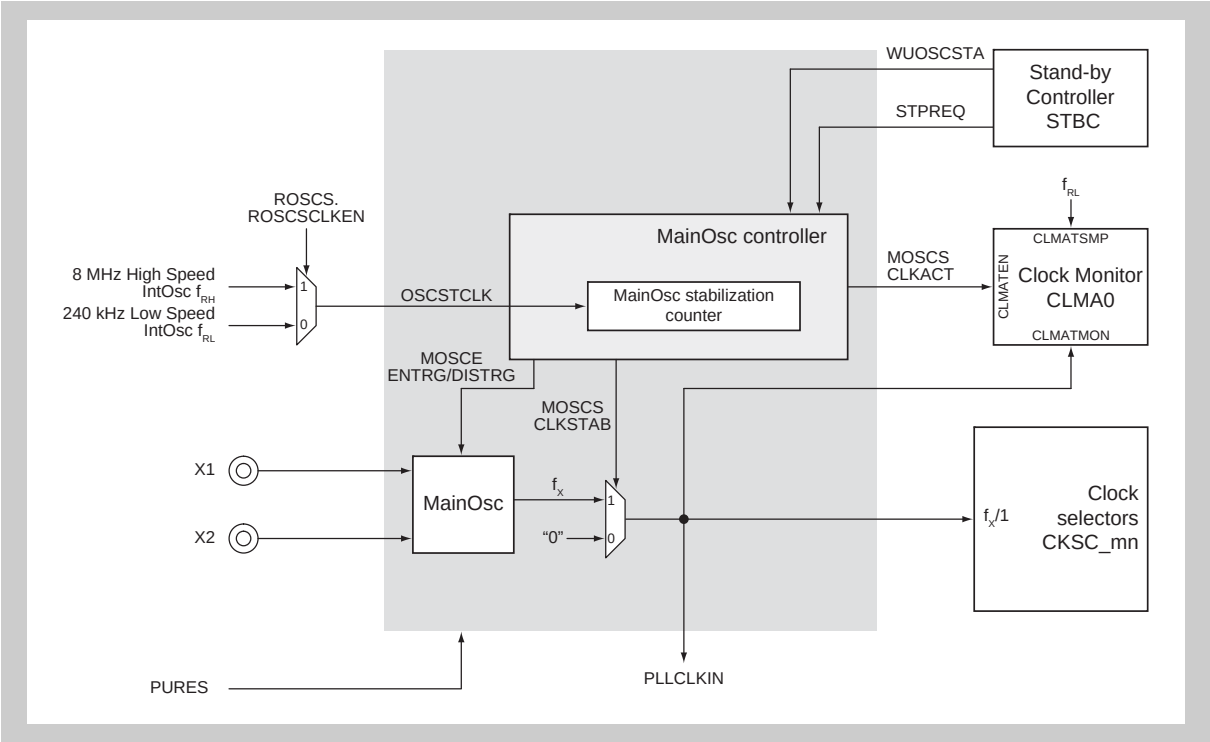



**Figure 7-4 Main Oscillator clock generator**

After release of the power-up reset PURES the MainOsc is disabled.

**Note** During assertion of the external $\overline{\text{RESET}}$ the MainOsc is always operating with its maximum amplification gain.
Refer to the descrition of the external $\overline{\text{RESET}}$ in the *"Reset Controller"* chapter for details.
All other resets except PURES and $\overline{\text{RESET}}$ do not affect the MainOsc.

**MainOsc enable/disable**

The MainOsc can be enabled and disabled and its status can be checked:

- enable trigger: MOSCE.MOSCEENTRG = 1,
  enabled status: MOSCS.MOSCSCLKEN = 1
- disable trigger: MOSCE.MOSCEDISTRG = 1
  disabled status: MOSCS.MOSCSCLKEN = 0

**MainOsc stabilization**

The MainOsc stabilization counter starts counting down the stabilization time. During count down the MOSCSCLKSTAB signal disables the $f_X$ output to the MainOsc prescaler.

If the stabilization counter has reached the stabilization count value, as defined in MOSCST.MOST[3:0], $f_X$ is judged as stable and change of the MOSCSCLKSTAB signal inputs $f_X$ to the prescaler. This status is indicated by the bit MOSCS.MOSCSCLKSTAB = 1.
The active $f_X$ clock at the clock selectors CKSC_mn is indicated by MOSCS.MOSCSCLKACT = 1.

The stabilization counter clock OSCSTCLK is selected from two sources:

- OSCSTCLK = 8 MHz High Speed IntOsc clock $f_{RH}$, if this oscillator is operating (ROSCS.ROSCSCLKEN = 1)
- OSCSTCLK = 240 kHz Low Speed IntOsc clock $f_{RL}$, if the High Speed IntOsc is disabled (ROSCS.ROSCSCLKEN = 0)

The stabilization counter clock source is selected automatically by the High Speed IntOsc operation status.

MOSCST.MOST[3:0] determines the number of OSCSTCLK periods as the MainOsc stabilization time, which can be specified in the range of $2^2$ to $2^{17}$ OSCSTCLK periods.

**Short stabilization time mode**

If the MainOsc has been stopped during stand-by mode (refer to the *"STOP stand-by mode request"* below) and is restarted after wake-up, the stabilization time - and so the wake-up time - can be shortened by selecting the short stabilization time mode (MOSCC.MOSCCSHTSTBY = 1). This sets the amplification gain of the MainOsc circuit to maximum.
Note that selection of the amplication gain by MOSCC.MOSCCSHTSTBY has only effect during the MainOsc stabilization. If the MainOsc is stable normal amplification gain is used.

**Note** Make sure to use the $f_X$ clock only if MOSCS.MOSCSCLKACT = 1.

**STOP stand-by mode requests**

By use of the STPREQ signal the Stand-by Controller indicates stand-by status of the Isolated-Area-0 and the Always-On-Area. The stop request mask bit MOSCE.MOSCESTPMK controls whether the MainOsc is stopped during stand-by or continues operation.
If the MainOsc is stopped during stand-by (MOSCE.MOSCESTPMK = 0), it is automatically re-started upon wake-up from stand-by mode, provided it was operating before stand-by mode entry.
If the MainOsc was stopped before stand-by mode, it remains stopped.

**Note** The MainOsc can also be activated after stand-by wake-up, independent of its status before stand-by mode, via the wake-up oscillator start signal WUOSCSTA. Generation of WUOSCSTA is controlled by the oscillator wake-up mask register OSCWUFMSK. Refer to section *"Wake-up"* in the chapter *"Stand-by Controller (STBC)"* for details.

**Clock Monitor control**

The MainOsc activity flag MOSCS.MOSCSCLKACT is output to the Clock Monitor CLMA0 to control its operation. In case the MainOsc is inactive, supervision of its output clock $f_X$ by CLMA0 is also deactivated.

The table below summarizes the different conditions for the Clock Monitor control.

**Table 7-3 Clock Monitor 0 status control**

| MainOsc enable status | MainOsc stand-by control | | MainOsc stand-by status | CLMA0 status |
|---|---|---|---|---|
| MOSCSCLKEN | MOSCESTPMK | STPREQ | MOSCSCLKACT | |
| 0 | X | X | 0 | stopped |
| 1 | 0 | 0 | 1 | active |
| | | 1 | 0 | stopped |
| | 1 | X | 1 | active |

**MainOsc enable/ disable trigger**

The MainOsc can be enabled and disabled by the enable and disable trigger control bits:

- enable trigger MOSCE.MOSCEENTRG = 1 starts the MainOsc
  Note that setting the enable trigger is only effective if the MainOsc is inactive, i.e. if MOSCS.MOSCSCLKACT = 0.
- disable trigger MOSCE.MOSCEDISTRG = 1 stops the MainOsc
  Note that setting the disable trigger is only effective if the MainOsc is active, i.e. if MOSCS.MOSCSCLKACT = 1.

### 7.3.2 Low Speed Internal Oscillator (Low Speed IntOsc) clock generator

The Low Speed Internal Oscillator generates the clock $f_{RL}$, which is supplied to the clock domain clock selectors CKSC_mn. $f_{RL}$ has a nominal frequency of 240 KHz.

The diagram below shows the basic structure and signals of the Low Speed IntOsc clock generator.

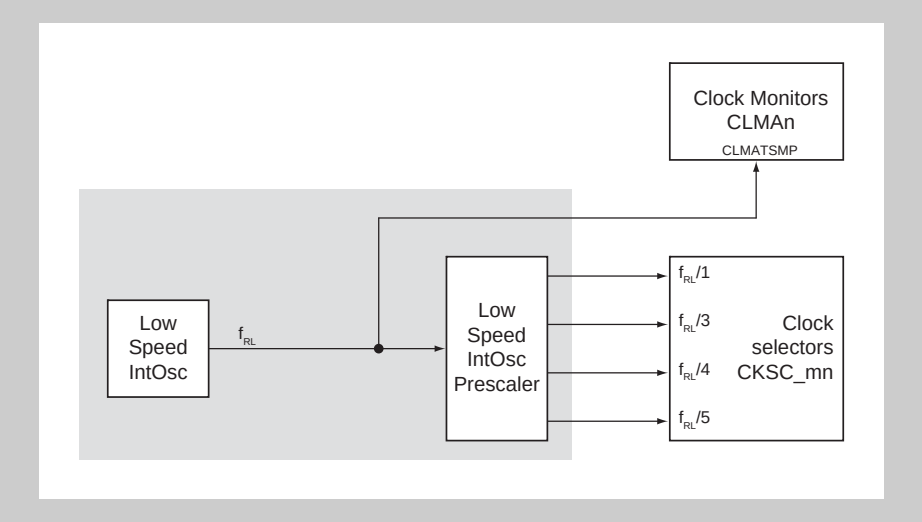



**Figure 7-5 Low Speed Internal Oscillator clock generator**

After reset release the Low Speed IntOsc starts operation. It can not be stopped.

The Low Speed IntOsc clock $f_{RL}$ is used as the sampling clock for Clock Monitors CMLA0 and CLMA2.

### 7.3.3 High Speed Internal Oscillator (High Speed IntOsc) clock generator

The High Speed Internal Oscillator generates the clock $f_{RH}$, which is supplied to the clock domain clock selectors CKSC_mn. $f_{RH}$ has a nominal frequency of 8 MHz.

The diagram below shows the basic structure and signals of the High Speed IntOsc clock generator.

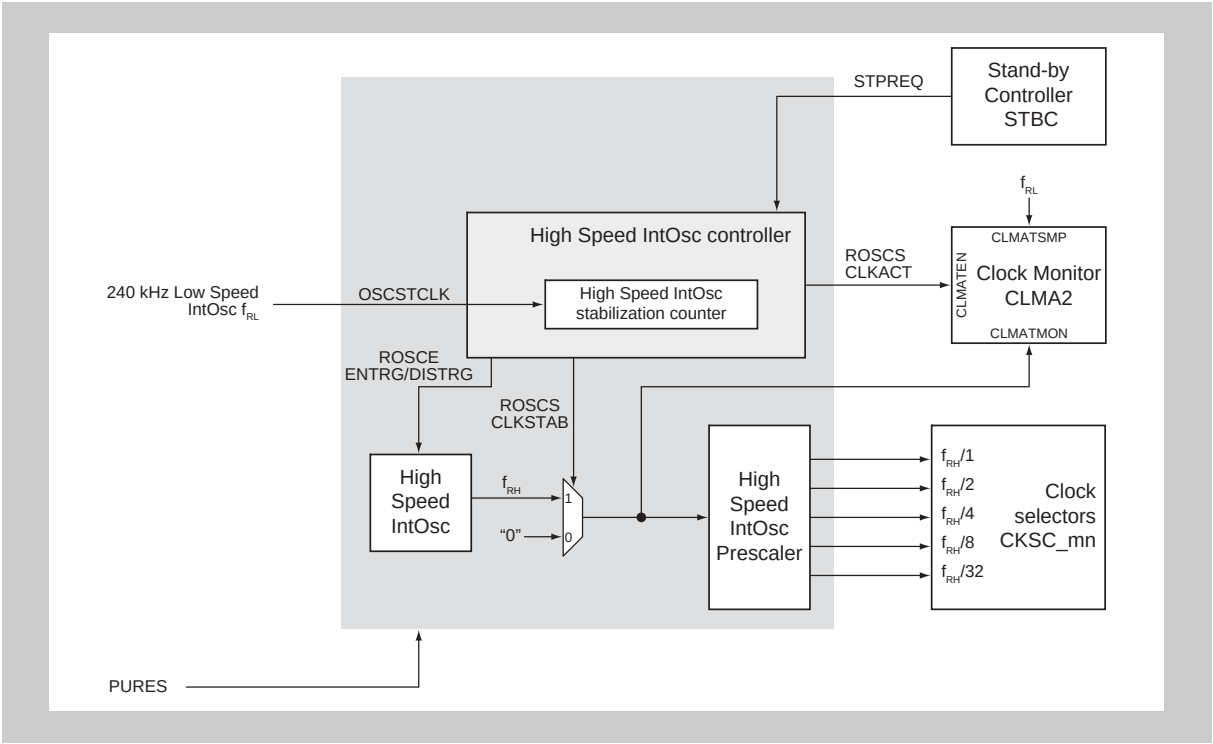



**Figure 7-6 High Speed Internal Oscillator clock generator**

After PURES release the High Speed IntOsc starts operation.

**Note** All other resets except PURES do not affect the High Speed IntOsc.

**High Speed IntOsc enable/disable**

The High Speed IntOsc can be enabled and disabled and its status can be checked:

- enable trigger: ROSCE.ROSCEENTRG = 1,
  enabled status: ROSCS.ROSCSCLKEN = 1
- disable trigger: ROSCE.ROSCEDISTRG = 1
  disabled status: ROSCS.ROSCSCLKEN = 0

**Caution** Pay attention to clock domains, which are using the High Speed IntOsc clock as source via the clock selectors CKSC_mn, when disabling $f_{RH}$ by ROSCE.ROSCEDISTRG = 1.
In particular all resets set all clock selectors CKSCLK_mn to their default selection, but only the PURES enables the High Speed IntOsc.
As a consequence all PBUS clocks PCLK will not operate after a reset - except PURES -, since the High Speed IntOsc is selected as their clock source, but the High Speed IntOsc is not enabled.

**High Speed IntOsc stabilization**

The High Speed IntOsc stabilization counter starts counting down the stabilization time.

As long as the High Speed IntOsc is not stable, the CLKSEL signal disables the $f_{RH}$ output to the MainOsc prescaler.
If the High Speed IntOsc stabilization counter has counted four periods of its input clock OSCSTCLK, $f_{RH}$ is judged as stable and change of CLKSEL inputs $f_{RH}$ to the prescaler, thus all prescaler outputs are available for the clock selectors CKSCmn.
The $f_{RH}$ clock stable status is indicated by the bit ROSCS.ROSCSCLKSTAB = 1 and its activation by ROSCS.ROSCSCLKACT = 1.

The stabilization counter clock OSCSTCLK is the 240 kHz Low Speed IntOsc clock $f_{RL}$.

**Note** Make sure to use the $f_{RH}$ clock only if ROSCS.ROSCSCLKACT = 1.

**Stand-by mode requests**

By use of the STPREQ signal the Stand-by Controller indicates stand-by status of the Isolated-Area-0 and the Always-On-Area. The stop request mask bit ROSCE.ROSCESTPMK controls whether the High Speed IntOsc is stopped during stand-by or continues operation.
If the High Speed IntOsc is stopped during stand-by (ROSCE.ROSCESTPMK = 0), it is automatically re-started upon wake-up from stand-by mode, provided it was operating before stand-by mode entry.
If the High Speed IntOsc was stopped before stand-by mode, it remains stopped.

**Clock Monitor control**

The High Speed IntOsc activity flag ROSCS.ROSCSCLKACT is output to the Clock Monitor CLMA2 to control its operation. In case the High Speed IntOsc is inactive, supervision of its output clock $f_{RH}$ by CLMA2 is also deactivated.

The table below summarizes the different conditions for the Clock Monitor control.

**Table 7-4 Clock Monitor 2 status control**

| High Speed IntOsc enable status | High Speed IntOsc stand-by control | | High Speed IntOsc stand-by status | CLMA2 status |
|---|---|---|---|---|
| ROSCSCLKEN | ROSCESTPMK | STPREQ | ROSCSCLKACT | |
| 0 | X | X | 0 | stopped |
| 1 | 0 | 0 | 1 | active |
| | | 1 | 0 | stopped |
| | 1 | X | 1 | active |

The High Speed IntOsc clock $f_{RH}$ is used as the sampling clock for Clock Monitor CMLA3.

**High Speed IntOsc enable/disable trigger**

The High Speed IntOsc can be enabled and disabled by the enable and disable trigger control bits:

- enable trigger ROSCE.ROSCEENTRG = 1 starts the High Speed IntOsc Note that setting the enable trigger is only effective if the High Speed IntOsc is inactive, i.e. if ROSCS.ROSCSCLKACT = 0.
- disable trigger ROSCE.ROSCEDISTRG = 1 stops the High Speed IntOsc Note that setting the disable trigger is only effective if the High Speed IntOsc is active, i.e. if ROSCS.ROSCSCLKACT = 1.

### 7.3.4 Phase-Locked Loop (PLL) clock generators

The Main Oscillator clock $f_X$ is input to the Phase-Locked Loops clock generator PLL0. The PLL0 output clock $f_{PL0}$ is a multiple of $f_X$ and serve as the main operation clocks for the microcontroller.

The diagram below shows the basic structure and signals of the PLL0 clock generator.

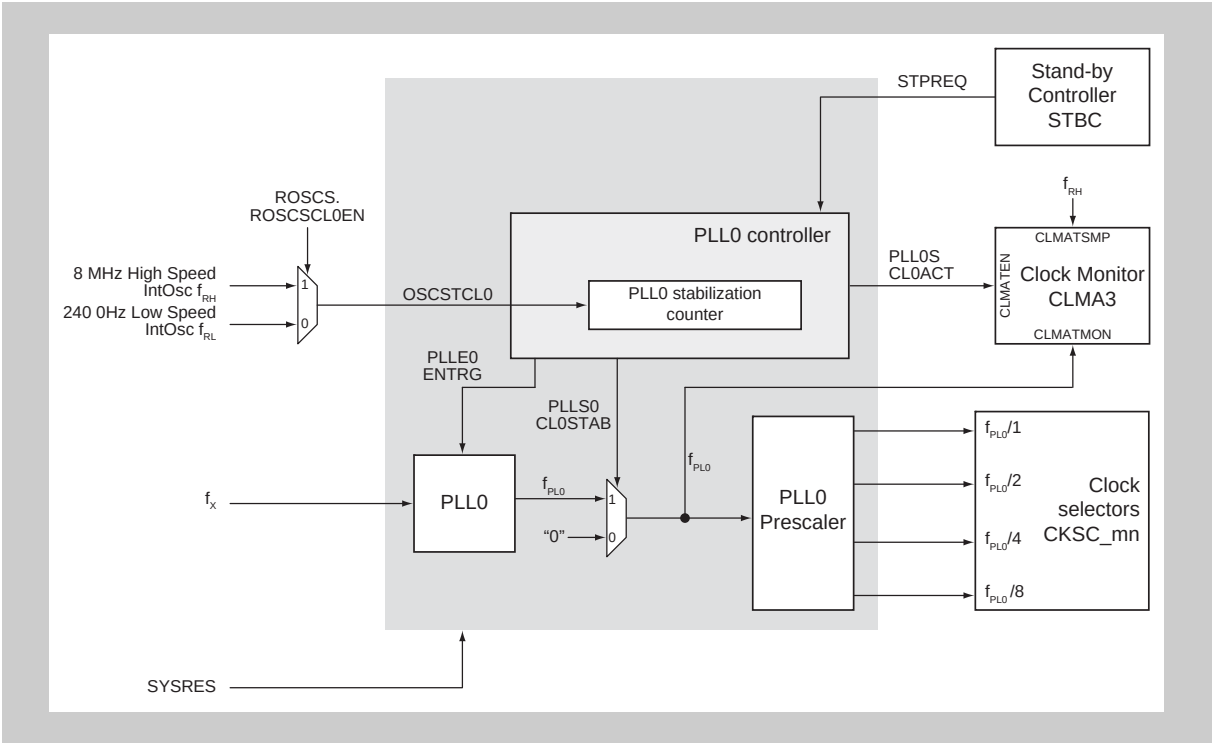



**Figure 7-7 PLL0 clock generator**

After power-up of the microcontroller the PLL0 are disabled.

**Note** All resets (SYSRES) stop the PLL0.

**PLL0 enable/disable**

The PLL0 can be enabled and disabled and its status can be checked:

- enable trigger: PLLE0.PLLE0ENTRG = 1,
  enabled status: PLLS0.PLLS0CLKEN = 1 = 1
- disable trigger: PLLE0.PLLE0DISTRG = 1
  disabled status: PLLS0.PLLS0CLKEN = 1 = 0

**PLL0 stabilization**

The PLL0 stabilization counter starts counting down the stabilization time.

As long as the PLL0 is not stable, the CLKSEL signal disables the $f_{PL0}$ output to the PLL0 prescaler.
If the PLL0 stabilization counter has reached the value, defined by PLLST0.PLLST0[2:0], $f_{PL0}$ is judged as stable and change of CLKSEL inputs $f_X$ to the prescaler, thus all prescaler outputs are available for the clock selectors CKSC_mn.
The $f_{PL0}$ clock stable status is indicated by the bit PLLS0.PLLS0CLKSTABk = 1 and its activation by PLLS0.PLLS0CLKACT = 1.

The stabilization counter clock OSCSTCLK is selected from two sources:

- OSCSTCLK = 8 MHz High Speed IntOsc clock $f_{RH}$, if this oscillator is operating (ROSCS.ROSCSCLKEN = 1)
- OSCSTCLK = 240 kHz Low Speed IntOsc clock $f_{RL}$, if the High Speed IntOsc is disabled (ROSCS.ROSCSCLKEN = 0)

The stabilization counter clock source is selected automatically by the High Speed IntOsc operation status.

PLLST0.PLLST0k[2:0] determines the number of OSCSTCLK periods as the PLL0 stabilization time, which can be specified in the range of $2^7$ to $2^{14}$ OSCSTCLK periods.

**Note** Make sure to use the $f_{PLk0}$ clock only if PLLS0.PLLS0CLKACT = 1.

**Stand-by mode requests**

By use of the STPREQ signal the Stand-by Controller indicates stand-by status of the Isolated-Area-0 and the Always-On-Area. The stop request mask bit PLLE0.PLLE0STPMK controls whether the PLL0 is stopped during stand-by or continues operation.
If the PLL0 is stopped during stand-by (PLLE0.PLLE0STPMK = 0), it is automatically re-started upon wake-up from stand-by mode, provided it was operating before stand-by mode entry.
If the PLL0 was stopped before stand-by mode, it remains stopped.

**Clock Monitor control**

The PLL0 activity signal CLKACT is output to the Clock Monitor CLMA3 to control its operation. In case the PLL0 is inactive, supervision of its output clock $f_{PL0}$ by CLMA3 is also deactivated.
The table below summarizes the different conditions for the Clock Monitor control.

**Table 7-5 Clock Monitor 3 status control**

| PLL0 enable status | PLL0 stand-by control | | PLL0 stand-by status | CLMA3 status |
|---|---|---|---|---|
| PLLS0CLKEN | PLLE0STPMK | STPREQ | PLLS0CLKACT | |
| 0 | X | X | 0 | stopped |
| 1 | 0 | 0 | 1 | active |
| | | 1 | 0 | stopped |
| | 1 | X | 1 | active |

**PLL0 enable/disable trigger**

The PLL0 can be enabled and disabled by the enable and disable trigger control bits:

- enable trigger PLLE0.PLLE0ENTRG = 1 starts the PLL0
  Note that setting the enable trigger is only effective if the PLL0 is inactive, i.e. if PLLS0.PLLS0CLKACT = 0.
- disable trigger PLLE0.PLLE0DISTRG = 1 stops the PLL0
  Note that setting the disable trigger is only effective if the PLL0 is active, i.e. if PLLS0.PLLS0CLKACT = 1.

### (1) PLL parameters

The PLL is configured by a set of parameters, derived from the control register PLLC0.

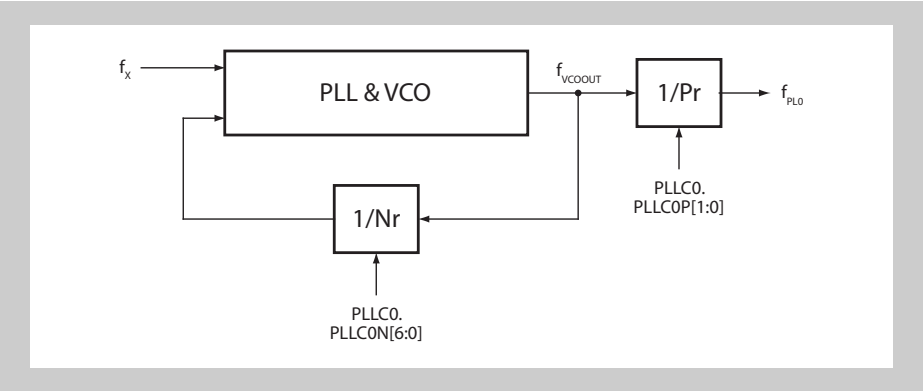



**Figure 7-8 PLL0 circuit**

**$f_{PL0}$** The frequency $f_{PL0}$ is calculated as follows:

$$f_{PL0} = f_X \bullet \frac{Nr}{Pr}$$

The values Nr, Mr and Pr are derived from PLLC0 register bits:

- Nr = PLLC0.PLLC0N[4:0] + 1
- Pr is determined by PLLC0.PLLC0P[1:0] according to the following table:

| PLLC0.PLLC0P[1:0] | Pr | PLL output frequency $f_{PL0}$ range |
|---|---|---|
| $10_B$ | 2 | 40 MHz to 64 MHz |
| $11_B$ | 4 | 20 MHz to 32 MHz |
| $0x_B$ | setting prohibited | |

## 7.4 Clock Selection

This section specifies all clock selection options for all clock domain on the three power areas of the V850E2/Fx4-G products.

The names of clock selector control and status registers are using two indices to identify the power domain and the clock domain:

**(1) Clock selectors indices**

**m = 0: Isolated-Area-0** The clock selectors CKSC_0n control the CKSCLK_0n clocks of any clock domain within Isolated-Area-0.

**m = A: Always-On-Area** The clock selectors CKSC_An controls the clock of any CKSCLK_An clocks domain within the Always-On-Area.

For each clock selector register a separate table is provided that informs about

- the power and the clock domain
- the clock selector register name, its address and initial value
- the domain clock name
- the clock selection options, their ID and - if applicable - clock limitations
- the availability of the clock stop mask bit CKSC_mn.STPMK_mn, that allows to determine the domain clock operation during STOP stand-by mode.

**(2) Clock ID and default clock selection**

**Clock ID** Each input clock to a clock selector CKSC_mn is identified by a unique ID. This ID has to be written to the clock selector register CKSC_mn to select the clock.

**Note** The ID must be written to the bits CKSC_mn.CKSCID_mn[30:0], which are located in the CKSC_mn register bits 31 to 1. Thus to select the ID clock

CKSC_mn = 2 x ID

must be written.

For details refer to the description of the CKSC_mn register in the section *“Clock Controller Registers”* of this chapter.

**Default clock** In the clock selection tables of the following section the default clock selection is emphasized in bold.

**(3) Clock STOP mode**

In STOP stand-by mode the clock output of the clock selector may be optionally stopped by setting the clock selection register bit CKSC_mn.STPMK_mn = 0. If

- STPMK_mn: available

  Clock CKSCLK_mn operation in STOP mode can be selected:

  – if CKSC_mn.STPMK_mn = 0: CKSCLK_mn is stopped in STOP mode

  – if CKSC_mn.STPMK_mn = 1: CKSCLK_mn continues in STOP mode

- STPMK_mn: not available, no STOP

  CKSC_mn.STPMK_mn is not available (STPMK_mn is fixed to 0).
  The clock CKSCLK_mn can not be stopped in STOP mode.

- STPMK_mn: not available, always STOP

  CKSC_mn.STPMK_mn is not available (STPMK_mn is fixed to 0).
  The clock CKSCLK_mn is always stopped in STOP mode.

### (4) Clock switching

**Caution** When changing the clock selection for a clock domain from one clock to another, make sure that both clocks are operating.

**Legal ID** When changing the source of a domain clock CKSCLK_mn by writing to the respective clock selector register CKSC_mn proceed as follows:

1. Write the ID of the new clock source:
   – CKSC_mn.CKSCID_mn = new_ID
2. Wait the time, as specified below under the key word *"Clock switching timing"*, before checking the status of the clock selector ID change in the next step.
3. Check that the domain clock CKSCLK_nm has changed to the new clock source, indicated by
   – CSCSTAT_mn.CLKSELID_mn[30:0] = new_ID
   – CSCSTAT_mn.CLKACT_mn = 1 (new clock active)

**Caution** Make sure that no accesses (by the CPU, DMA, etc.) to modules, supplied by the domain clock to be changed, are performed before activation of the new_ID is confirmed by CSCSTAT_mn.CLKACT_mn = 1.

**Illegal ID** If an illegal clock source ID, i.e. an ID not permitted for a certain clock domain m_n, is written to CKSC_mn.CKSCID_mn, the clock selector behaves as follows:

- clock source ID $0000_H$ (no clock selected) is a *legal* selection
  – illegal clock ID is not accepted, instead clock ID $0000_H$ is selected, thus the domain clock CKSCLK_mn is stopped
  – clock selector control register shows the illegal_ID:
  CKSC_mn.CKSCID = illegal_ID
  – clock selector status register shows ID $0000_H$ as inactive:
  CSCSTAT_mn.CLKSELID_mn[30:0] = $0000_H$
  CSCSTAT_mn.CLKACT_mn = 0

- clock source ID $0000_H$ (no clock selected) is *not a legal* selection
  - illegal clock ID is not accepted and the old clock ID remains valid, thus the domain clock CKSCLK_mn does not change
  - clock selector control register shows the old ID: CKSC_mn.CKSCID = old_ID
  - clock selector status register shows the old ID as active: CSCSTAT_mn.CLKSELID_mn[30:0] = old_ID CSCSTAT_mn.CLKACT_mn = 1

**Clock switching timing**

The time between writing an ID to a clock selector control register CKSC_mn (for mn ≠ 000) until the status can be evaluated via the content of its status register CSCSTAT_mn is calculated as follows:
Conditions: new_ID ≠ $0000_H$, old_ID ≠ $0000_H$, new_ID ≠ old_ID

- if the High Speed IntOsc is active (ROSCS.ROSCSCLKEN = 1): $13 / f_{RH} + 3 / f_{old\_ID} + 3 / f_{new\_ID}$
- if the High Speed IntOsc is inactive (ROSCS.ROSCSCLKEN = 0): $13 / f_{RL} + 3 / f_{old\_ID} + 3 / f_{new\_ID}$

with

| | | |
|---|---|---|
| $f_{RH}$ | = | frequency of the High Speed IntOsc (nom. 8 MHz) |
| $f_{RL}$ | = | frequency of the Low Speed IntOsc (nom. 240 kHz) |
| $f_{old\_ID}$ | = | frequency of the old_ID clock source |
| $f_{new\_ID}$ | = | frequency of the new_ID clock source |

**ID = $0000_H$**

If the old_ID = $0000_H$ or new_ID = $0000_H$, i.e. no clock selected, the respective summand ($3/f_{new\_ID}$) or ($3/f_{new\_ID}$), respectively, becomes 0 ($f \rightarrow \infty$ in above formulas).

**new_ID = old_ID**

If new_ID = old_ID the clock switching time becomes 0.

**Clock switching time for CKSCLK_000 (CPUCLK) domain clock**

If the CPU clock CPUCLK (domain clock CKSCLK_000) is changed, the CPU operation stalls during the change of the clock.
The CPU stall time can be calculated as follows:
Conditions: new_ID ≠ old_ID

- if the High Speed IntOsc is active (ROSCS.ROSCSCLKEN = 1): $4 / f_{RH} + 3 / f_{new\_ID}$
- if the High Speed IntOsc is inactive (ROSCS.ROSCSCLKEN = 0): $4 / f_{RL} + 3 / f_{new\_ID}$

### 7.4.1 Clock domains of Always–On–Area

#### (1) Clock domain AWO_2

| Clock selector control register: CKSC_A02 | | Power domain: | Always–On–Area | |
|---|---|---|---|---|
| Address: | FF42 2020$_H$ | Clock domain: | AWO_2 | |
| Initial value: | 0000 000E$_H$ | STPMK_A02: | available | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| 0001$_H$ | Low Speed IntOsc [240 kHz] / 1 | ≤ 48 MHz | CKSCLK_A02 | WDTA0: PCLK<br>CLMA0: PCLK<br>CLMA2: PCLK<br>KR0: PCLK<br>AWO port control and filters: PCLK[a] |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | | | |
| 0008$_H$ | High Speed IntOsc [8 MHz] / 2 | | | |
| 0009$_H$ | High Speed IntOsc [8 MHz] / 4 | | | |
| 000A$_H$ | High Speed IntOsc [8 MHz] / 8 | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| 0000$_H$ | No clock selected, domain clock is stopped | | | |
| others than above | Setting prohibited | | | |

a) Domain clock CKSCLK_A02 supplies the port control and filter modules of the Always-On-Area, i.e. for port groups P0, JP0.

#### (2) Clock domain AWO_3

| Clock selector control register: CKSC_A03 | | Power domain: | Always–On–Area | |
|---|---|---|---|---|
| Address: | FF42 2030$_H$ | Clock domain: | AWO_3 | |
| Initial value: | 0000 000E$_H$ | STPMK_A03: | available | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| 0001$_H$ | Low Speed IntOsc [240 kHz] / 1 | ≤ 48 MHz | CKSCLK_A03 | TAUJ0: PCLK |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | | | |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| 0000$_H$ | No clock selected, domain clock is stopped | | | |
| others than above | Setting prohibited | | | |

### (3) Clock domain AWO_5

| **Clock selector control register: CKSC_A05** | | **Power domain:** | **Always–On–Area** | |
|---|---|---|---|---|
| **Address:** | **FF42 2050$_H$** | **Clock domain:** | **AWO_5** | |
| **Initial value:** | **0000 000E$_H$** | **STPMK_A05:** | **available** | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 40 MHz | CKSCLK_A05 | BURAM: PCLK |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| others than above | Setting prohibited | | | |

### (4) Clock domain AWO_7

| **Clock selector control register: CKSC_A07** | | **Power domain:** | **Always–On–Area** | |
|---|---|---|---|---|
| **Address:** | **FF42 2070$_H$** | **Clock domain:** | **AWO_7** | |
| **Initial value:** | **0000 0006$_H$** | **STPMK_A07:** | **available** | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| 0001$_H$ | Low Speed IntOsc [240 kHz] / 1 | – | CKSCLK_A07 | WDTA0: WDTACKI |
| **0003$_H$** | **Low Speed IntOsc [240 kHz] / 4** | | | |
| 0005$_H$ | Low Speed IntOsc [240 kHz] / 512 | | | |
| others than above | Setting prohibited | | | |

## 7.4.2 Clock domains of Isolated–Area-0

### (1) Clock domain ISO0_0

| Clock selector control register: CKSC_000 | | Power domain: Isolated-Area-0 | | |
|---|---|---|---|---|
| Address: | $FF42\ 6000_H$ | Clock domain: | ISO0_0 | |
| Initial value: | $0000\ 0074_H$ | STPMK_000: | not available, always STOP | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| $0008_H$ | High Speed IntOsc [8 MHz] / 2 | – | CKSCLK_000 | CPU Subsystem: CPUCLK |
| $0009_H$ | High Speed IntOsc [8 MHz] / 4 | | | |
| $000A_H$ | High Speed IntOsc [8 MHz] / 8 | | | |
| $000B_H$ | High Speed IntOsc [8 MHz] / 32 | | | |
| $000C_H$ | MainOsc | | | |
| $0014_H$ | PLL0 / 1 | | | |
| $0015_H$ | PLL0 / 2 | | | |
| $0017_H$ | PLL0 / 4 | | | |
| $001A_H$ | PLL0 / 8 | | | |
| **$003A_H$** | **High Speed IntOsc [8 MHz] (Low Speed IntOsc [240 kHz])[a]** | | | |
| others than above | Setting prohibited | | | |

a) If the High Speed IntOsc is disabled, the Low Speed IntOsc is automatically selected.

### (2) Clock domain ISO0_1

| Clock selector control register: – | | Power domain: Isolated-Area-0 | | |
|---|---|---|---|---|
| Address: | – | Clock domain: | ISO0_1 | |
| Initial value: | – | STPMK_001: | – | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| – | CKSCLK_000: CPU system clock CPUCLK / 2 | – | CKSCLK_001 | WDTA1: PCLK<br>CLMA3: PCLK<br>Iso0 port control and filters: PCLK[a] |

a) Domain clock CKSCLK_001 supplies the port control and filter modules of the Isolated-Area-0, i.e. for port groups P1, P3, P4, P10, P27.

CKSCLK_001 has no clock selector, but is supplied by the half of the CPU system clock CPUCLK (CPUCLK / 2 = CKSCLK_000 / 2).

### (3) Clock domain ISO0_6

| **Clock selector control register: CKSC_006** | | **Power domain:** | **Isolated-Area-0** | |
|---|---|---|---|---|
| **Address:** | **FF42 6060$_H$** | **Clock domain:** | **ISO0_6** | |
| **Initial value:** | **0000 000E$_H$** | **STPMK_006:** | **available** | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_006 | TAUB0: PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| 0000$_H$ | No clock selected, domain clock is stopped | | | |
| others than above | Setting prohibited | | | |

### (4) Clock domain ISO0_7

| **Clock selector control register: CKSC_007** | | **Power domain:** | **Isolated-Area-0** | |
|---|---|---|---|---|
| **Address:** | **FF42 6070$_H$** | **Clock domain:** | **ISO0_7** | |
| **Initial value:** | **0000 0006$_H$** | **STPMK_007:** | **not available, no STOP** | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| 0001$_H$ | Low Speed IntOsc [240 kHz] / 1 | – | CKSCLK_007 | WDTA1: WDTATCKI |
| **0003$_H$** | **Low Speed IntOsc [240 kHz] / 4** | | | |
| others than above | Setting prohibited | | | |

### (5) Clock domain ISO0_11

| Clock selector control register: CKSC_011 | | Power domain: | Isolated-Area-0 | |
|---|---|---|---|---|
| Address: | FF42 60B0$_H$ | Clock domain: | ISO0_11 | |
| Initial value: | 0000 000E$_H$ | STPMK_011: | available | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_011 | URTE10: PCLK<br>LMA10: PCLK<br>URTE11: PCLK<br>LMA11: PCLK<br>CNTA2: PCLK<br>CSIG4: PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| 0000$_H$ | No clock selected, domain clock is stopped | | | |
| others than above | Setting prohibited | | | |

### (6) Clock domain ISO0_12

| Clock selector control register: CKSC_012 | | Power domain: | Isolated-Area-0 | |
|---|---|---|---|---|
| Address: | FF42 60C0$_H$ | Clock domain: | ISO0_12 | |
| Initial value: | 0000 000E$_H$ | STPMK_012: | available | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_012 | ADAA0: PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| 0000$_H$ | No clock selected, domain clock is stopped | | | |
| others than above | Setting prohibited | | | |

### (7) Clock domain ISO0_16

| Clock selector control register: CKSC_016 | | Power domain: | Isolated-Area-0 | |
|---|---|---|---|---|
| Address: | FF42 6100$_H$ | Clock domain: | ISO0_16 | |
| Initial value: | 0000 000E$_H$ | STPMK_016: | not available, no STOP | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_016 | Iso0 Port filters: DNFATCKI |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| others than above | Setting prohibited | | | |

### (8) Clock domain ISO0_28

| Clock selector control register: CKSC_028 | | Power domain: | Isolated-Area-0 | |
|---|---|---|---|---|
| Address: | FF42 A080$_H$ | Clock domain: | ISO0_28 | |
| Initial value: | 0000 000E$_H$ | STPMK_028: | available | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_028 | CSIG0: PCLK<br>IICB0: PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| others than above | Setting prohibited | | | |

### (9) Clock domain ISO0_29

| **Clock selector control register: CKSC_029** | | **Power domain:** | **Isolated-Area-0** | |
|---|---|---|---|---|
| **Address:** | **FF42 A090$_H$** | **Clock domain:** | **ISO0_29** | |
| **Initial value:** | **0000 000E$_H$** | **STPMK_029:** | **available** | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKS CLK_029 | CSIG7: PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| others than above | Setting prohibited | | | |

### (10) Clock domain ISO0_32

| **Clock selector control register: CKSC_032** | | **Power domain:** | **Isolated-Area-0** | |
|---|---|---|---|---|
| **Address:** | **FF42 A0C0$_H$** | **Clock domain:** | **ISO0_32** | |
| **Initial value:** | **0000 000E$_H$** | **STPMK_032:** | **available** | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_032 | OSTM0: PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| others than above | Setting prohibited | | | |

### (11) Clock domain ISO0_33

| Clock selector control register: CKSC_033 | | Power domain: | Isolated-Area-0 | |
|---|---|---|---|---|
| Address: | FF42 A0D0$_H$ | Clock domain: | ISO0_33 | |
| Initial value: | 0000 000E$_H$ | STPMK_033: | available | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_033 | FCN0: PCLK<br>FCN1 : PCLK<br>FCN2 : PCLK<br>FCN3 : PCLK<br>FCN4 : PCLK<br>FCN5 : PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| others than above | Setting prohibited | | | |

### (12) Clock domain ISO0_34

| Clock selector control register: CKSC_034 | | Power domain: | Isolated-Area-0 | |
|---|---|---|---|---|
| Address: | FF42 A0E0$_H$ | Clock domain: | ISO0_34 | |
| Initial value: | 0000 000E$_H$ | STPMK_034: | available | |
| **Clock source ID** | **Clock source** | **Clock limitation** | **Domain clock** | **Module: clock** |
| **0007$_H$** | **High Speed IntOsc [8 MHz] / 1** | ≤ 48 MHz | CKSCLK_034 | URTE2: PCLK<br>LMA2: PCLK<br>CNTA1: PCLK |
| 000C$_H$ | MainOsc | | | |
| 0014$_H$ | PLL0 / 1 | | | |
| 0015$_H$ | PLL0 / 2 | | | |
| 0017$_H$ | PLL0 / 4 | | | |
| 001A$_H$ | PLL0 / 8 | | | |
| others than above | Setting prohibited | | | |

## 7.5 Clock Domain Figures

The following figures show the PBUS structures with regards to the PBUS modules in the different clock domains.

**Note** CPU (domain clock CKSCLK_000) accesses to modules, located in a different clock domain are passed through bus synchronizers, as shown in the following figures.
These synchronizers induce an access latency of several clock cycles.
Refer to the section *"CPU Access Bus Structures and Latencies"* in the chapter *"CPU System Function"* for details.

### (1) Clock domains AWO_2, AWO_3, AWO_5

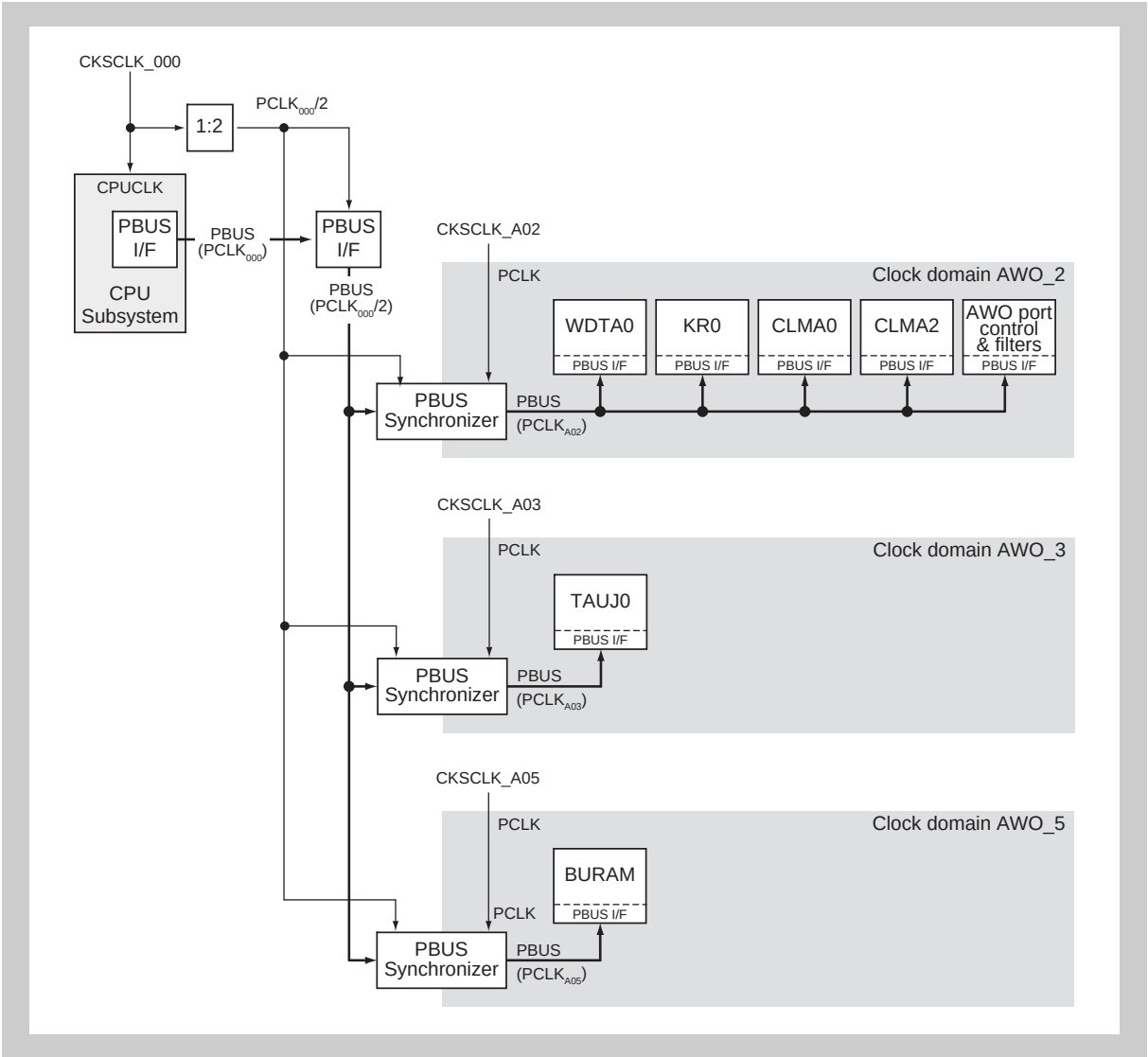



**Figure 7-9 Clock domains AWO_2, AWO_3, AWO_5**

### (2) Clock domains AWO_7

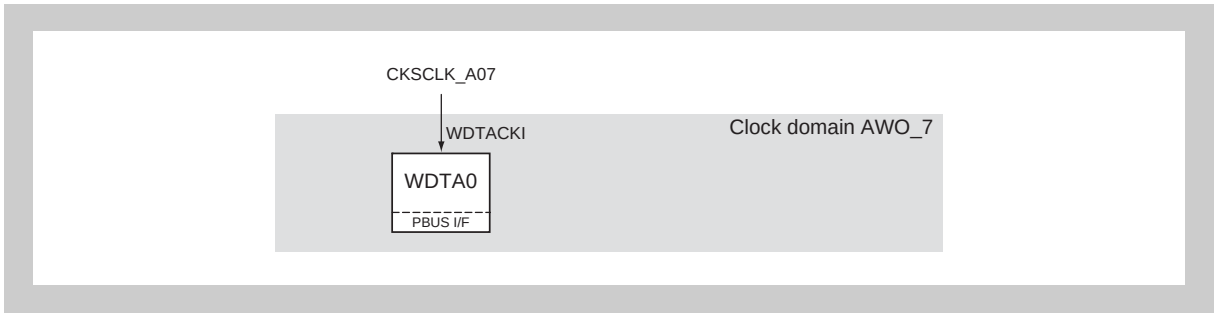



**Figure 7-10 Clock domains AWO_7**

### (3) Clock domains ISO0_1, ISO0_6, ISO0_11, ISO0_12

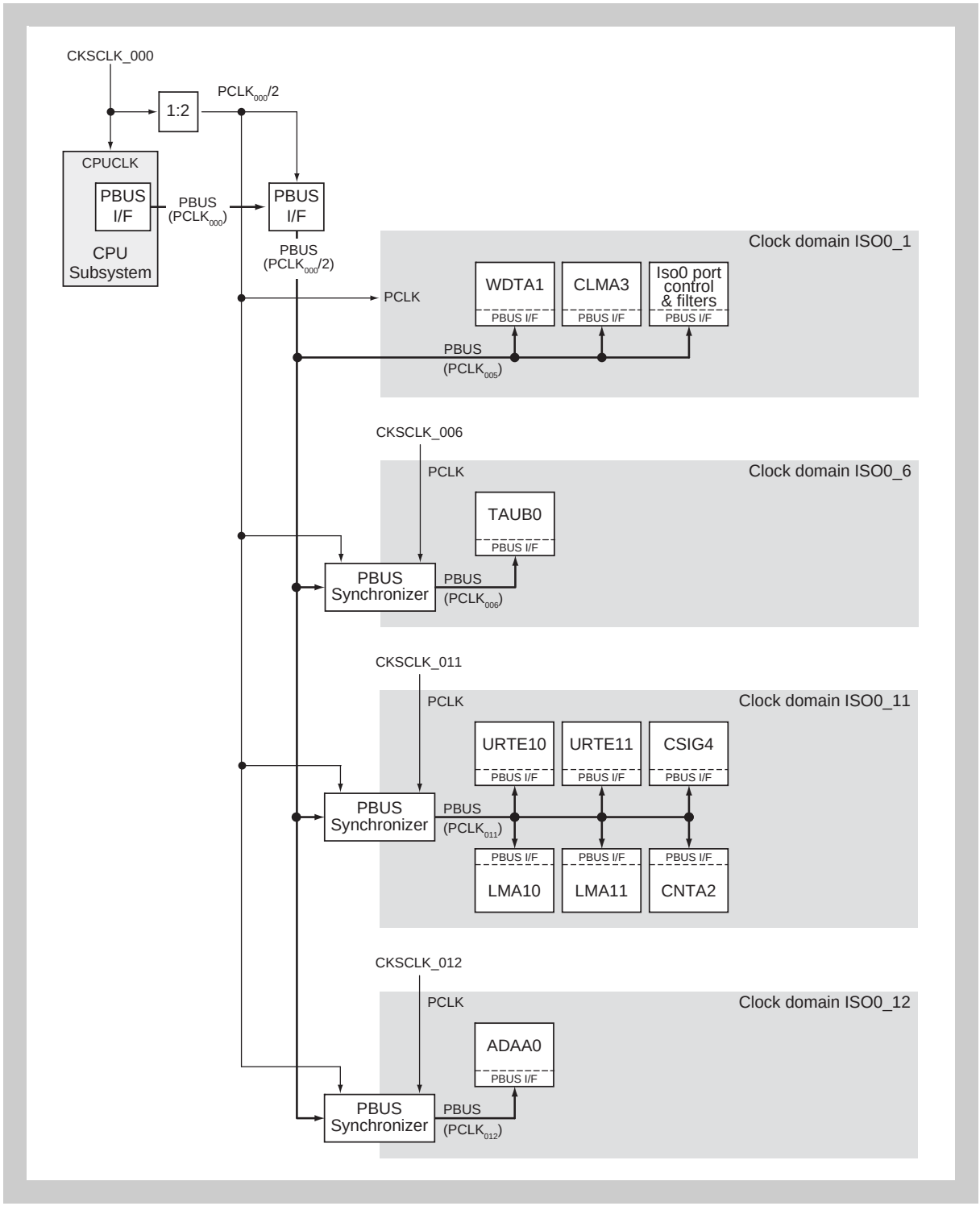



**Figure 7-11 Clock domains ISO0_1, ISO0_6, ISO0_11, ISO0_12**

### (4) Clock domains ISO0_7, ISO0_16

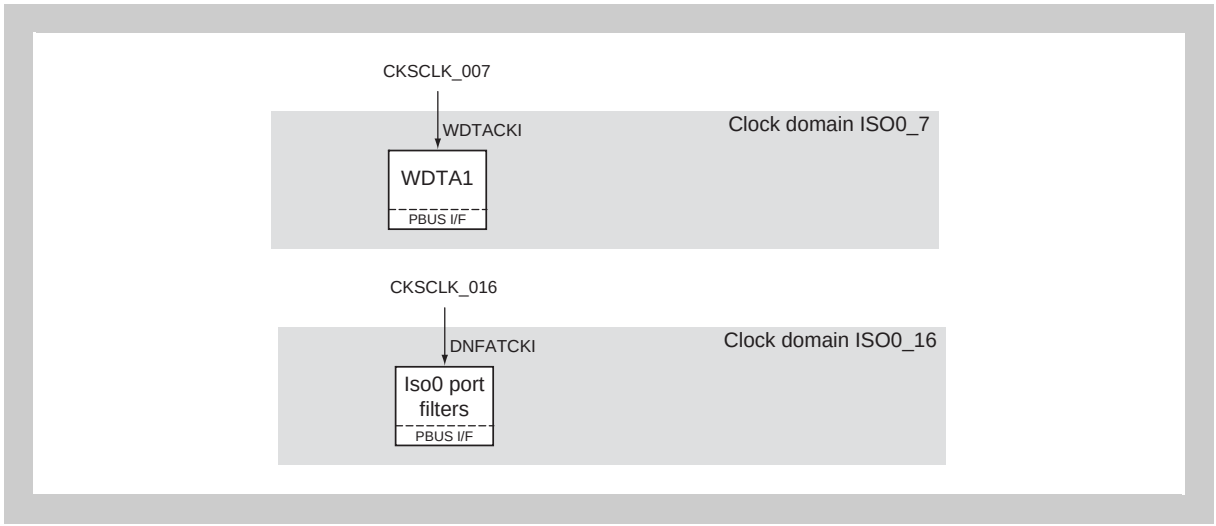



**Figure 7-12 Clock domains ISO0_7, ISO0_16**

### (5) Clock domains ISO0_28, ISO0_29, ISO0_32 to ISO0_34

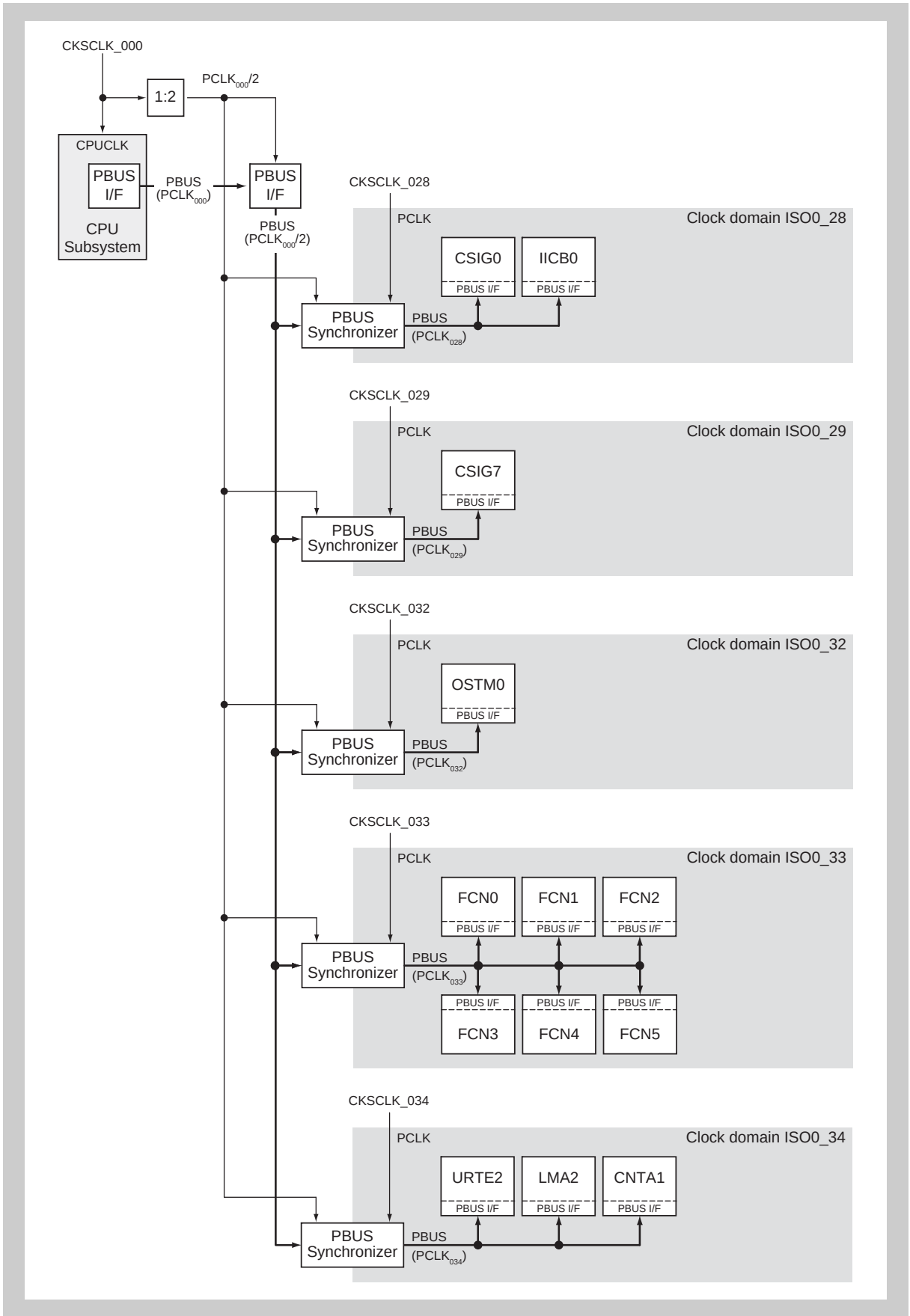



**Figure 7-13 Clock domains ISO0_28, ISO0_29, ISO0_32 to ISO0_34**

## 7.6 Clock Monitor A (CLMA)

This section contains a generic description of the Clock Monitor A (CLMA).

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

### 7.6.1 V850E2/Fx4-G CLMA features

**Instances**

This microcontroller has the following number of instances of the Clock Monitor A.

**Table 7-6 Instances of CLMA**

| Clock Monitor A | |
|---|---|
| Instances | 3 |
| Names | CLMA0, CLMA2, CLMA3 |

**Instances index n**

Throughout this chapter, the individual instances of a Clock Monitor A are identified by the index “n” (n = 0, 2, 3), for example, CLMAnCTL0 for the control register 0 of CLMAn.

**Register addresses**

All CLMAn register addresses are given as address offsets from the individual base address <CLMAn_base>.
The base address <CLMAn_base> of each CLMAn is listed in the following table:

**Table 7-7 Register base addresses <CLMAn_base>**

| CLMAn instance | <CLMAn_base> address |
|---|---|
| CLMA0 | FF80 $2000_H$ |
| CLMA2 | FF80 $4000_H$ |
| CLMA3 | FF80 $5000_H$ |

**Clock supply** The monitored and the sampling clocks of all Clock Monitors A are listed in the following table:

**Table 7-8 CLMAn clock supply**

| CLMAn clock | CLMAn clock | Connected to |
|---|---|---|
| **CLMA0:** | | |
| CLMATSMP | CLMA0 sampling clock | Low Speed IntOsc $f_{RL}$ |
| CLMATMON | CLMA0 monitored clock | MainOsc $f_X$ |
| PCLK | PBUS clock | CKSCLK_A02 |
| **CLMA2:** | | |
| CLMATSMP | CLMA2 sampling clock | Low Speed IntOsc $f_{RL}$ |
| CLMATMON | CLMA2 monitored clock | High speed IntOsc $f_{RH}$ |
| PCLK | PBUS clock | CKSCLK_A02 |
| **CLMA3:** | | |
| CLMATSMP | CLMA3 sampling clock | High Speed IntOsc $f_{RH}$ |
| CLMATMON | CLMA3 monitored clock | PLL0 $f_{PL0}$ |
| PCLK | PBUS clock | CKSCLK_001 |

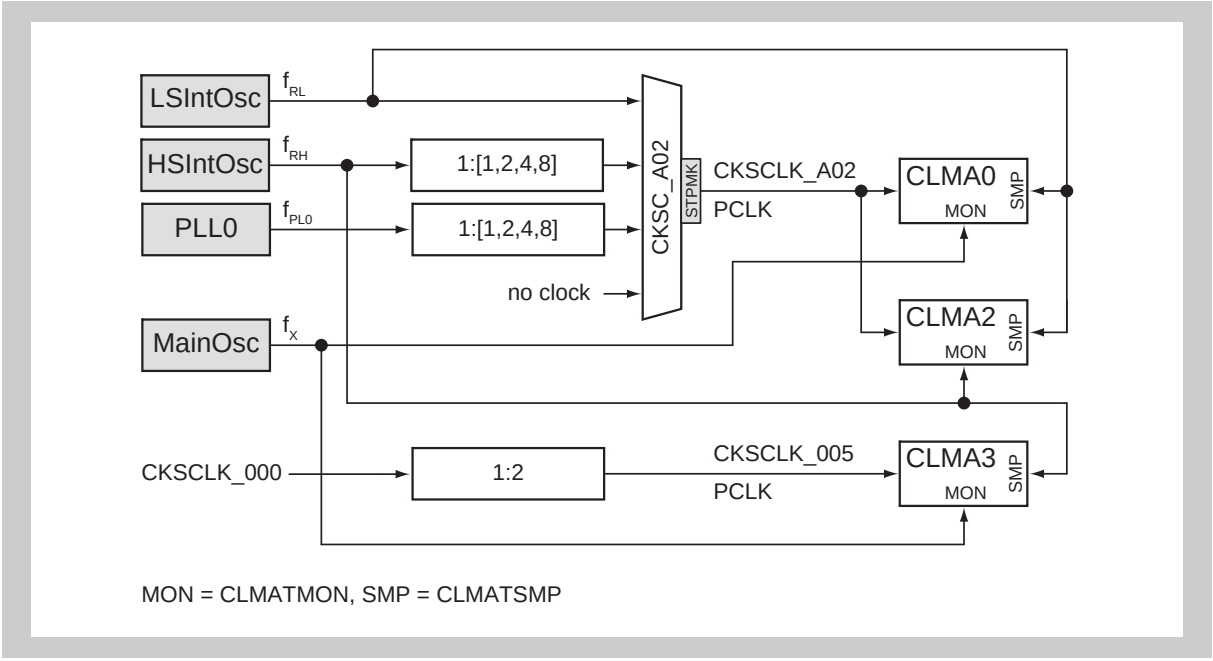



**Figure 7-14 CLMA clock supply**

**Interrupts and reset outputs**

The interrupts and reset outputs of the CLMAn are listed in the table below.

**Table 7-9 CLMA interrupts and reset outputs**

| CLMAn signals | Function | Connected to |
|---|---|---|
| **CLMA0:** | | |
| $\overline{\text{CLMARES}}$ | CLMA0 error reset | Reset Controller $\overline{\text{CLMA0RES}}$ |
| CLMATI | CLMA0 error interrupt request | Interrupt Controller INTCLMA0 |
| **CLMA2:** | | |
| $\overline{\text{CLMARES}}$ | CLMA2 error reset | Reset Controller $\overline{\text{CLMA2RES}}$ |
| CLMATI | CLMA2 error interrupt request | not connected |
| **CLMA3:** | | |
| $\overline{\text{CLMARES}}$ | CLMA3 error reset | Reset Controller $\overline{\text{CLMA3RES}}$ |
| CLMATI | CLMA3 error interrupt request | not connected |

**CLMA H/W reset**

The Clock Monitors A and their registers are initialized by the following reset signal:

**Table 7-10 CLMAn reset signal**

| CLMAn instance | Reset signal |
|---|---|
| CLMA0, CLMA2 | • Reset Controller SYSRES |
| CLMA3 | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

## 7.6.2 CLMA enable and start-up options

### (1) CLMA enable and suspend/resume

**Enable**

A Clock Monitor's operation must be generally enabled by CLMAnCTL0.CLMAnCLME = 1.
General enable after reset is also controlled by the start-up option CLMATC0INI, refer to the section *"CLMA start-up options"* below.

**Suspend/resume**

If generally enabled, a Clock Monitor automatically

- suspends clock supervision, if the clock to be monitored is inactive
- resumes clock supervision, if the clock to be monitored is active.

For that purpose the clock generator, whose output is to be supervised, indicates its active - and stable - status by an active signal, that suspends respectively resumes the associated Clock Monitor's operation via the Clock Monitor's enable signal CLMATEN.

In case the monitored clock is stopped in STOP stand-by mode, the respective Clock Monitor suspends its operation and resumes it if the monitored clock is active and stable again.

**Table 7-11 Clock Monitors start/stop signals**

| CLMAn instance | CLMAn enable signal | Clock generator active signal | Comment |
|---|---|---|---|
| CLMA0 | CLMATEN | MainOsc MOSCSCLKACT | CLMA0 operation suspends/ resumes, if MainOsc clock $f_X$ is inactive/active. |
| CLMA2 | CLMATEN | High Speed IntOsc ROSCSCLKACT | CLMA2 operation suspends/ resumes, if High Speed IntOsc clock $f_{RH}$ is inactive/active. |
| CLMA3 | CLMATEN | PLL0 PLL0CLKACT | CLMA3 operation suspends/ resumes, if PLL0 clock $f_{PL0}$ is inactive/active. |

### (2) CLMA start-up options

The start-up options determine the start-up configuration of the CLMA after reset release. A description of the start-up options is given in the following table.

**Table 7-12 CLMA start-up options**

| Start-up option | Function | Description | Connected to |
|---|---|---|---|
| CLMATC0INI | Defines whether CLMAn is automatically enabled or stays disabled after a reset | Specifies the initial value of CLMAnCTL0.CLMAnCLME:<br>0: CLMA is disabled<br>1: CLMA is enabled | 0 |
| CLMATC1INI | Specifies the error indication method | Specifies the initial value of CLMAnCTL1.CLMAnOSEL:<br>0: Reset request $\overline{\text{CLMATRES}}$ on error<br>1: Interrupt request CLMATI on error | 0 |
| CLMATCSEL | Defines whether the default lower and upper thresholds are supplied by CLMATCLINI[11:0]/ CLMATCHINI[11:0] | Specifies how the threshold registers CLMAnCMPL/CLMAnCMPH are set after reset release:<br>0: set to default reset values<br>1: set by CLMATCLINI[11:0] and CLMATCHINI[11:0] | 0 |
| CLMATCHINI [11:0] | Sets the initial value of the threshold register CLMAnCMPH | Only effective if CLMATCSEL = 1 | $000_H$ |
| CLMATCLINI [11:0] | Sets the initial value of the threshold register CLMAnCMPL | Only effective if CLMATCSEL = 1 | $000_H$ |

### 7.6.3 Functional Overview

The Clock Monitor CLMAn indicates an abnormal frequency of the monitored clock.

**Features summary** It has the following features:

- continuous monitoring of the frequency of an input clock CLMATMON by using a sampling clock CLMATSMP
- indication of abnormal clock frequencies by the following means:
  - output of a reset signal, or
  - generation of an interrupt
- configuration after reset is based on start-up options

**Note** Once enabled, the CLMAn can neither be configured nor stopped by software. Only a reset can stop the CLMAn.

The following figure shows the main components of the Clock Monitor.

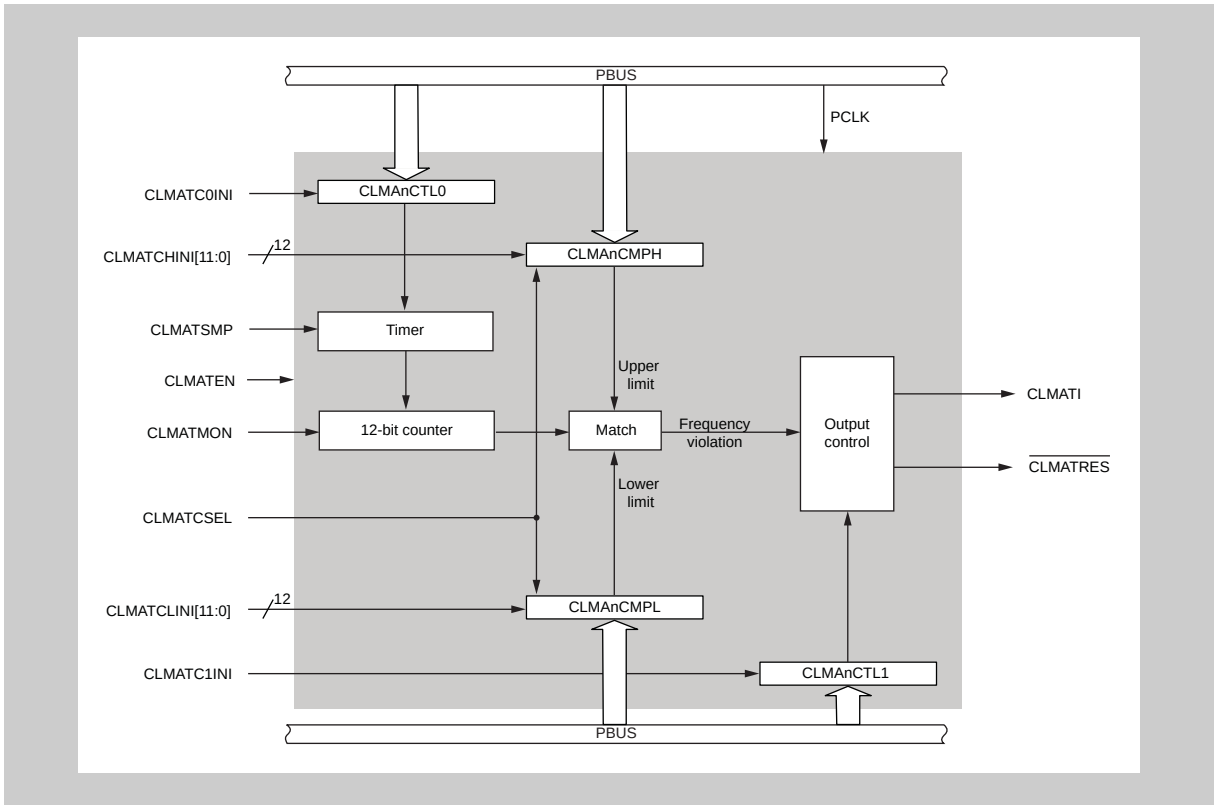



**Figure 7-15 Block diagram of the Clock Monitor A**

### 7.6.4 Functional Description

The Clock Monitor CLMAn is used to ensure that the frequency of a clock (CLMATMON) stays between certain limits.

**(1) Detection of abnormal clock frequencies**

**Method**

1. CLMAn counts the rising edges of the monitored clock CLMATMON within 16 cycles of the sampling clock CLMATSMP and then compares the counter with the configured thresholds:
   – CLMAnCMPL.CLMAnCMPL[11:0] defines the lower threshold.
   – CLMAnCMPH.CLMAnCMPH[11:0] defines the upper threshold.
2. When CLMATMON stops or its frequency is too low, the counter falls below CLMAnCMPL.CLMAnCMPL[11:0].
3. When the frequency of CLMATMON is too high, the counter exceeds CLMAnCMPH.CLMAnCMPH[11:0].

In both cases, CLMAn indicates an abnormal clock frequency as described in *2 “Indication of abnormal clock frequency” on page 297* .

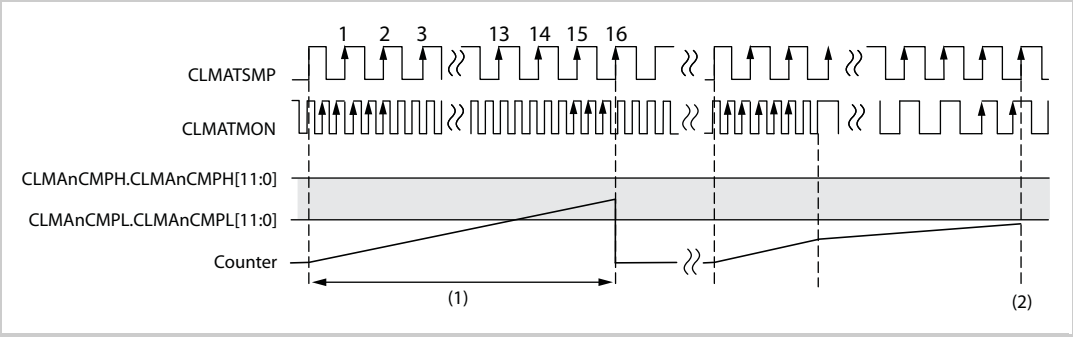



**Figure 7-16 Example: $f_{CLMATMON}$ is too low**

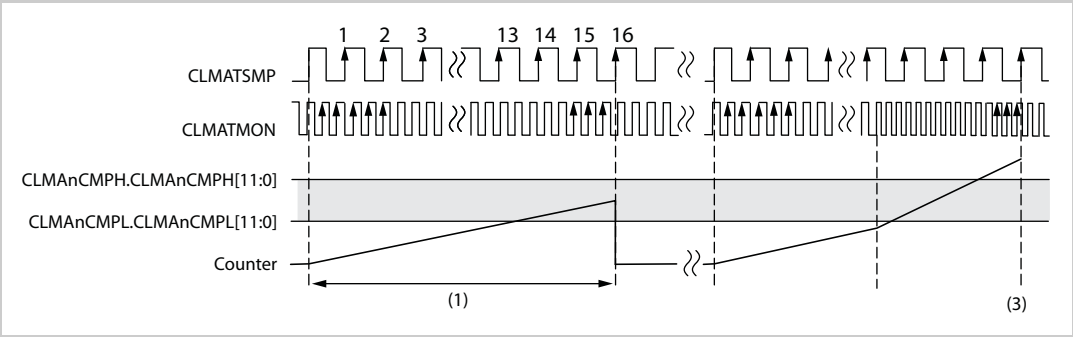



**Figure 7-17 Example: $f_{CLMATMON}$ is too high**

**Note** When $f_{CLMATMON}$ changes within the sampling interval, the counter might be within the valid range although $f_{CLMATMON}$ became too high/low.

The abnormal $f_{CLMATMON}$ is detected one sampling interval later.

**(a) Calculation of thresholds CLMAnCMPL.CLMAnCMPL[11:0] and CLMAnCMPH.CLMAnCMPH[11:0]**

The compare registers CLMAnCMPL and CLMAnCMPH are configured with the minimum and maximum number of clock cycles of CLMATMON that are assumed to be valid within 16 cycles of the sampling clock CLMATSMP. The expected number of clock cycles is denoted by N.

$$\frac{16}{f_{CLMATSMP}} = \frac{N}{f_{CLMATMON}}$$

$$N = \frac{f_{CLMATMON}}{f_{CLMATSMP}} \times 16$$

Considering the allowed frequency deviations of CLMATMON and CLMATSMP, the threshold values can be calculated by the following formulas:

$$\text{Lower threshold} = N_{min}$$

$$= \text{round down} \left( \frac{f_{CLMATMON(min)}}{f_{CLMATSMP(max)}} \times 16 - 1 \right)$$

$$\text{Upper threshold} = N_{max}$$

$$= \text{round up} \left( \frac{f_{CLMATMON(max)}}{f_{CLMATSMP(min)}} \times 16 + 1 \right)$$

**Example** For $f_{CLMATSMP}$ = 240 kHz (± 8%) and $f_{CLMATMON}$ = 16 MHz (± 5%) the recommended threshold values are the following:

$N_{min}$ = [(15.2 MHz / 259.2 kHz) x 16] - 1

= 937.27

CLMAnCMPL = 937 = $03A9_H$

$N_{max}$ = [(16.8 MHz/ 220.8 kHz) x 16] + 1

= 1218.38

CLMAnCMPH = 1219 = $04C3_H$

**Minimum thresholds** The following restrictions must be taken into account:

- CLMAnCMPL ≥ $0001_H$
- CLMAnCMPH ≥ CLMAnCMPL + $0003_H$

**(b) Definition of the initial value input of the threshold registers**

**CLMATCSEL = 1** When CLMATCSEL is active, the initial values of the threshold registers are set by the following input signals:

- CLMAnCMPL[11:0] is set by CLMATCLINI[11:0].
- CLMAnCMPH[11:0] is set by CLMATCHINI[11:0].

**CLMATCSEL = 0** If CLMATCSEL is inactive, the initial values of the threshold registers are set in such a way, that the maximum frequency deviation of the monitored clock is allowed:

- CLMAnCMPL[11:0] = $0001_H$
- CLMAnCMPH[11:0] = $03FF_H$

**Note** Refer to the above section *"CLMA enable and start-up options"* about the setting of the CLMATCSEL signal.

**(2) Indication of abnormal clock frequency**

In case an abnormal frequency is detected, following indications are generated:

**Table 7-13 Abnormal frequency indications**

| CLMAnCTL1. CLMAnOSEL | Monitored clock frequency $f_{CLMATMON}$ is | |
|---|---|---|
| | too low | too high |
| 1 | Interrupt: assertion of CLMATI | Reset: assertion of $\overline{\text{CLMATRES}}$ |
| 0 | Reset: assertion of $\overline{\text{CLMATRES}}$ | |

Note that if a too high frequency is detected the reset CLMATRES is always generated, while in case of too low frequency detection the control bit CLMAnCTL1.CLMAnOSEL determines the generated indicator.

**Note** The initial value of CLMAnCTL1.CLMAnOSEL is defined by the input signal CLMATC1INI. Refer to the section *"CLMA enable and start-up options"* above.

**(3) Enabling and disabling CLMAn**

**Enabling CLMAn** CLMAn is enabled by CLMAnCTL0.CLMAnCLME = 1.

The initial value of CLMAnCTL0 is defined by the input signal CLMATC0INI, refer to the section *"CLMA enable and start-up options"* above.
This allows to define whether CLMAn is automatically enabled after a reset or stays disabled.

**Table 7-14 Initial value of CLMAnCTL0**

| CLMATC0INI | Initial value of CLMAnCTL0 | Function |
|---|---|---|
| Low | $00_H$ | Disable CLMAn |
| High | $01_H$ | Enable CLMAn |

Refer to the section *"CLMA enable and start-up options"* above for information about CLMATC0INI.

**Disabling CLMAn** CLMAn can only be disabled by a reset, not by writing to CLMAnCTL0.

**Note** Once enabled, the CLMAn can neither be configured nor stopped by software, except by a reset.

**(4) Suspend/resume control**

The operation is suspended respectively resumed under control of the CLMATEN signal.
Refer to the section *"CLMA enable and start-up options"* above.

### 7.6.5 Clock Monitor registers

This section contains a description of all registers of the Clock Monitor.

#### (1) Writing to protected registers

Write protected registers are protected from inadvertent write access due to erroneous program execution, etc.

Following Clock Monitor registers feature this special write protection:

- CLMAn control register 0 CLMAnCTL0

Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

#### (2) Clock Monitor registers overview

The Clock Monitor is controlled and operated by the following registers.

**Table 7-15 Clock Monitor registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| CLMAn control register 0 | CLMAnCTL0 | <CLMAn_base> + $00_H$ |
| CLMAn control register 1 | CLMAnCTL1 | <CLMAn_base> + $04_H$ |
| CLMAn comparison register L | CLMAnCMPL | <CLMAn_base> + $08_H$ |
| CLMAn comparison register H | CLMAnCMPH | <CLMAn_base> + $0C_H$ |
| CLMAn emulation register 0 | CLMAnEMU0 | <CLMAn_base> + $18_H$ |

**<CLMAn_base>** The base addresses <CLMAn_base> of the CLMAn are defined in the first part of section *"Clock Monitor A (CLMA)"* in this chapter under the key word *"Register addresses"*.

### (3) CLMAnCTL0 – CLMAn control register 0

This register is used to enable the clock monitor CLMAn.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register CLMAnPCMD.
Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 8-bit units.

**Address** <CLMAn_base> + $00_H$

**Initial Value** Depends on start-up option CLMATC0INI. Refer to the section *"CLMA enable and start-up options"* above.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | CLMAn CLME |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-16 CLMAnCTL0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | CLMAnCLME | Enables/disables the clock monitor:<br>0: Disable CLMAn<br>1: Enable CLMAn<br>This bit can only be cleared by a reset if not automatically enabled by CLMATC0INI. |

### (4) CLMAnCTL1 – CLMAn control register 1

This register specifies the abnormal frequency detection indicator generation output when the frequency of the monitored clock CLMATMON is too low.

**Access** This register can be read/written in 8-bit units.
It can only be written when CLMAn is disabled (CLMAnCTL0.CLMAnCLME = 0).

**Address** <CLMAn_base> + $04_H$

**Initial Value** Depends on start-up option CLMATC1INI. Refer to the section *"CLMA enable and start-up options"* above.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | CLMAn OSEL |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-17 CLMAnCTL1 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | CLMAnOSEL | Specifies the indicator signals that are generated when the frequency of CLMATMON is too low:<br>0: reset CLMATRES<br>1: interrupt CLMATI |

### (5) CLMAnCMPL – CLMAn comparison register L

This register specifies the *lower* frequency limit.

Refer to *a "Calculation of thresholds CLMAnCMPL.CLMAnCMPL[11:0] and CLMAnCMPH.CLMAnCMPH[11:0]" on page 296* for details.

**Access** This register can be read/written in 16-bit units.
It can only be written, when CLMAn is disabled (CLMAnCTL0.CLMAnCLME = 0).

**Address** <CLMAn_base> + $08_H$

**Initial Value** $0001_H$
When CLMATCSEL is active, the initial value is set by the input signals CLMATCLINI[11:0]. Refer to the section *"CLMA enable and start-up options"* above.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | CLMAnCMPL[11:0] | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-18 CLMAnCMPL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 11 to 0 | CLMAnCMPL[11:0] | Specifies the lower threshold<br>• The recommended value is round down[($f_{CLMATMON(min)}$/ $f_{CLMATSMP(max)}$) x 16 - 1]<br>• The minimum value is $0001_H$. |

### (6) CLMAnCMPH – CLMAn comparison register H

This register specifies the *upper* frequency limit.

Refer to *a "Calculation of thresholds CLMAnCMPL.CLMAnCMPL[11:0] and CLMAnCMPH.CLMAnCMPH[11:0]" on page 296* for details.

**Access** This register can be read/written in 16-bit units.
It can only be written, when CLMAn is disabled (CLMAnCTL0.CLMAnCLME = 0).

**Address** <CLMAn_base> + $0C_H$

**Initial Value** $03FF_H$
When CLMATCSEL is active, the initial value is set by the input signals CLMATCHINI[11:0]. Refer to the section *"CLMA enable and start-up options"* above.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | CLMAnCMPH[11:0] | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-19 CLMAnCMPH register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 11 to 0 | CLMAnCMPH[11:0] | Specifies the upper threshold<br>• The recommended value is round up[($f_{CLMATMON(max)}$/ $f_{CLMATSMP(min)}$) x 16 + 1]<br>• The minimum value is CLMAnCMPL + $0003_H$. |

### (7) CLMAnEMU0 – CLMAn emulation register 0

This register provides bits to emulate a frequency deviation error while the microcontroller is set in break mode during debugging.

**Access** This register can be read/written in 8-bit units.

**Address** <CLMAn_base> + $18_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | CLMAn SLFST | CLMAn SLSLW |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-20 CLMAnEMU0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | CLMAnSLSLW | Specifies whether $f_{CLMATMON}$ is assumed to be too low during emulation:<br>0: CLMATMON is within the normal frequency range<br>1: CLMATMON is too slow |
| 1 | CLMAnSLFST | Specifies whether $f_{CLMATMON}$ is assumed to be too high during emulation:<br>0: CLMATMON is within the normal frequency range<br>1: CLMATMON is too fast |

**Caution** It is prohibited to emulate a too low and too high CLMATMON at the same time. Thus CLMAnEMU0 must not be set to $03_H$.

## 7.7 Clock Controller Registers

This section contains a description of all registers of the Clock Controller.

### 7.7.1 Writing to protected registers

Write protected registers are protected from inadvertent write access due to erroneous program execution, etc.

Following Clock Controller registers feature this special write protection:

- MainOsc enable register MOSCE
- High Speed IntOsc enable register ROSCE
- PLL0 enable registers PLLE0
- Clock selector control registers CKSC_mn

Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

### 7.7.2 Clock Controller registers overview

The Clock Controller is controlled and operated by the following registers:

**Table 7-21 Clock Controller registers overview (1/2)**

| Register name | Shortcut | Address |
|---|---|---|
| **Clock generators registers:** | | |
| MainOsc enable register | MOSCE | FF42 $1010_H$ |
| MainOsc status register | MOSCS | FF42 $1014_H$ |
| MainOsc control register | MOSCC | FF42 $1018_H$ |
| MainOsc stabilization time register | MOSCST | FF42 $101C_H$ |
| High Speed IntOsc enable register | ROSCE | FF42 $1000_H$ |
| High Speed IntOsc status register | ROSCS | FF42 $1004_H$ |
| PLL0 enable register | PLLE0 | FF42 $5000_H$ |
| PLL0 status register | PLLS0 | FF42 $5004_H$ |
| PLL0 control register | PLLC0 | FF42 $5008_H$ |
| PLL0 stabilization time register | PLLST0 | FF42 $500C_H$ |
| **Always-On-Area clock selector registers:** | | |
| Clock selector control registers for Always_On_Area | CKSC_An | FF42 $2000_H$ + n x 16 |
| Clock selector status registers for Always_On_Area | CSCSTAT_An | FF42 $2004_H$ + n x 16 |

**Table 7-21 Clock Controller registers overview (2/2)**

| Register name | Shortcut | Address |
|---|---|---|
| **Isolated-Area-0 clock selector registers:** | | |
| Clock selector control registers for Isolated-Area-0 | CKSC_0n | FF42 $6000_H$ + n x 16 |
| | CKSC_1n | FF42 $A000_H$ + n x 16 |
| Clock selector status registers for Isolated-Area-0 | CSCSTAT_0n | FF42 $6004_H$ + n x 16 |
| | CSCSTAT_1n | FF42 $A004_H$ + n x 16 |

### 7.7.3 Clock generators registers

#### (1) MOSCE - MainOsc enable register

This register is used to start and stop the MainOsc and to specify its operation during stand-by modes.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register PROTCMD2.
Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 32-bit units.

**Address** FF42 $1010_H$

**Initial Value** 0000 $0004_H$. This register is initialized by the power-up reset PURES (Power-On-Clear or debugger reset).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | MOSCESTPMK | MOSCEDISTRG | MOSCEENTRG |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-22 MOSCE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | MOSCE STPMK | MainOsc stop request mask<br>0: stop request not masked<br>1: stop request masked<br>If the MainOsc stop request is masked (MOSCESTPMK = 1) a stop request does not change the status of the MainOSC.<br>If MOSCESTPMSK = 0, the MainOsc is stopped in stand-by mode.<br>Upon wake-up from stand-by mode, the MainOsc returns to its status as before stand-by mode, i.e. continues operation when it was operating or remains stopped, when it was stopped before.<br>**Note:** The MainOsc can also be activated after stand-by wake-up, independent of its status before stand-by mode. This features is controlled by the oscillator wake-up mask register OSCWUFMSK. Refer to section *"Wake-up"* in the chapter *"Stand-by Controller (STBC)"* for details. |
| 1 | MOSCE DISTRG | MainOsc disable trigger<br>0: no function<br>1: stops MainOsc<br>Stopping the MainOsc by MOSCEDISTRG = 1 is only possible, if the MainOsc is active, i.e. MOSCS.MOSCSCLKACT = 1.<br>Reading of this bit returns always 0. |
| 0 | MOSCE ENTRG | MainOsc enable trigger<br>0: no function<br>1: starts MainOsc<br>Starting the MainOsc by MOSCEENTRG = 1 is only possible, if the MainOsc is inactive, i.e. MOSCS.MOSCSCLKACT = 0.<br>Reading of this bit returns always 0. |

### (2) MOSCS - MainOsc status register

This register provides various status information about the MainOsc status.

**Access** This register can be read in 32-bit units.

**Address** FF42 $1014_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by the power-up reset PURES (Power-On-Clear or debugger reset).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | MOSCS CLK EN | MOSCS CLK ACT | MOSCS CLK STAB |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 7-23 MOSCS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | MOSCS CLKEN | MainOsc enable status<br>0: MainOsc is disabled<br>1: MainOsc is enabled |
| 1 | MOSCS CLKACT | MainOsc activation status<br>0: MainOsc is inactive<br>1: MainOsc is active |
| 0 | MOSCS CLKSTAB | MainOsc stabilization status<br>0: MainOsc is unstable<br>1: MainOsc is stable |

### (3) MOSCC - MainOsc control register

This register is used to specify several options of the MainOsc start-up behaviour.

This register can only be written, if the MainOsc is disabled (MOSCS.MOSCSCLKEN = 0).

**Access** This register can be read/written in 32-bit units.

**Address** FF42 $1018_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by the power-up reset PURES (Power-On-Clear or debugger reset).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | MOSCC SHT STBY | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-24 MOSCC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | MOSCC SHTSTBY | Short stabilization time mode<br>0: normal stabilization time mode: normal amplification gain<br>1: short stabilization time mode: MainOsc amplification gain is maximum<br>**Note:**<br>MOSCCSHTSTBY = 1 is only effective during the MainOsc stabilization time. After the MainOsc has become active (MOSCS.MOSCSCLKACT = 1), normal amplification gain is used. |

**Caution** During assertion of the external $\overline{\text{RESET}}$ the MainOsc is always operating with its maximum amplification gain.
Refer to the descrition of the external $\overline{\text{RESET}}$ in the *"Reset Controller"* chapter for details.

**(4) MOSCST - MainOsc stabilization time register**

This register determines the MainOsc stabilization time.

This register can only be written, if the MainOsc is disabled (MOSCS.MOSCSCLKEN = 0).

**Access** This register can be read/written in 32-bit units.

**Address** FF42 101C$_H$

**Initial Value** 0000 0000$_H$. This register is initialized by the power-up reset PURES (Power-On-Clear or debugger reset).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | MOST[3:0] | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-25 MOSCST register contents**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>3 to 0</td><td>MOST[3:0]</td><td>MainOsc stabilization time setting<br>Per default the MainOsc stabilization counter is operating with the High Speed IntOsc.<br>If the High Speed IntOsc is disabled (ROSCS.ROSCSCLKEN = 0), the stabilization counter clock is automatically changed to the Low Speed IntOsc.
<table>
<tr><th rowspan="2">MOST[3:0]</th><th colspan="2">MainOsc stabilization time</th></tr>
<tr><th>High Speed IntOsc enabled (ROSCS.ROSCSCLKEN = 1)</th><th>High Speed IntOsc disabled (ROSCS.ROSCSCLKEN = 0)</th></tr>
<tr><td>$0000_B$</td><td>$2^2$ / 8 MHz = 0.5 µs</td><td>$2^2$ / 240 kHz = 17.7 µs</td></tr>
<tr><td>$0001_B$</td><td>$2^3$ / 8 MHz = 1 µs</td><td>$2^3$ / 240 kHz = 33.3 µs</td></tr>
<tr><td>$0010_B$</td><td>$2^4$ / 8 MHz = 2 µs</td><td>$2^4$ / 240 kHz = 66.7 µs</td></tr>
<tr><td>$0011_B$</td><td>$2^5$ / 8 MHz = 4 µs</td><td>$2^5$ / 240 kHz = 133 µs</td></tr>
<tr><td>$0100_B$</td><td>$2^6$ / 8 MHz = 8 µs</td><td>$2^6$ / 240 kHz = 267 µs</td></tr>
<tr><td>$0101_B$</td><td>$2^7$ / 8 MHz = 16 µs</td><td>$2^7$ / 240 kHz = 533 µs</td></tr>
<tr><td>$0110_B$</td><td>$2^8$ / 8 MHz = 32 µs</td><td>$2^8$ / 240 kHz = 1.067 ms</td></tr>
<tr><td>$0111_B$</td><td>$2^9$ / 8 MHz = 64 µs</td><td>$2^9$ / 240 kHz = 2.133 ms</td></tr>
<tr><td>$1000_B$</td><td>$2^{10}$ / 8 MHz = 128 µs</td><td>$2^{10}$ / 240 kHz = 4.267 ms</td></tr>
<tr><td>$1001_B$</td><td>$2^{11}$ / 8 MHz = 256 µs</td><td>$2^{11}$ / 240 kHz = 8.533 ms</td></tr>
<tr><td>$1010_B$</td><td>$2^{12}$ / 8 MHz = 512 µs</td><td>$2^{12}$ / 240 kHz = 17.06 ms</td></tr>
<tr><td>$1011_B$</td><td>$2^{13}$ / 8 MHz = 1.024 ms</td><td>$2^{13}$ / 240 kHz = 34.13 ms</td></tr>
<tr><td>$1100_B$</td><td>$2^{14}$ / 8 MHz = 2.048 ms</td><td>$2^{14}$ / 240 kHz = 68.27 ms</td></tr>
<tr><td>$1101_B$</td><td>$2^{15}$ / 8 MHz = 4.096 ms</td><td>$2^{15}$ / 240 kHz = 136.5 ms</td></tr>
<tr><td>$1110_B$</td><td>$2^{16}$ / 8 MHz = 8.192 ms</td><td>$2^{16}$ / 240 kHz = 273.1 ms</td></tr>
<tr><td>$1111_B$</td><td>$2^{17}$ / 8 MHz = 16.38 ms</td><td>$2^{17}$ / 240 kHz = 546.1 ms</td></tr>
</table>
Notes:<br>• 8 MHz is the nominal frequency of the High Speed IntOsc.<br>• 240 kHz is the nominal frequency of the Low Speed IntOsc.</td></tr>
</table>

**Note** Refer to the Data Sheet for information about the MainOsc stabilization time.

### (5) ROSCE - High Speed IntOsc enable register

This register is used to start and stop the High Speed IntOsc and to specify its operation during stand-by modes.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register PROTCMD2.
Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 32-bit units.

**Address** FF42 $1000_H$

**Initial Value** 0000 $0004_H$. This register is initialized by the power-up reset PURES (Power-On-Clear or debugger reset).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ROSCESTPMK | ROSCEDISTRG | ROSCEENTRG |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-26 ROSCE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | ROSCE STPMK | High Speed IntOsc stop request mask<br>0: stop request not masked<br>1: stop request masked<br>If the High Speed IntOsc stop request is masked (STPMK = 1) the High Speed IntOsc continues operation in any stand-by mode.<br>If STPMSK = 0, the High Speed IntOsc is stopped in a stand-by mode and is re-started upon wake-up from stand-by mode. |
| 1 | ROSCE DISTRG | High Speed IntOsc disable trigger<br>0: no function<br>1: stops High Speed IntOsc<br>Stopping the High Speed IntOsc by ROSCEDISTRG = 1 is only possible, if the High Speed IntOsc is active, i.e. ROSCS.ROSCSCLKACT = 1.<br>Reading of this bit returns always 0. |
| 0 | ROSCE ENTRG | High Speed IntOsc enable trigger<br>0: no function<br>1: starts High Speed IntOsc<br>Starting the High Speed IntOsc by ROSCEENTRG = 1 is only possible, if the High Speed IntOsc is inactive, i.e. ROSCS.ROSCSCLKACT = 0.<br>Reading of this bit returns always 0. |

**Cautions**

1. Pay attention to clock domains, which are using the High Speed IntOsc clock as source via the clock selectors CKSC_mn, when disabling $f_{RH}$ by ROSCE.ROSCEDISTRG = 1.
   In particular all resets set all clock selectors CKSCLK_mn to their default selection, but only the PURES enables the High Speed IntOsc.
   As a consequence all PBUS clocks PCLK will not operate after a reset - except PURES -, since the High Speed IntOsc is selected as their clock source, but the High Speed IntOsc is not enabled.
2. If the High Speed IntOsc is stopped and a reset occurs during a Back-up RAM access, the Back-up RAM may get corrupted. Thus do not disable the High Speed IntOsc in order to avoid Back-up RAM corruption.

### (6) ROSCS - High Speed IntOsc status register

This register provides various status information about the High Speed IntOsc status.

**Access** This register can be read in 32-bit units.

**Address** FF42 $1004_H$

**Initial Value** 0000 $0004_H$. This register is initialized by the power-up reset PURES (Power-On-Clear or debugger reset).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ROSCSCLKEN | ROSCSCLKACT | ROSCSCLKSTAB |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 7-27 ROSCS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | ROSCS CLKEN | High Speed IntOsc enable status<br>0: High Speed IntOsc is disabled<br>1: High Speed IntOsc is enabled |
| 1 | ROSCS CLKACT | High Speed IntOsc activation status<br>0: High Speed IntOsc is inactive<br>1: High Speed IntOsc is active |
| 0 | ROSCS CLKSTAB | High Speed IntOsc stabilization status<br>0: High Speed IntOsc is unstable<br>1: High Speed IntOsc is stable |

### (7) PLLE0 - PLL0 enable register

This register is used to start and stop the PLL0 and to specify its operation during stand-by modes.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register PROTCMD2.
Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 32-bit units.

**Address** FF42 $5000_H$

**Initial Value** 0000 $0004_H$. This register is initialized by the system reset SYSRES.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PLLE0 STP MK | PLLE0 DIS TRG | PLLE0 EN TRG |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-28 PLLE0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | PLLE0 STPMK | PLL0 stop request mask<br>0: stop request not masked<br>1: stop request masked<br>If the PLL0 stop request is masked (PLLE0STPMK = 1) the PLL0 continues operation in any STOP mode.<br>If PLLE0STPMSK = 0, the PLL0 is stopped in Isolated-Area-0 STOP mode and is re-started upon wake-up from stand-by mode.<br>Note that the power supply of PLL0 is switched off, if the Isolated-Area-0 is in DEEPSTOP mode. |
| 1 | PLLE0 DISTRG | PLL0 disable trigger<br>0: no function<br>1: stops PLL0<br>Stopping the PLL0 by PLLE0DISTRG = 1 is only possible, if the PLL0 is active, i.e. PLLS0.PLLS0CLKACT = 1.<br>Reading of this bit returns always 0. |
| 0 | PLLE0 ENTRG | PLL0 enable trigger<br>0: no function<br>1: starts PLL0<br>Starting the PLL0 by PLLE0ENTRG = 1 is only possible, if the PLL0 is inactive, i.e. PLLS0.PLLS0CLKACT = 0.<br>Reading of this bit returns always 0. |

### (8) PLLS0 - PLL0 status register

This register provides various status information about the PLL0 status.

**Access** This register can be read in 32-bit units.

**Address** FF42 $5004_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by any reset.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PLLS0 CLK EN | PLLS0 CLK ACT | PLLS0 CLK STAB |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 7-29 PLLS0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | PLLS0 CLKEN | PLL0 enable status<br>0: PLL0 is disabled<br>1: PLL0 is enabled |
| 1 | PLLS0 CLKACT | PLL0 activation status<br>0: PLL0 is inactive<br>1: PLL0 is active |
| 0 | PLLS0 CLKSTAB | PLL0 stabilization status<br>0: PLL0 is unstable<br>1: PLL0 is stable |

### (9) PLLC0 - PLL0 control register

This register is used to specify the PLL0 output clock $f_{PL0}$ frequency.

This register can only be written, if the PLL0 is disabled (PLLS0.PLLS0CLKEN = 0).

**Access** This register can be read/written in 32-bit units.

**Address** FF42 $5008_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by any reset.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0[a] | 0[a] | 0[a] | 0[a] | 0[a] | 0[a] | 0[a] | 0[a] | 0[a] |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

a) The default values 0 of these bits must not be changed.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0[a] | 0[a] | 0[a] | 0[a] | 0[a] | 0[a] | PLLC0P[1:0] | | 0[a] | 0[a] | 0[a] | PLLC0N[4:0] | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

a) The default values 0 of these bits must not be changed.

**Table 7-30 PLLC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 9 to 8 | PLLC0 P[1:0] | P divider selection |
| 4 to 0 | PLLC0 N[4:0] | Divider value Nr |

P divider selection

| PLLC0 P[1:0] | Divider value Pr | PLL output frequency range |
|---|---|---|
| $00_B$ | – | setting prohibited |
| $01_B$ | – | setting prohibited |
| $10_B$ | 2 | 40 MHz to 64 MHz |
| $11_B$ | 4 | 20 MHz to 32 MHz |

Divider value Nr

| PLLC0N[4:0] | Divider value Nr = PLLCKN[4:0] + 1 |
|---|---|
| $0\ 0100_B$ | 5 |
| $0\ 0101_B$ | 6 |
| ... | ... |
| $1\ 1110_B$ | 31 |
| $1\ 1111_B$ | 32 |

### (10) PLLST0 - PLL0 stabilization time register

This register determines the PLL0 stabilization time.

This register can only be written, if the PLL0 is disabled (PLLS0.PLLS0CLKEN = 0).

**Access** This register can be read/written in 32-bit units.

**Address** FF42 500C$_H$

**Initial Value** 0000 0000$_H$. This register is initialized by any reset.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PLLST0[2:0] | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-31 PLLST0 register contents**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>2 to 0</td><td>PLLST0[2:0]</td><td>PLL0 stabilization time setting<br>Per default the PLL0 stabilization counter is operating with the High Speed IntOsc. If the High Speed IntOsc is disabled (ROSCS.ROSCSCLKEN = 0), the stabilization counter clock is automatically changed to the Low Speed IntOsc.
<table>
<tr><th rowspan="2">PLLST0[2:0]</th><th colspan="2">PLL0 stabilization time</th></tr>
<tr><th>High Speed IntOsc enabled (ROSCS.ROSCSCLKEN = 1)</th><th>High Speed IntOsc disabled (ROSCS.ROSCSCLKEN = 0)</th></tr>
<tr><td>$000_B$</td><td>$2^7$ / 8 MHz = 16 µs</td><td>$2^7$ / 240 kHz = 533 µs</td></tr>
<tr><td>$001_B$</td><td>$2^8$ / 8 MHz = 32 µs</td><td>$2^8$ / 240 kHz = 1.067 ms</td></tr>
<tr><td>$010_B$</td><td>$2^9$ / 8 MHz = 64 µs</td><td>$2^9$ / 240 kHz = 2.133 ms</td></tr>
<tr><td>$011_B$</td><td>$2^{10}$ / 8 MHz = 128 µs</td><td>$2^{10}$ / 240 kHz = 4.267 ms</td></tr>
<tr><td>$100_B$</td><td>$2^{11}$ / 8 MHz = 256 µs</td><td>$2^{11}$ / 240 kHz = 8.533 ms</td></tr>
<tr><td>$101_B$</td><td>$2^{12}$ / 8 MHz = 512 µs</td><td>$2^{12}$ / 240 kHz = 17.057 ms</td></tr>
<tr><td>$110_B$</td><td>$2^{13}$ / 8 MHz = 1.024 ms</td><td>$2^{13}$ / 240 kHz = 34.133 ms</td></tr>
<tr><td>$111_B$</td><td>$2^{14}$ / 8 MHz = 2.048 ms</td><td>$2^{14}$ / 240 kHz = 68.267 ms</td></tr>
</table>
Note:<br>• 8 MHz is the nominal frequency of the High Speed IntOsc.<br>• 240 kHz is the nominal frequency of the Low Speed IntOsc.</td></tr>
</table>

**Note** Refer to the Data Sheet for information about the PLL0 stabilization time.

## 7.7.4 Clock selector control register

### (1) CKSC_mn – Clock selector control registers

These registers select the clock for all clock domains which allow to select different clocks. For details on the selectable values for each register and its initial value refer to the section *"Clock Selection"* of this chapter.

**Caution** Changing safely the clock selection for a clock domain requires particular attention.

Refer to *"Clock switching*" in the section *"Clock Selection"* of this chapter.

**Protection** Writing to these registers is protected by a special sequence of instructions by using a protection command register, that depends on the power area:

- m = 0: Iso0
  The access to writing CKSC_0n registers is protected by the protection register PRTCMD0.
- m = A: AWO
  The access to writing CKSC_An registers is protected by the protection register PRTCMD2.

Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** These registers can be read/written in 32-bit units.

**Address** Isolated-Area-0 clock selectors CKSC_0n: FF42 $6000_H$ + n x 16
Always-On-Area clock selectors CKSC_An: FF42 $2000_H$ + n x 16

**Initial Value** Refer to the section *"Clock Selection"*.

These registers are initialized following resets:

- Isolated-Area-0 clock selectors CKSC_0n:
  - Reset Controller: SYSRES
  - Stand-by Controller:
    DPSTPWU (wake-up from DEEPSTOP mode)
- Always-On-Area clock selectors CKSC_An:
  - Reset Controller: SYSRES

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CKSCID_mn[30:23] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CKSCID_mn[22:15] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CKSCID_mn[14:7] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CKSCID_mn[6:0] | | | | | | | STPMK_mn |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 7-32 CKSC_mn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 1 | CKSCID _mn[30:0] | Clock Source ID:<br>These bits specify the clock CKSCLK_mn for clock domain mn. |
| 0 | STPMK_mn | Controls the clock output CKSCLK_mn during stand-by mode:<br>0: output clock CKSCLK_mn is stopped during stand-by mode<br>1: output clock CKSCLK_mn continues during stand-by mode |

### (2) CSCSTAT_mn – Clock selector status registers

These status registers return the ID of the clock source, that is currently selected by CKSC_mn and its activity status.

**Access** These registers can be read in 32-bit units.

**Address** Isolated-Area-0 clock selectors status CSCSTAT_0n: FF42 $6004_H$ + n x 16
Always-On-Area clock selectors status CSCSTAT_An: FF42 $2004_H$ + n x 16

**Initial Value** Refer to the section *"Clock Selection"*.

These registers are initialized following resets:

- Isolated-Area-0 clock selectors CKSC_0n:
  - Reset Controller: SYSRES
  - Stand-by Controller:
    DPSTPWU (wake-up from DEEPSTOP mode)
- Always-On-Area clock selectors CKSC_An:
  - Reset Controller: SYSRES

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| CLKSELID_mn[30:23] | | | | | | | |
| R | R | R | R | R | R | R | R |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| CLKSELID_mn[22:15] | | | | | | | |
| R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CLKSELID_mn[14:7] | | | | | | | |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CLKSELID_mn[6:0] | | | | | | | CLKACT_mn |
| R | R | R | R | R | R | R | R |

**Table 7-33 CSCSTAT_mn register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 1 | CLKSELID _mn[30:0] | Clock source ID of the current CKSC_mn clock selection |
| 0 | CLKACT_mn | CKSCLK_mn activity indicator<br>0: CKSCLK_mn is disabled<br>1: CKSCLK_mn is enabled |

# Chapter 8 Stand-by Controller (STBC)

This chapter contains a generic description of the Stand-by Controller.

The first section describes all V850E2/Fx4-G specific properties, such as register base addresses and wake-up factors.
The subsequent sections describe the features that apply to all implementations.

## 8.1 V850E2/Fx4-G Stand-by Controller Features

**STBC reset**

The Stand-by Controller and its registers are initialized by the following reset signal:

**Table 8-1 STBC reset signal**

| STBC | Reset signal |
|---|---|
| Stand-by Controller | • Reset Controller SYSRES<br>• Reset Controller PURES |

**Wake-up factors**

The wake-up events for terminating a power save mode are controlled and monitored by the following Stand-by Controller registers:

- WUFL0, WUFMSKL0, WUFCL0
- WUFM0, WUFMSKM0, WUFCM0
- WUFH0, WUFMSKH0, WUFCH0

The assignment of the wake-up events to the wake-up factors control and status register bits is given in the table below.

- WUF[L,M,H]m[31:00] are bits of the wake-up factor registers WUF[L,M,H]m
- WUFMSK[L,M,H]m[31:00] are bits of the wake-up factor mask registers WUFMSK[L,M,H]m
- WUFC[L,M,H]m[31:00] are bits of the wake-up factor clear registers WUFC[L,M,H]m

For details about the wake-up factors control and status registers refer to the section *"Wake-up factor controller registers details"* later on in this chapter.

**Table 8-2 Wake-up factors registers assignments (WUFL0/WUFMSKL0/WUFCL0) (1/2)**

| Assignment of WUFL0/WUFMSKL0/WUFCL0 register bits | | | Wake-up source | | | |
|---|---|---|---|---|---|---|
| | | | Wake up event | Module | Power area | Domain clock |
| WUFL000 | WUFMSKL000 | WUFCL000 | NMI | Port | AWO | - |
| WUFL001 | WUFMSKL001 | WUFCL001 | INTWDTA0 | WDTA0 | AWO | CKSCLK_A07 |
| WUFL002 | WUFMSKL002 | WUFCL002 | INTLVI | LVI | AWO | - |
| WUFL003 | WUFMSKL003 | WUFCL003 | INTKR0 | KR0 | AWO | CKSCLK_A02 |

**Table 8-2 Wake-up factors registers assignments (WUFL0/WUFMSKL0/WUFCL0) (2/2)**

| Assignment of WUFL0/WUFMSKL0/WUFCL0 register bits | | | Wake-up source | | | |
|---|---|---|---|---|---|---|
| | | | Wake up event | Module | Power area | Domain clock |
| WUFL007 | WUFMSKL007 | WUFCL007 | INTP0 | Port | AWO | - |
| WUFL008 | WUFMSKL008 | WUFCL008 | INTP1 | Port | | - |
| WUFL009 | WUFMSKL009 | WUFCL009 | INTP2 | Port | | - |
| WUFL010 | WUFMSKL010 | WUFCL010 | INTP3 | Port | | - |
| WUFL011 | WUFMSKL011 | WUFCL011 | INTP4 | Port | | - |
| WUFL012 | WUFMSKL012 | WUFCL012 | INTP5 | Port | | - |
| WUFL013 | WUFMSKL013 | WUFCL013 | INTP6 | Port | | - |
| WUFL014 | WUFMSKL014 | WUFCL014 | INTP7 | Port | | - |
| WUFL015 | WUFMSKL015 | WUFCL015 | INTP8 | Port | | - |
| WUFL016 | WUFMSKL016 | WUFCL016 | INTP9 | Port | | - |
| WUFL017 | WUFMSKL017 | WUFCL017 | INTP10 | Port | | - |
| WUFL018 | WUFMSKL018 | WUFCL018 | INTP11 | Port | | - |
| WUFL019 | WUFMSKL019 | WUFCL019 | INTP12 | Port | | - |
| WUFL023 | WUFMSKL023 | WUFCL023 | FCN0RX | Port | AWO[a] | - |
| WUFL024 | WUFMSKL024 | WUFCL024 | FCN1RX | Port | | - |
| WUFL025 | WUFMSKL025 | WUFCL025 | FCN2RX | Port | AWO[a] | - |
| WUFL026 | WUFMSKL026 | WUFCL026 | FCN3RX | Port | AWO[a] | - |
| WUFL027 | WUFMSKL027 | WUFCL027 | FCN4RX | Port | AWO[a] | - |
| WUFL028 | WUFMSKL028 | WUFCL028 | FCN5RX | Port | AWO[a] | - |
| WUFL029 | WUFMSKL029 | WUFCL028 | INTFCN0REC | FCN0 | ISO0 | CKSCLK_033 |
| WUFL030 | WUFMSKL030 | WUFCL030 | INTFCN1REC | FCN1 | ISO0 | CKSCLK_033 |
| WUFL031 | WUFMSKL031 | WUFCL031 | SEQWU | SEQ0 | AWO | - |

a) Though the CAN Controllers FCNn reside on the Isolated-Area-0, the CAN bus receive inputs FCNnRX can be selected as a wake-up source in any DEEPSTOP mode.

**Table 8-3 Wake-up factors registers assignments (WUFM0/WUFMSKM0/WUFCM0) (1/2)**

| Assignment of WUFM0/WUFMSKM0/WUFCM0 register bits | | | Wake-up source | | | |
|---|---|---|---|---|---|---|
| | | | Wake up event | Module | Power area | Domain clock |
| WUFM002 | WUFMSKM002 | WUFCM002 | INTTAUJ0I0 | TAUJ0 | AWO | CKSCLK_A03 |
| WUFM003 | WUFMSKM003 | WUFCM003 | INTTAUJ0I1 | | | |
| WUFM004 | WUFMSKM004 | WUFCM004 | INTTAUJ0I2 | | | |
| WUFM005 | WUFMSKM005 | WUFCM005 | INTTAUJ0I3 | | | |
| WUFM015 | WUFMSKM015 | WUFCM015 | INTWDTA1 | WDTA1 | ISO0 | CKSCLK_007 |
| WUFM016 | WUFMSKM016 | WUFCM016 | INTFCN2REC | FCN2 | ISO0 | CKSCLK_033 |
| WUFM017 | WUFMSKM017 | WUFCM017 | INTLMA10IT | LMA10 | ISO0 | CKSCLK_011 |
| WUFM018 | WUFMSKM018 | WUFCM018 | INTLMA10IR | | | |
| WUFM019 | WUFMSKM019 | WUFCM019 | INTLMA10IS | | | |

**Table 8-3 Wake-up factors registers assignments (WUFM0/WUFMSKM0/WUFCM0) (2/2)**

| Assignment of WUFM0/WUFMSKM0/WUFCM0 register bits | | | Wake-up source | | | |
|---|---|---|---|---|---|---|
| | | | Wake up event | Module | Power area | Domain clock |
| WUFM020 | WUFMSKM020 | WUFCM020 | INTLMA11IT | LMA11 | ISO0 | CKSCLK_011 |
| WUFM021 | WUFMSKM021 | WUFCM021 | INTLMA11IR | | | |
| WUFM022 | WUFMSKM022 | WUFCM022 | INTLMA11IS | | | |
| WUFM023 | WUFMSKM023 | WUFCM023 | INTCSIG4IC | CSIG4 | ISO0 | CKSCLK_011 |
| WUFM024 | WUFMSKM024 | WUFCM024 | INTCSIG4IR | | | |
| WUFM025 | WUFMSKM025 | WUFCM025 | INTCSIG4IRE | | | |
| WUFM026 | WUFMSKM026 | WUFCM026 | INTFCN3REC | FCN3 | ISO0 | CKSCLK_033 |
| WUFM027 | WUFMSKM027 | WUFCM027 | INTFCN4REC | FCN4 | ISO0 | CKSCLK_033 |
| WUFM028 | WUFMSKM028 | WUFCM028 | INTFCN5REC | FCN5 | ISO0 | CKSCLK_033 |
| WUFM029 | WUFMSKM029 | WUFCM029 | INTTAUB0I0 | TAUB0 | ISO0 | CKSCLK_006 |
| WUFM030 | WUFMSKM030 | WUFCM030 | INTTAUB0I1 | | | |
| WUFM031 | WUFMSKM031 | WUFCM031 | INTTAUB0I2 | | | |

**Table 8-4 Wake-up factors registers assignments (WUFH0/WUFMSKH0/WUFCH0)**

| Assignment of WUFM0/WUFMSKM0/WUFCM0 register bits | | | Wake-up source | | | |
|---|---|---|---|---|---|---|
| | | | Wake up event | Module | Power area | Domain clock |
| WUFH000 | WUFMSKH000 | WUFCH000 | INTTAUB0I3 | TAUB0 | ISO0 | ISO0_6 |
| WUFH001 | WUFMSKH001 | WUFCH001 | INTTAUB0I4 | | | |
| WUFH002 | WUFMSKH002 | WUFCH002 | INTTAUB0I5 | | | |
| WUFH003 | WUFMSKH003 | WUFCH003 | INTTAUB0I6 | | | |
| WUFH004 | WUFMSKH004 | WUFCH004 | INTTAUB0I7 | | | |
| WUFH005 | WUFMSKH005 | WUFCH005 | INTTAUB0I8 | | | |
| WUFH006 | WUFMSKH006 | WUFCH006 | INTTAUB0I9 | | | |
| WUFH007 | WUFMSKH007 | WUFCH007 | INTTAUB0I10 | | | |
| WUFH008 | WUFMSKH008 | WUFCH008 | INTTAUB0I11 | | | |
| WUFH009 | WUFMSKH009 | WUFCH009 | INTTAUB0I12 | | | |
| WUFH010 | WUFMSKH010 | WUFCH010 | INTTAUB0I13 | | | |
| WUFH011 | WUFMSKH011 | WUFCH011 | INTTAUB0I14 | | | |
| WUFH012 | WUFMSKH012 | WUFCH012 | INTTAUB0I15 | | | |
| WUFH013 | WUFMSKH013 | WUFCH013 | WDTA0NMI | WDTA0 | AWO | CKSCLK_A07 |
| WUFH014 | WUFMSKH014 | WUFCH014 | WDTA1NMI | WDTA1 | ISO0 | CKSCLK_007 |

**Wake-up by URTEnRX**

Though the URTEnRx signals can not be directly used as wake-up factors, external interrupts INTPx can be used instead, since URTEnRX and INTPx are alternative port functions of the same pin.

## 8.2 Stand-by Controller functions

The V850E2/Fx4-G supports following operation respectively stand-by - or power save - modes:

- HALT mode
  HALT mode can be entered from normal run mode by performing the CPU instruction “HALT”. This stops the CPU operation, while all clocks continue to operate and all areas remain under power.
- STOP mode
  In STOP mode the clock supply of the entire CPU Subsystem, including the CPU, is stopped. Thus no program is executed in STOP mode.
  Additionally certain clock supplies of a power domain can be stopped.
  The selection of the clock to stop respectively continue in STOP mode, is done by the clock selector’s stop mask CKSC_mn.STPMK_mn.
- DEEPSTOP mode
  In order to reduce power consumption further, the power supply of the Isolated-Area-0 can be switched off. The power supply of the I/O buffer of the isolated area in DEEPSTOP can also be stopped.
- RUN
  All operation modes with the CPU system on Isolated-Area-0 in operation are called RUN modes.

**Always-On-Area STOP mode**

Always-On-Area STOP mode is no separate stand-by mode, but is activated together with

- STOP mode
- DEEPSTOP mode

and means, that Always-On-Area domain clocks CKSCLK_An can be stopped.

Note that the Always-On-Area’s power supply has to be active all the time.

### 8.2.1 Stand-by Controller signal connections

The Stand-by controller's main signal connections with other microcontroller modules is shown in the diagram below.

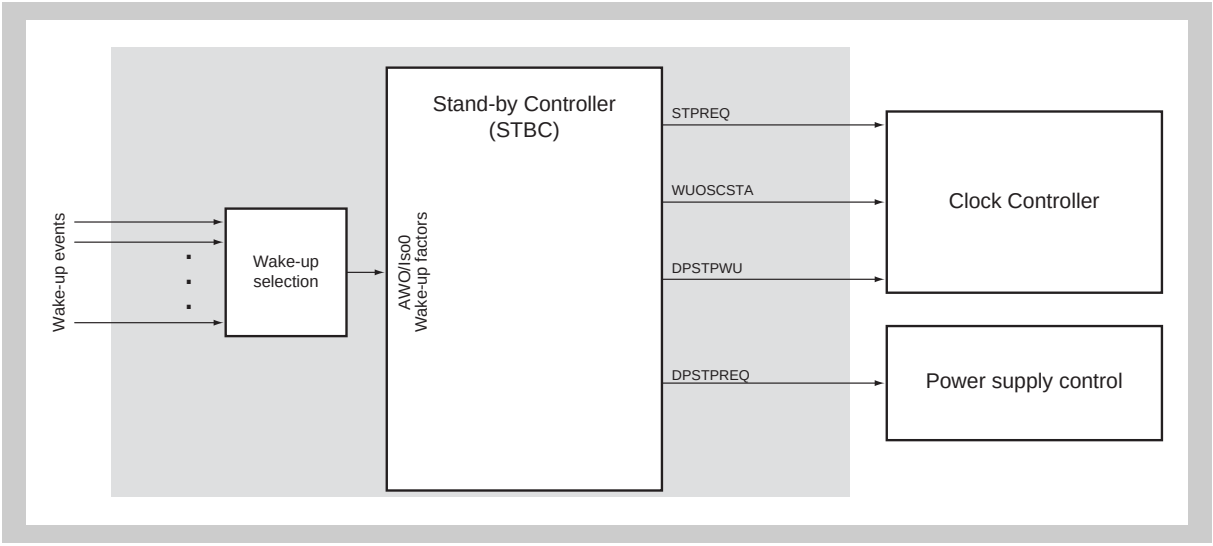



**Figure 8-1 Stand-by Controller main signal connections**

**Wake-up events and factors**

Wake-up events are used from various other modules. Each wake-up event can be separately enabled to wake-up from stand-by mode, and thus becomes a wake-up factor.

Refer to the wake-up factor tables in the first section of this chapter under the key word *"Wake-up factors"*.

**Power supply control signals**

If the Isolated-Area-0 area is set into DEEPSTOP mode, the Stand-by Controller generates following signal towards the Power Sequencer in order to switch off the power supply of the Isolated-Area-0:

- DPSTPREQ is asserted, when the Isolated-Area-0 is set in DEEPSTOP mode.

Refer to the chapter *24 "Power Supply" on page 1211* for details concerning the Power supply control.

**Clock Controller signals**

Upon entering stand-by mode or wake-up from DEEPSTOP the Stand-by Controller generates following signals towards the Clock Controller:

- STPREQ is asserted upon entering STOP or DEEPSTOP mode and is used to stop clock generators and the Isolated-Area-0 and Always-On-Area domain clocks.
- DPSTPWU is asserted upon wake-up from DEEPSTOP and is used to reset the Isolated-Area-0 clock selectors.
- WUOSCSTA can optionally be asserted, when the MainOsc shall start operation upon wake-up from a stand-by mode. The generation of WUOSCSTA is controlled by the OSCWUFMSK.

Refer to the *7 "Clock Controller" on page 254* chapter for details about the Clock Controller behaviour in stand-by.

### 8.2.2 Stand-by modes control

This section describes how the microcontroller is set into stand-by mode.

#### (1) HALT mode control

The HALT mode has no influence on any power or clock domain. Thus this mode is not subject to this chapter. Refer to the document *"V850E2 32-bit Microprocessor Core Architecture"* for details concerning the HALT mode.

#### (2) STOP and DEEPSTOP control

STOP and DEEPSTOP control is accomplished by the stand-by control register PSC0.

The power save control registers PSC0 holds bits to

- select a stand-by mode and initiate its entry (PSC0POF, PSC0STP)
- control the I/O buffers of the Isolated-Area-0 (PSC0IOHLDSET, PSC0IOHLDMSK, PSC0IOHLDCLR), for details refer to the section *"I/O buffer control"* in this chapter
- control the power-down and -up process for DEEPSTOP mode entry and wake-up (PSC1REGSTP), for detail refer to the chapter *24 "Power Supply" on page 1211 .*

**Stand-by mode entry**

To enter a stand-by mode the stand-by mode trigger bit PSC0STP has to be set to 1.

Whether STOP or DEEPSTOP mode is entered by PSC0STP = 1, is determined by the power-off selection bit PSC0POF:

- PSC0STP = 1 and PSC0POF = 0: STOP mode
- PSC0STP = 1 and PSC0POF = 1: DEEPSTOP mode

The following table gives an overview about the stand-by control options:

**Table 8-5 Stand-by modes control**

| Power domain | PSC0STP = 1 | |
|---|---|---|
| | PSC0POF = 0 | PSC0POF = 1 |
| Always-On-Area | STOP | STOP |
| Isolated-Area-0 | STOP | DEEPSTOP |

**(3) Stand-by status**

The status of each power domain can be evaluated by use of the power status register PWS0. This register holds bits, indicating the status of the Isolated-Area-0.

- operational or stand-by mode
  - PWS0PSS = 0: Isolated-Area-0 and Always-On-Area are not in stand-by
  - PWS0PSS = 1: Isolated-Area-0 Isolated-Area-0 and Always-On-Area are in stand-by respectively transition to stand-by ongoing
- I/O buffer hold mode (PWS0IOHOLD)

### 8.2.3 Stand-by modes overview

The clock and power supply options are summarized in the table below.

**Table 8-6 Stand-by modes overview**

| Mode | Always-On | Isolated-Area-0 | |
|---|---|---|---|
| | Clock | Power | Clock |
| RUN | not stopped | on | not stopped |
| | PSC0STP=0 | | |
| STOP | stopped | on | stopped |
| | PSC0STP=1, PSC0POF=0 | | |
| DEEPSTOP | stopped | off | stopped |
| | PSC0STP=1, PSC0POF=1 | | |

**Notes**

1. The clock status in the above table has the following meaning:
   - “not stopped”:
     The clock selectors of the respective power domain do not receive a stop request. Thus its status are not changed.
   - “stopped”:
     The clock selectors of the respective power domain receive a stop request. Thus a certain clock domain clock is stopped, if the clock stop is unmasked by CKSC_mn.STPMK_mn = 0.

   Refer to the section *7.4 “Clock Selection” on page 273* for details concerning clock stop masking.
2. The power status in the above table has the following meaning:
   - on:
     The power supply of the Isolated-Area-0 is retained.
   - off:
     The power supply of the Isolated-Area-0 is switched off.

**Stand-by and On-Chip Debug**

Setting the device in STOP or DEEPSTOP mode is not possible, if the microcontroller is operated under control of an On-Chip Debugger.

### 8.2.4 Clock generators in stand-by

Following table shows the status of the various clock generators in the different stand-by modes.

**Table 8-7 Clock sources in power-save modes**

| Mode | Always-On | | | Isolated-Area-0 |
|---|---|---|---|---|
| | Low Speed IntOsc | High Speed IntOsc | MainOsc | PLL0 |
| RUN | enabled | not stopped | not stopped | not stopped |
| STOP | | stopped | stopped | stopped |
| DEEPSTOP | | stopped | stopped | off |

**Note** The clock generator status in the above table has the following meaning:

- "not stopped":
  The respective clock generator does not receive a stop request and its status is not changed. Thus it remains stopped when it was stopped or is operating in stand-by mode, when it was enabled.
- "stopped":
  The respective clock generator receives a stop request. Thus it is stopped, if the clock stop is unmasked by the associated stop request enable bit
  - MOSCE.MOSCESTPMK = 1 for the MainOsc
  - ROSCE.ROSCESTPMK = 1 for the High Speed IntOsc
  - PLLE0.PLLEkSTPMK = 1 for the PLL0

  Refer to the section *7.3 "Clock Generators" on page 263* for details concerning clock stop masking.
- "off":
  The power supply of the PLL0, which is located on the Isolated-Area-0, is switched off in DEEPSTOP mode.

#### (1) Module clocks during transition to stand-by mode

If Isolated-Area-0 is set in any stand-by mode, the domain clocks CKSCLK_0n of all modules on Isolated-Area-0 must be

- *either* active, i.e. the clock source, selected by the clock ID CKSC_0n.CKSCID[30:0], must be active
- *or* stopped by setting CKSC_0n = 0.

**Caution** Do not set Isolated-Area-0 in stand-by mode, if any module on Isolated-Area-0 is connected to a non-active clock.

### 8.2.5 Wake-up

#### (1) Wake-up events

Following wake-up events are provided to leave a stand-by mode:

**Table 8-8 Wake-up events**

| Mode | Reset | NMI | INTLVI | INTPx[a] | Functional module on power domain[ab] | CAN reception FCNnRX[a] | OCD |
|---|---|---|---|---|---|---|---|
| STOP | yes | yes | yes | yes | • operating on AWO<br>• operating on Iso0 | yes | yes |
| DEEPSTOP | yes | yes | yes | yes | • operating on AWO | yes | yes |

a) The effective wake-up event must be available as a wake-up factor. Refer to the wake-up factor tables in the first section of this chapter under the key word *"Wake-up factors"*.

b) The functional module to issue a wake-up event must be supplied with its clock from the Clock Controller. If its clock domain is subject to the STOP mode, the stand-by mode request must be masked (CKSC_mn.STPMK_mn = 1), thus the clock for the functional module remains in operation during stand-by mode.

**HALT mode wake-up**

The wake-up events that terminate the HALT mode are described in the document *"V850E2 32-bit Microcontroller Core Architecture"*.

**External interrupts INTPx wake-up**

All external interrupts INTPx can terminate all stand-by modes.
If an external interrupt INTPx shall be used to wake-up from a DEEPSTOP mode, the analog filter of INTPx must be configured for edge detection (FCLAnINTm = 0). Refer to the section *2.6 "Port Filters Functional Description" on page 95* for details.

**CAN FCNnRX wake-up**

A falling edge of a CAN reception signal FCNnRX can generate a wake-up from stand-by mode. For further information about CAN wake-up refer to the section "Stand-by Modes" in the chapter "CAN Controller (FCN)".
If FCNnRX wakes up from DEEPSTOP mode, the FCNnRX interrupt service routine is not processed. In this case the FCNnRX is only used as a wake-up factor, thus is not part of any data communication via the CAN interface.

**Functional modules interrupt wake-up**

Interrupts from a functional module can generate a wake-up, provided

- the functional module is not located on Isolated-Area-0 in DEEPSTOP mode (power of Isolated-Area-0 is switched off) and
- the functional module is supplied with its operation clock (clock stop is masked by CKSC_mn.STPMK_m0 = 1) and
- the functional module event is enabled as a wake-up factor (refer to the wake-up factor tables in the first section of this chapter under the key word "Wake-up factors)

**On-Chip Debug wake-up**

The On-Chip Debug unit (OCD) is generating a wake-up event while the microcontroller runs the application program in following cases:

- the debugger issues a stop request
- a breakpoint is hit

In either case any stand-by mode is terminated, provided the OCD debug event is enabled as a wake-up factor via the WUFMSKH register.

**Caution**

If the OCD wake-up event is disabled, it is not possible to wake-up the microcontroller from stand-by mode by a manual stop via the debugger. Thus it is recommended to enable the OCD wake-up for terminating all stand-by modes by setting

WUFMSKH0.WUFMSKH015 = 0.

### (2) Wake-up control

All wake-up events can be separately enabled to trigger a wake-up from stand-by mode by means of a set of wake-up factors control registers:

- WUFMSKL0, WUFMSKM0, WUFMSKH0 = WUFMSK[L,M,H]0
- WUFL0, WUFM0, WUFH0 = WUF[L,M,H]0
- WUFCL0, WUFCM0, WUFCH0 = WUFC[L,M,H]0.
- Wake-up mask registers: WUFMSK[L,M,H]0
  Each bit of these registers is assigned to a certain wake-up event.
  Wake-up by this event is enabled if its mask bit WUFMSK[L,M,H]0.WUFMSK[L,M,H]0[31:0] is set to 0. Upon occurrence of the enabled wake-up event the microcontroller is woken up.
- Wake-up factor registers: WUF[L,M,H]0
  Upon occurrence of an unmasked wake-up event, the associated wake-up factor flag WUF[L,M,H]0.WUF[L,M,H]0[31:0] is set to 1.
  By use of this register the application program can identify the wake-up source.
- Wake-up factor clear registers: WUFC[L,M,H]0
  In order to reset a wake-up factor flag WUF[L,M,H]0[31:0] of a wake-up factor register, its assigned bit WUFC[L,M,H]0.WUFC[L,M,H]0[31:0] has to be set to 1.

**Notes**

1. The wake-up factor flags in the wake-up factors registers WUF[L,M,H]0 indicate only the occurrence of a wake-up factor.
   Thus an asserted wake-up factor flag does not mean, that the transition from stand-by to operation mode of the concerned power domain is already accomplished.
   Refer to *3 “Stand-by status” on page 328* for information how to check the stand-by and operational status.
2. If an interrupt event can also be used as a wake-up event, and this wake-up event is not masked (WUFMSK[L,M,H]0[31:0] = 0), the occurrence of this interrupt event sets also its wake-up factor flag WUF[L,M,H]0[31:0], even if the device is not in stand-by mode.

### (3) CPU wake-up interrupt factor processing

Depending on the stand-by mode, the CPU has to react on wake-up by an interrupt differently:

- STOP mode
  If the microcontroller is woken up from STOP mode, the wake-up interrupt event sets an interrupt request flag (ICn.RFn = 1), provided the interrupt event was not masked (ICn.MKn = 0).
  Thus normal interrupt acknowledge can be performed, after interrupt acknowledgement has been enabled by the “EI” instruction.
- DEEPSTOP mode
  If the microcontroller is woken up from DEEPSTOP mode, the wake-up interrupt event is not recorded as an interrupt request, i.e. no interrupt request flag ICn.RFn = 1 is set.
  Thus the CPU has to read the wake-up factor registers WUF[L,M,H]0 to evaluate the wake-up interrupt event and proceed accordingly.

### (4) MainOsc wake-up

The MainOsc can be automatically started with the wake-up from stand-by. For that purpose the Stand-by Controller generates the wake-up oscillator start signal WUOSCSTA towards the Clock Controller, that starts the MainOsc.

Every wake-up factor, i.e. a wake-up event with its mask WUFMSK[L,M,H]0n cleared, can start the MainOsc.

A control bit in the OSCWUFMSK register allow to select whether to start also the MainOsc upon wake-up:

- OSCWUFMSK.OSCWUFMSK00 = 0
  enables the MainOsc start with wake-up.
- OSCWUFMSK.OSCWUFMSK00 = 1
  disables the MainOsc start with wake-up.

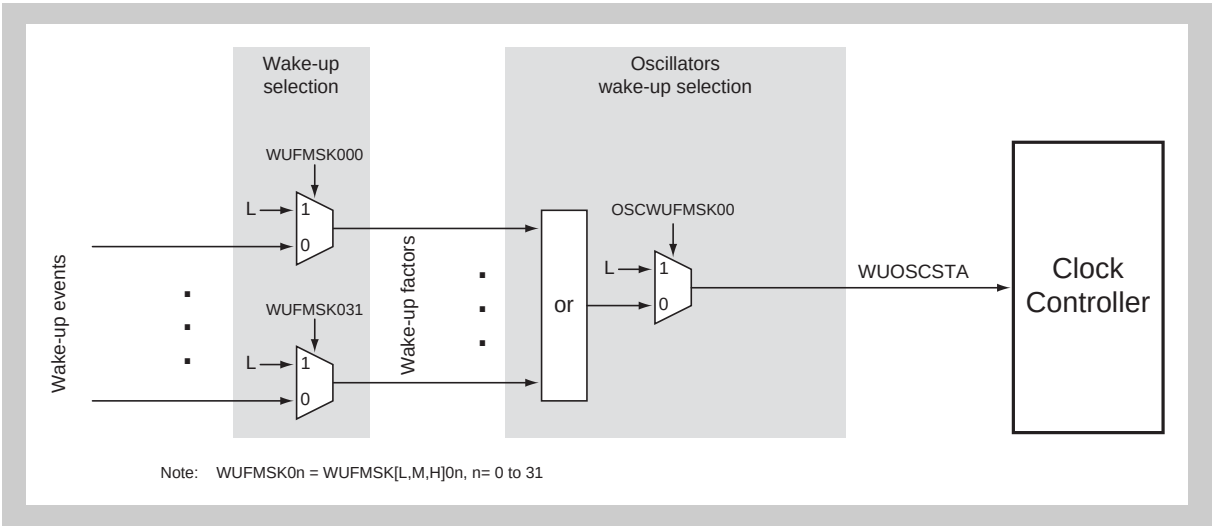



**Figure 8-2 MainOsc start and wake-up**

**(a) MainOsc operating before stand-by**

If the MainOsc was operating before stand-by, it is operating also after wake-up.

The behaviour of the MainOsc during stand-by mode is determined by the MainOsc's stop mask MOSCE.MOSCESTPMK.

**Note** The MainOsc's start refers to the start of their stabilization counter.

**STOP mode** In case of STOP mode the MainOsc wake-up mask OSCWUFMSK00 has no influence.
If stopped during STOP mode, the MainOsc is started at wake-up.

**Table 8-9 MainOsc operating before STOP mode**

| MOSCE STPMSK | During DEEPSTOP | OSCWUFMSK00 | Comment | After wake-up |
|---|---|---|---|---|
| 1 | operating | X | MainOsc remains in operation | operating |
| 0 | stopped | | MainOsc is restarted at wake-up | |

**DEEPSTOP mode** In case of DEEPSTOP mode the MainOsc's wake-up mask OSCWUFMSK00 has an influence on the time the MainOsc is started, as shown in the figure below.

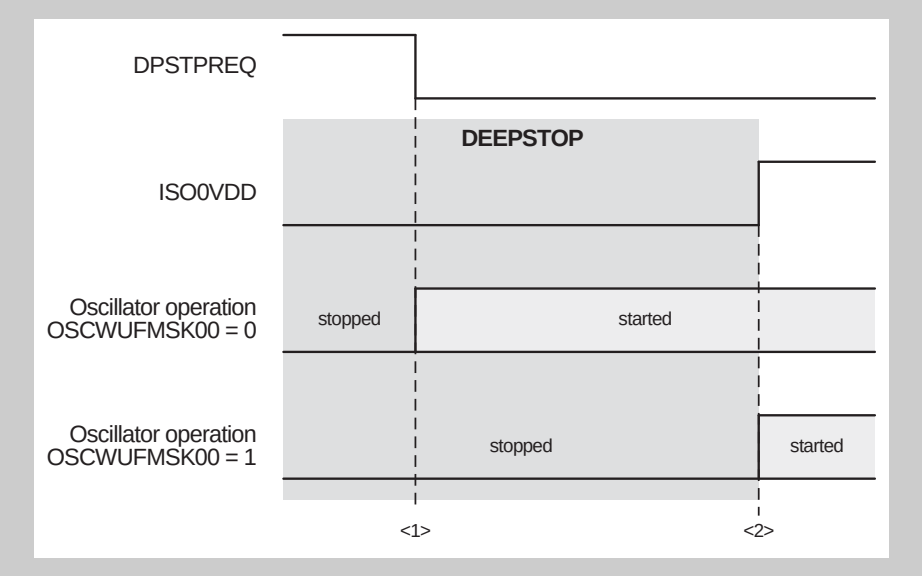



**Figure 8-3 OSCWUFMSK effect on MainOsc start time**

**<1>** If the MainOsc wake-up factor mask is not set (OSCWUFMSK00 = 0), the MainOsc is started upon DEEPSTOP wake-up (DEEPSTOP request signal DPSTPREQ is de-asserted).
Thus the MainOsc is started before the Power Sequencer has switched on the Isolated-Area-0 power supply ISO0VDD and released its reset ISO0RES.

**<2>** If the MainOsc wake-up factor mask is set (OSCWUFMSK00 = 1), the MainOsc is started if the power supply of the Isolated-Area-0 (ISO0VDD) is switched on and it is released from reset.

Refer to the chapter *24 “Power Supply” on page 1211* for detailed information about the Power Sequencer and the wake-up procedures from DEEPSTOP mode.

**Table 8-10 MainOsc operating before DEEPSTOP mode**

| MOSCE STPMSK | During DEEPSTOP | OSCWUFMSK00 | Comment | After wake-up |
|---|---|---|---|---|
| 1 | operating | X | MainOsc remains in operation | operating |
| 0 | stopped | 1 | MainOsc is restarted at Isolated-Area-0 power on (<2>) | |
| | | 0 | MainOsc is restarted at wake-up (<1>) | |

**(b) MainOsc stopped before stand-by**

If the MainOsc was stopped before stand-by, it remains stopped in stand-by in any case.
The MainOsc’s stop mask MOSCE.MOSCESTPMK has no impact.

Depending on the MainOsc wake-up mask OSCWUFMSK00 the MainOsc may remain stopped or start automatically, as shown in the table below:

**Table 8-11 MainOsc stopped before stand-by**

| MOSCE STPMSK | During DEEPSTOP | OSCWUFMSK00 | Comment | After wake-up |
|---|---|---|---|---|
| X | stopped | 1 | MainOsc wake-up masked, no automatic start | stopped |
| | | 0 | MainOsc wake-up unmasked, automatic start at wake-up | operating |

For details about the MainOsc control refer to the description of the MainOsc in the chapter *7 “Clock Controller” on page 254* .

### 8.2.6 I/O buffer control

This section describes the behaviour of the I/O buffers during various stand-by modes.

**Note** The buffers of the port groups P0, P5 and JP0, which reside on the Always-On-Area, always remain in their active state, independent of any stand-by mode.

#### (1) I/O buffer hold state

If an I/O buffer is set into I/O buffer hold state, the state of the buffer is frozen. Thus its input and/or output remains in the state before entering I/O buffer hold state. No external or internal signal can change its state, until the I/O buffer hold state is terminated.

Besides I/O buffer hold states during stand-by modes, the application software can also set all buffers of the Isolated-Area-m into I/O buffer hold state via the power save control registers PSC0:

- PSC0.PSC0IOHLDSET = 1:
  Isolated-Area-0 buffers enter I/O buffer hold state
- PSC0.PSC0IOHLDSET = 0:
  Isolated-Area-0 buffers terminate I/O buffer hold state

**Note** The I/O buffers of the port groups P0 and JP0, which are located on the Always-On-Area, do not support the I/O hold function.

#### (2) I/O buffers during STOP mode

The I/O buffers of areas in STOP mode (clock has been stopped) remain in their configuration. Since the functional module or port function, that is connected to the I/O buffer, also remains in the state before entering STOP mode, it still controls the I/O buffer.

#### (3) I/O buffers during DEEPSTOP mode

The I/O buffers of Isolated-Area-0 in DEEPSTOP are changing into I/O buffer hold state by default, thus the buffer status is not changed.

After wake-up the user application has to re-configure the peripheral or port function, that generates the signals connected to a certain I/O buffer. Afterwards the I/O buffer hold state has to be terminated by setting PSC0.PSC0IOHLDCLR = 1.
After setting this I/O buffer hold clear function trigger the concerned I/O buffer operates as configured by the peripheral or port function.

The automatic change to I/O buffer hold state upon entering DEEPSTOP mode can be suppressed via the power save control registers PSC0:

- PSC0.PSC0IOHLDMSK = 0:
  Isolated-Area-0 buffers *enter* I/O buffer hold state in DEEPSTOP
- PSC0.PSC0IOHLDMSK = 1:
  Isolated-Area-0 buffers *do not enter* I/O buffer hold state in DEEPSTOP

### 8.2.7 Mode transitions

**Note** Following abbreviations are used in the figure above:

- AWO: Always-On-Area
- Iso0: Isolated-Area-0

#### (1) Stand-by mode transitions

The following diagram shows the possible transition between the various operation and stand-by modes.

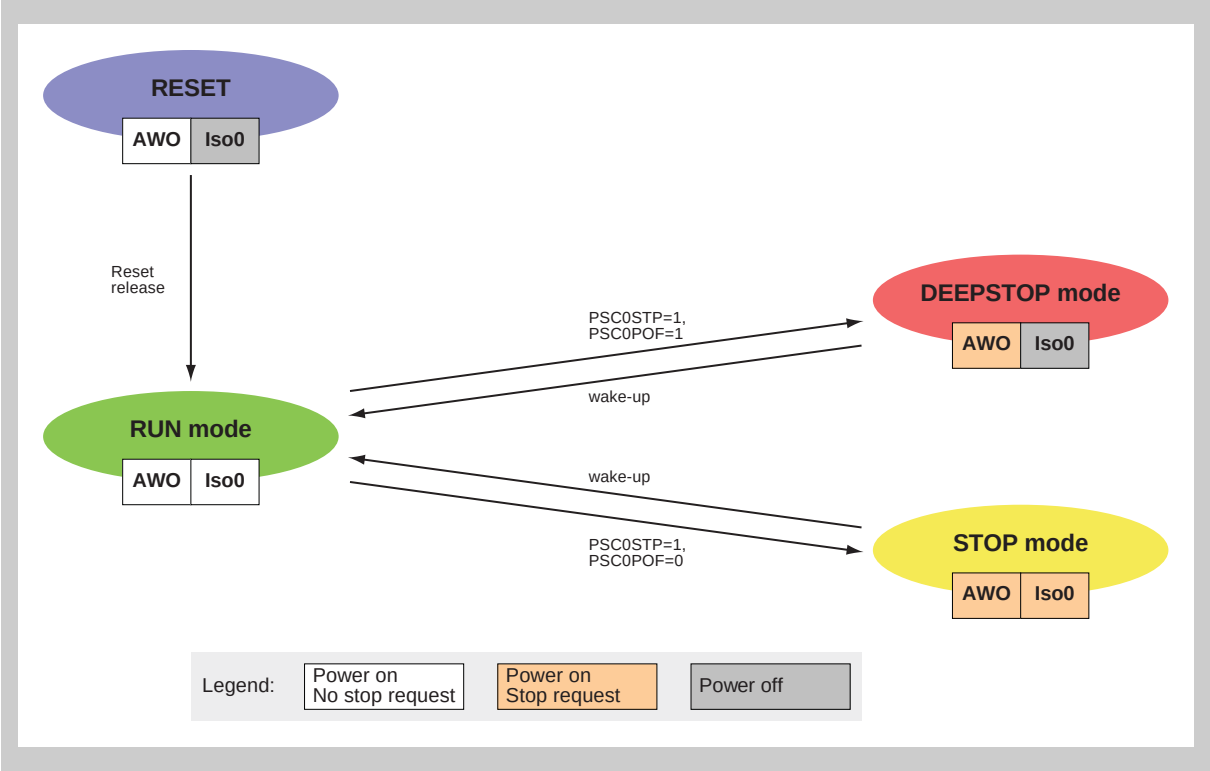



**Figure 8-4 Stand-by mode transitions**

## 8.3 Stand-by modes entry and exit example flows

In the following some recommended example flows are shown how to enter and exit stand-by modes.

**Cautions**

1. Make sure that all modules, which are subject to a stand-by mode are not in operation before starting stand-by mode preparations.
2. If you intend to stop any clock sources (High Speed IntOsc, MainOsc, PLL0) before entering any stand-by mode, make one of the following provisions:
   - Make sure to switch clock domains using a clock source to be stopped to another clock source, that remains active.
   - Stop the respective clock domains by setting the CKSC_mn register to 0.

### 8.3.1 STOP mode

In STOP mode selectable domain clocks of the Always-On-Area (CKSCLK_An) and Isolated-Area-0 (CKSCLK_0n) can be stopped.

#### (1) STOP preparation

Depending on the succession of the STOP mode preparation steps, early occurring wake-up events, to be used as valid wake-up factors, may be discarded or saved as wake-up factors.
Here only one option for preparing the STOP mode is described. Refer to the section *"Application hint: Handling of wake-up events during stand-by mode preparation"* below for other options.

**DMA/interrupt stop**

- Stop all DMA Controller (DMAC) and interrupt activities.

**Interrupts**

Disable interrupt acknowledgement by the CPU instruction "DI".

Interrupt handling, in particular interrupt masking, has no influence on the wake-up behaviour. However if wake-up interrupts shall launch automatic interrupt acknowledgement after wake-up, proceed as follows:

- Clear the interrupt flags (ICn.RFn = 0) and
  - mask non-wake-up interrupts (ICn.MKn = 1)
  - unmask wake-up interrupts (ICn.MKn = 0).

**Prepare domain clocks**

- Select the clock domains to stop or continue operation during STOP. Set the clock stop masks:
  - CKSC_mn.STPMK_mn = 1:
    domain clock CKSCLK_mn operates during STOP mode
  - CKSC_mn.STPMK_mn = 0:
    domain clock CKSCLK_mn stops during STOP mode.

**Prepare clock generators**

- Choose clock generators to operate respectively stop during stand-by mode (by MOSCE.MOSCESTPMK, ROSCE.ROSCESTPMK, PLLE0.PLLE0STPMK bits).
- Choose if the MainOsc shall be automatically started upon a wake-up from selected isolated areas via the OSCWUFMSK register.

**Note** Make sure that clock generators, used as the clock source for domain clocks CKSCLK_mn, which shall not stop in stand-by mode (CKSC_mn.STPMK_mn = 1), must also not be stopped in stand-by.

**Wake-up factors**

- Clear wake-up factor flags in the WUF[L,M,H]0 registers (by WUFC[L,M,H]0n = 1 in WUFC[L,M,H]0 registers) and
  - mask non-wake-up factors (WUFMSK[L,M,H]0n = 1 in WUFMSK[L,M,H]0 registers)
  - unmask wake-up factors to allow them to issue a wake-up (WUFMSK[L,M,H]0n = 0 in WUFMSK[L,M,H]0 registers).

**Caution** If any previously set wake-up factor flag remains set in WUF[L,M,H]0 when starting transition to stand-by mode, an early wake-up will immediately occur and the microcontroller will not enter stand-by mode.

**(2) STOP entry**

- Set PSC0.PSC0POF = 0 and PSC0.PSC0STP = 1 to trigger Isolated-Area-0 transition to STOP mode.
- Since PWS0.PWS0PSS turns immediately to "1" after PSC0STP = 1, run an infinite loop by waiting for PWS0.PWS0PSS = 0.
- While the CPU remains in the infinite loop, transition STOP starts.
- When transition to STOP mode is completed, all domain clocks are stopped, if their stop mask is not set (CKSC_mn.STPMK_mn = 0).
  The CPU is stopped in any case.

**(3) Wake-up before STOP completion (early wake-up)**

If a wake-up occurs before the transition to DEEPSTOP is completed, PWS0.PWS0PSS is set to 0 and the infinite loop is left, before the Isolated-Area-0 set in STOP mode. Wake-up service can start immediately.

**(4) Wake-up service**

- The wake-up factor is evaluated via the wake-up factor flags, which shall be cleared afterwards.
- If the wake-up factor was an interrupt, the interrupt request is stored in the Interrupt Controller and can be acknowledged after interrupt acknowledgement is enabled by the “EI” instruction, if it is unmasked.
- If the wake-up factor was not an interrupt, the wake-up service routine may be called.

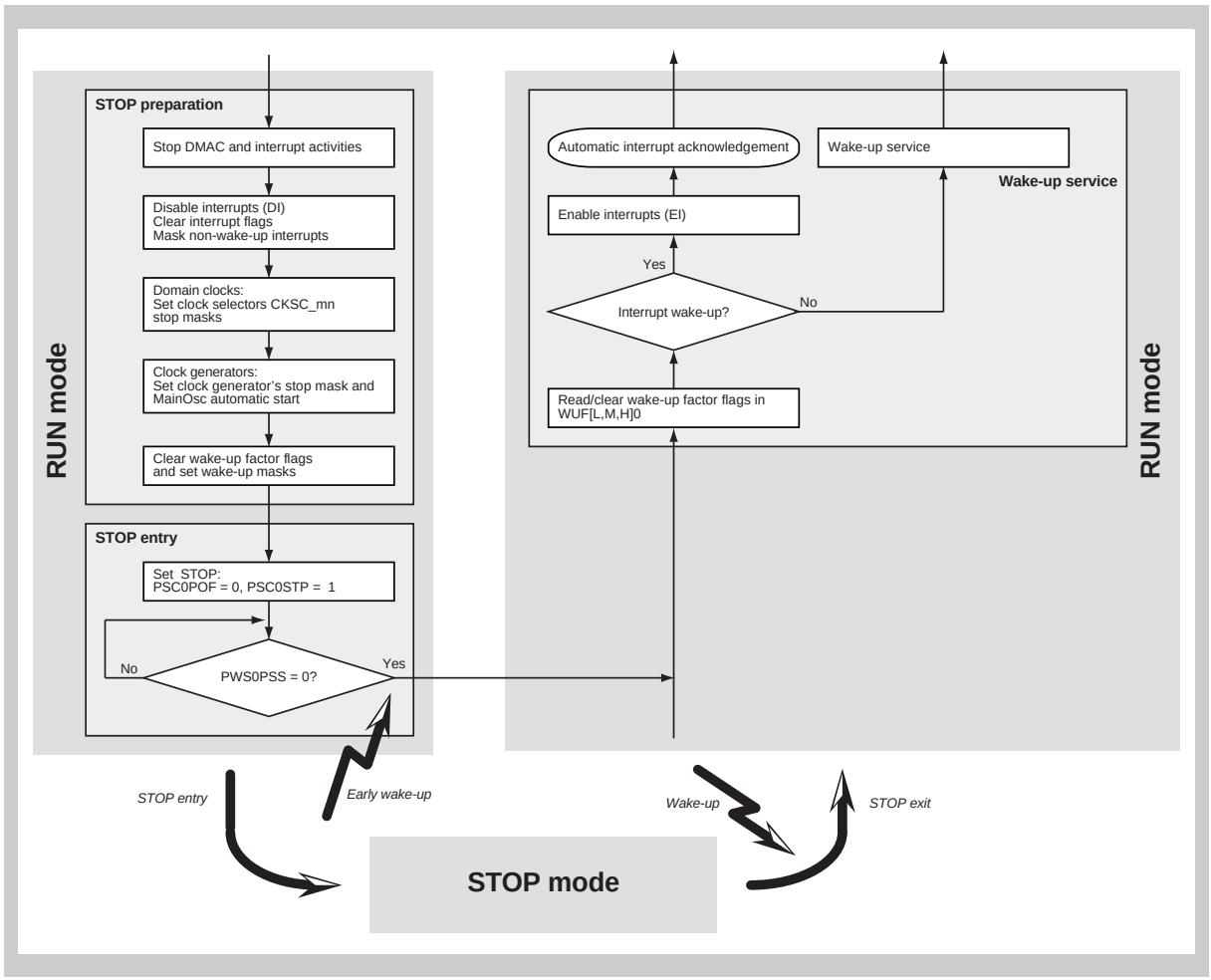



**Figure 8-5 Recommended flow of STOP mode**

### 8.3.2 DEEPSTOP mode

In DEEPSTOP mode the power supply of the Isolated-Area-0 is switched off. Selectable Always-On-Area domain clocks CKSCLK_An are stopped.
Note that also the PLL0 is stopped in DEEPSTOP. This requires to change all Isolated-Area-0 and Always-On-Area domain clocks (CKSCLK_0n, CKSCLK_An) to an active non-PLL0 clock source or to stop the clock domain prior to DEEPSTOP mode.

#### (1) DEEPSTOP preparation

Depending on the succession of the DEEPSTOP mode preparation steps, early occurring wake-up events, to be used as valid wake-up factors, may be discarded or saved as wake-up factors.
Here only one option for preparing the DEEPSTOP mode is described. Refer to the section *"Application hint: Handling of wake-up events during stand-by mode preparation"* below for other options.

**DMA/interrupt stop**

- Stop all DMA Controller (DMAC) and interrupt activities.

**Interrupts**

Disable interrupt acknowledgement by the CPU instruction "DI".

Interrupt handling, in particular interrupt masking, has no influence on the wake-up behaviour. However if wake-up interrupts shall launch automatic interrupt acknowledgement under certain wake-up conditions (refer to *3 "Wake-up before DEEPSTOP completion (early wake-up)" on page 342* ), proceed as follows:

- Clear the interrupt flags (ICn.RFn = 0) and
  - mask non-wake-up interrupts (ICn.MKn = 1)
  - unmask wake-up interrupts (ICn.MKn = 0).

**Prepare Iso0 clocks**

- Change all Isolated-Area-0 domain clocks CKSCLK_0n
  - to a currently active clock sources, which is not derived from PLL0, or
  - stop the respective clock domains by setting the CKSC_0n register to 0.

**Prepare AWO clocks**

- Select the Always-On-Area clock domains to stop or continue operation during DEEPSTOP. Set the clock stop masks:
  - CKSC_An.STPMK = 1:
    domain clock CKSCLK_An operates during DEEPSTOP mode
  - CKSC_An.STPMK = 0:
    domain clock CKSCLK_An stops during DEEPSTOP mode.
- Change all Always-On-Area domain clocks CKSCLK_An to currently active clock sources, which are not derived from any PLL, or stop the clock domains by CKSC_An.CKSCID_An = $0000_H$.

**Prepare clock generators**

- Choose clock generators to operate respectively stop during stand-by mode (by MOSCE.MOSCESTPMK, ROSCE.ROSCESTPMK bits).
- Choose if the MainOsc shall be automatically started upon a wake-up from selected isolated areas via the OSCWUFMSK register.

**Wake-up factors**

- Clear wake-up factor flags in the WUF[L,M,H]0 registers (by WUFC[L,M,H]0n = 1 in WUFC[L,M,H]0 registers) and
  - mask non-wake-up factors (WUFMSK[L,M,H]0n = 1 in WUFMSK[L,M,H]0 registers)
  - unmask wake-up factors to allow them to issue a wake-up (WUFMSK[L,M,H]0n = 0 in WUFMSK[L,M,H]0 registers).

**Caution** If any previously set wake-up factor flag remains set in WUF[L,M,H]0 when starting transition to stand-by mode, an early wake-up will immediately occur and the microcontroller will not enter stand-by mode.

**(2) DEEPSTOP entry**

- Set PSC0.PSC0POF = PSC0.PSC0STP = 1 to trigger transition to DEEPSTOP mode.
- Since PWS0.PWS0PSS turns immediately to "1" after PSC0STP = 1, run an infinite loop by waiting for PWS0.PWS0PSS = 0.
- While the CPU remains in the infinite loop, transition to DEEPSTOP starts.

**(3) Wake-up before DEEPSTOP completion (early wake-up)**

If a wake-up occurs before the transition to DEEPSTOP is completed, PWS0.PWS0PSS is set to 0 and the infinite loop is left, before the Isolated-Area-0 power supply is switched off.
Proceed as described in *5 “DEEPSTOP wake-up processing” on page 343* .

**(4) Wake-up after DEEPSTOP completion**

When transition to DEEPSTOP mode is completed, the power supply for Isolated-Area-0, i.e. also for the CPU, is switched off. Thus the CPU starts from its reset state upon wake-up.

- The reset cause can be evaluated in the following succession:
  - If any reset flag in the RESF register is set, the reset flag in RESF indicates the reset cause.
  - If no wake-up factor flag is set in WUF[L,M,H]0 registers, a power-up reset (Power-On-Clear or debugger reset) occurred.
  - If non of the above reset causes apply, the reset was caused by a DEEPSTOP wake-up.

**(5) DEEPSTOP wake-up processing**

- Since the Isolated-Area-0 was reset, the CPU conducts complete initialization of Isolated-Area-0.

**Iso0 ports**

- Isolated-Area-0 port buffer operation is resumed by
  - configuration of the Iso0 ports
  - release of the Iso0 I/O buffers hold (PSC0.PSC0IOHLDCLR = 1)

**Wake-up service**

- The wake-up factor is evaluated via the wake-up factor flags, which can be cleared afterwards.
- According to the evaluated wake-up factor from the WUF[L,M,H]0 registers, the user program may have to serve the concerned wake-up.
  Note that also a wake-up interrupt does not generate an interrupt request towards the CPU after wake-up from DEEPSTOP mode, so an interrupt is not acknowledged automatically after enabling interrupt acknowledgement by the “EI” CPU instruction.

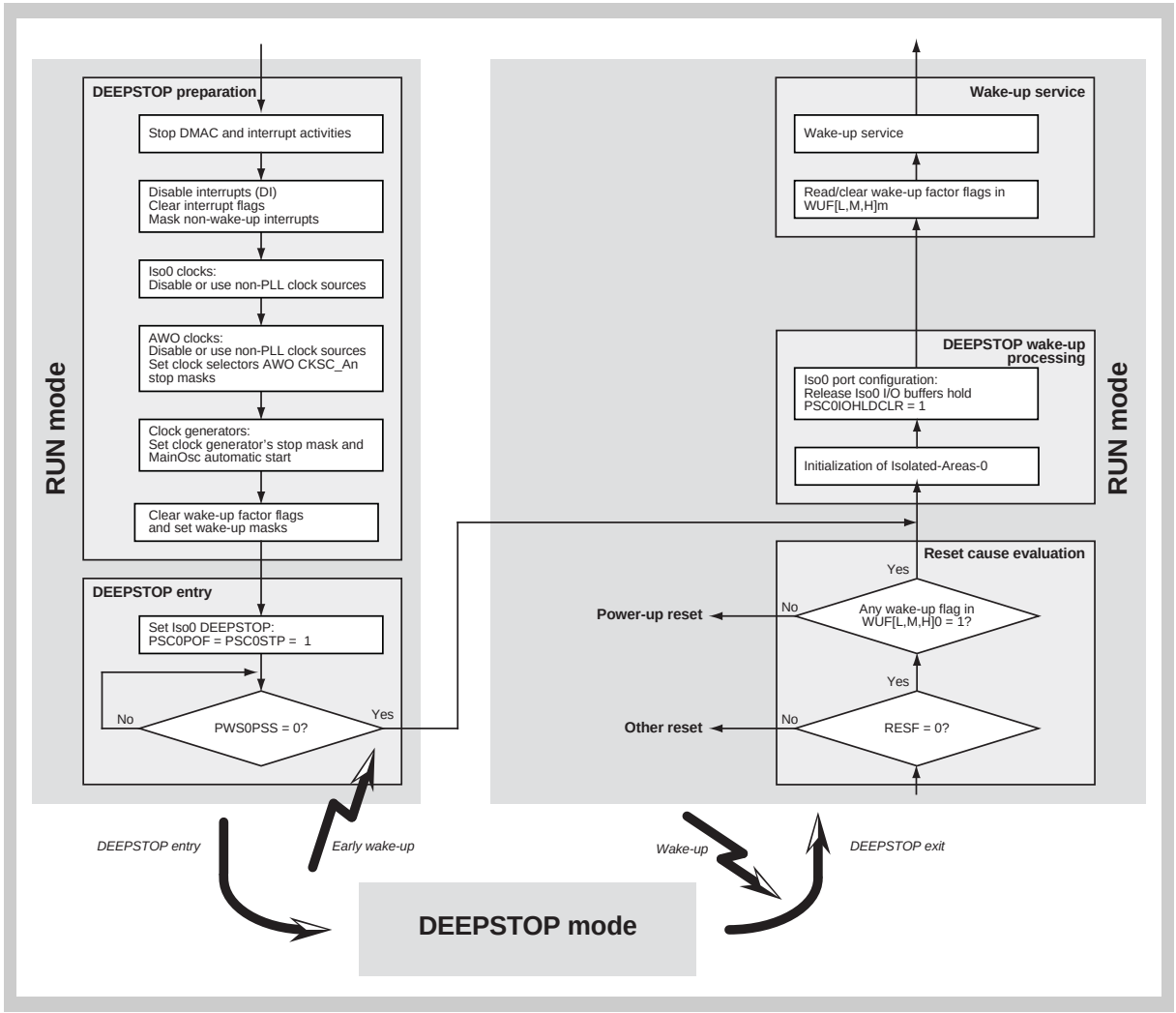



**Figure 8-6 Recommended flow of DEEPSTOP mode**

### 8.3.3 Application hint: Handling of wake-up events during stand-by mode preparation

This application hint describes in more detail how the stand-by mode preparation determines, at which point of the flow wake-up events provoke an early wake-up or are discarded.

Wake-up interrupts set their interrupt request flag (the related interrupt control register bit ICn.RFn), however not all wake-up events are interrupts.

Additionally in case of DEEPSTOP mode, all interrupt request flags will be discarded, if the microcontroller completely enters DEEPSTOP mode.

Thus the occurrence of *all* wake-up events during stand-by mode preparation can only be saved as wake-up flags in the wake-up factor registers WUF[L,M,H]0. Wake-up servicing can then be performed after wake-up from stand-by mode by evaluating the wake-up factor registers.

It depends on the application requirements, after which step of the stand-by entry procedure wake-up events must be recorded and served. In the following two different flows are described as examples.

**Non wake-up interrupts**

All examples assume that interrupts, which shall not be used for wake-up, in particular all interrupts related to modules

- on an isolated area to enter DEEPSTOP
- which are to be stopped in a STOP mode,

are deactivated in the first step of the stand-by mode preparation.
As these interrupts will be masked as interrupts and wake-up events afterwards, they shall not occur.

### (1) Discarding wake-up events during stand-by preparation

The following diagram shows a stand-by preparation flow, where all wake-up events, occurring during the stand-by preparation sequence, are discarded and will not be served after stand-by wake-up.

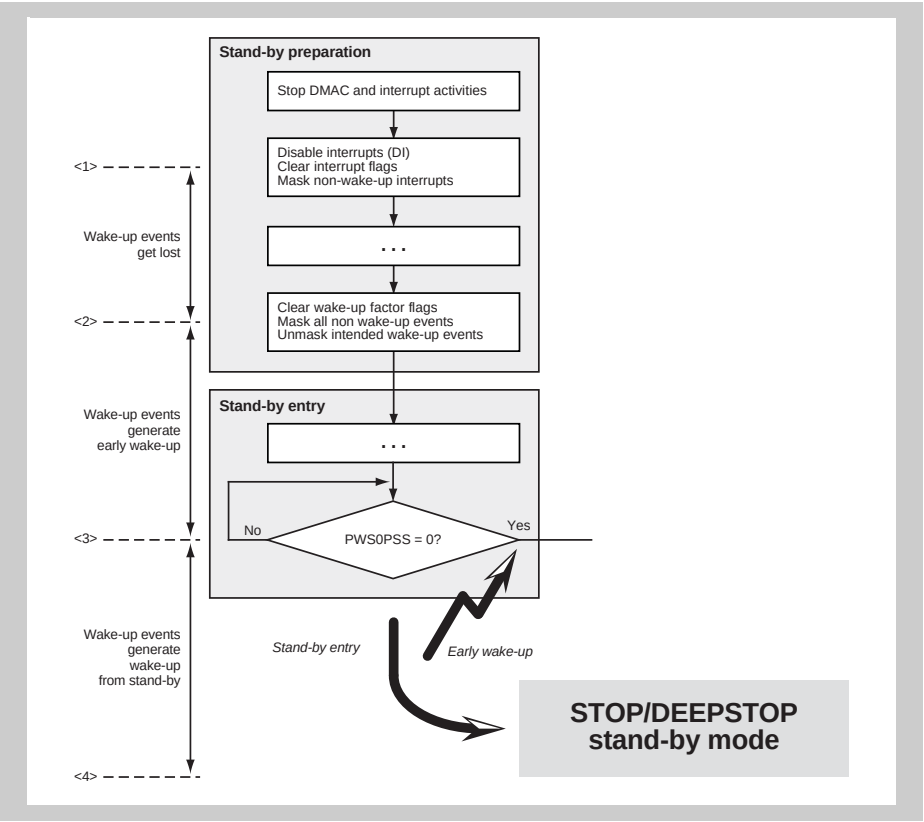



**Figure 8-7 Discarding wake-up events during stand-by preparation**

**<1>** Wake-up interrupts

- are not masked, but not acknowledged yet (because of “DI”),
- are also not saved as wake-up, because wake-up flags WUF[L,M,H]0 are cleared later in <2>.

**<1> – <2>** During <1> and <2> all occurring wake-up events are discarded, thus will neither cause a wake-up nor any wake-up service routine after stand-by wake-up.

**<2>** Since wake-up events are unmasked, their occurrence will be saved in the wake-up flag registers WUF[L,M,H]0 from now on.

**<3>** If an unmasked wake-up event is registered in WUF[L,M,H]0 before stand-by mode is completely entered, an early wake-up will occur.

**<4>** All unmasked wake-up events occurring after stand-by mode was completely entered terminate the stand-by mode.

### (2) Saving wake-up events during stand-by preparation

The following diagram shows a stand-by preparation flow, where wake-up events are registered as wake-up factors already during the stand-by preparation sequence.

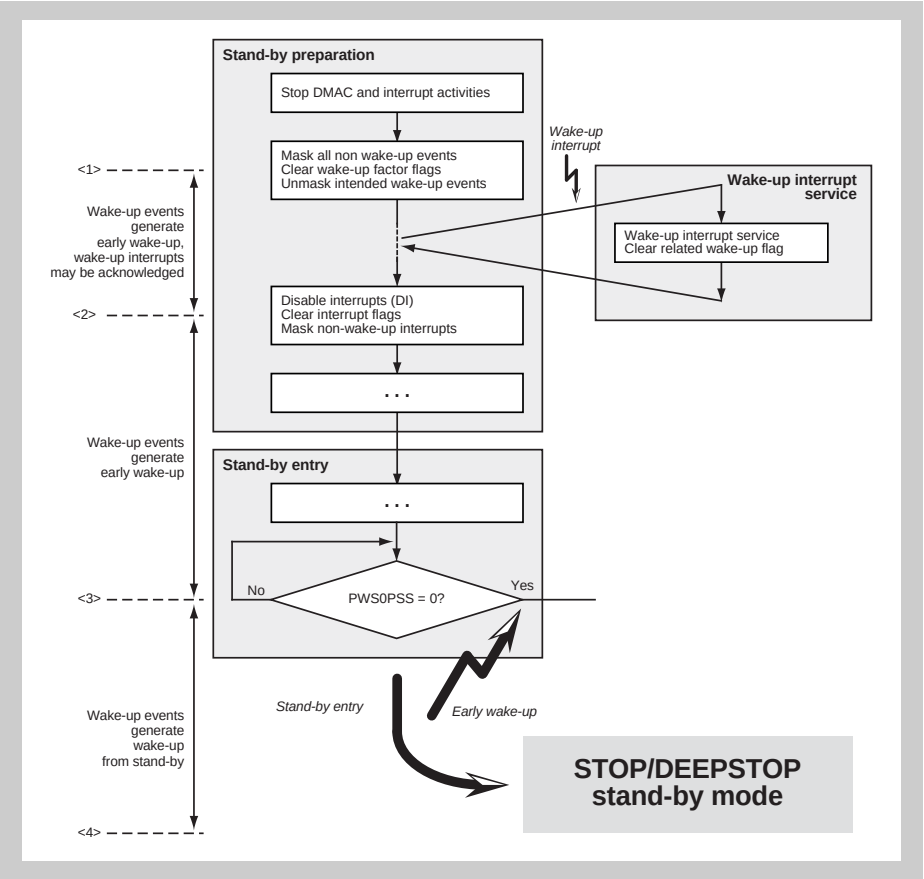



**Figure 8-8 Saving wake-up events during stand-by preparation**

**<1>** All intended wake-up events are unmasked here, thus they will be saved as wake-up factors in the WUF[L,M,H]0 registers upon occurrence and will cause an early wake-up later on.

**<1> – <2>** In case wake-up interrupts are already enabled (ICn.MKn = 0), they will be served.
Since a wake-up interrupt is served here and is also registered as a wake-up factor, the related wake-up factor must be cleared via the WUFC[L,M,H]0 registers in this interrupt service routine. Otherwise it remains registered as a wake-up factor and may be served a second time after stand-by wake-up.

**<2>** After disabling interrupt service in general (DI) and masking all non wake-up interrupts all wake-up events are only saved as wake-up factors in WUF[L,M,H]0 and cause an early wake-up later on.

**<3>** All unmasked wake-up events occurring after stand-by mode was completely entered terminate stand-by mode.

## 8.4 Stand-by Controller Registers

This section contains a description of all the registers of the Stand-by Controller.

### 8.4.1 Writing to protected registers

Write protected registers are protected from inadvertent write access due to erroneous program execution, etc.

Following Stand-by Controller registers feature this special write protection:

- Power save control registers PSC0

Refer to the section *" 3.11 "Write protected Registers" on page 135* for a detailed description how to write to write protected registers.

### 8.4.2 Stand-by Controller registers overview

The Stand-by Controller is controlled and operated by the following registers:

**Table 8-12 Stand-by Controller register overview**

| Register Name | Shortcut | Address |
|---|---|---|
| **Power save registers:** | | |
| Power save control register 0 | PSC0 | FF42 0000$_H$ |
| Power status register 0 | PWS0 | FF42 0004$_H$ |
| **Wake-up factor control registers:** | | |
| Wake-up factor register L | WUFL0 | FF42 0100$_H$ |
| Wake-up factor register M | WUFM0 | FF42 0110$_H$ |
| Wake-up factor register H | WUFH0 | FF42 0120$_H$ |
| Wake-up factor mask register L | WUFMSKL0 | FF42 0104$_H$ |
| Wake-up factor mask register M | WUFMSKM0 | FF42 0114$_H$ |
| Wake-up factor mask register H | WUFMSKH0 | FF42 0124$_H$ |
| Wake-up factor clear register L | WUFCL0 | FF42 0108$_H$ |
| Wake-up factor clear register M | WUFCM0 | FF42 0118$_H$ |
| Wake-up factor clear register H | WUFCH0 | FF42 0128$_H$ |
| **Oscillators wake-up registers:** | | |
| Oscillator wake-up mask register | OSCWUFMSK | FF42 01A4$_H$ |

### 8.4.3 Stand-by Controller control registers details

#### (1) PSC0 – Power save control register 0

This register controls the stand-by modes.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register PROTCMD2.
Refer to the section *"Write protected Registers"* in the chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 32-bit units.

**Address** FF42 $0000_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by a system reset SYSRES.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | PSC0 IOHLDSET | PSC0 IOHLDMSK | 0[a] | PSC0 IOHLDCLR | 0 | PSC0 POF | PSC0 STP |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

a) The default value "0" of bit 4 of PSC0 must be changed to "1", when using DEEPSTOP mode.

**Caution** The default value "0" of bit 4 of PSC0 must be changed to "1", when using DEEPSTOP mode.

**Table 8-13 PSC0 register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 6 | PSC0 IOHLDSET | Isolated-Area-0 I/O buffer hold function trigger<br>0: terminate Isolated-Area-0 I/O buffer hold function<br>1: enter Isolated-Area-0 I/O buffer hold function<br>Read value of this bit is always "0". |
| 5 | PSC0 IOHLDMSK | Isolated-Area-0 I/O buffer hold function mask<br>0: enable Isolated-Area-0 I/O buffer hold function<br>1: disable Isolated-Area-0 I/O buffer hold function<br>If the PSC0IOHLDMSK = 1, the I/O buffers of Isolated-Area-0 do not go into hold mode during DEEPSTOP mode. |
| 4 | Bit 4 | The default value "0" of bit 4 of PSC0 must be changed to "1", when using DEEPSTOP mode. |

**Table 8-13 PSC0 register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 3 | PSC0 IOHLDCLR | Isolated-Area-0 I/O buffer hold clear function trigger<br>0: no function<br>1: clear I/O buffer hold function<br>Reading of this bit returns always 0.<br><br>**Note:** Setting PSC0IOHLDCLR = 1 clears the I/O hold state of all Isolated-Area-0 buffers, independent of how I/O buffer hold was entered: either by setting PSC0IOHLDSET = 1 or automatically during transition to DEEPSTOP mode. |
| 1 | PSC0 POF | Isolated-Area-0 power-off, i.e. DEEPSTOP, selection<br>0: Isolated-Area-0 power remains on during stand-by mode (= STOP)<br>1: Isolated-Area-0 power switched off during stand-by mode (= DEEPSTOP) |
| 0 | PSC0 STP | Always-On-Area and Isolated-Area-0 stand-by mode trigger<br>0: no function<br>1: Always-On-Area and Isolated-Area-0 enter stand-by mode<br>Reading of this bit returns always 0.<br><br>Upon entering stand-by mode with PSC0STP = 1,<br>• Always-On-Area always enters STOP mode<br>• Isolated-Area-0 enters STOP (if PSC0POF = 0) or DEEPSTOP (if PSC0POF = 1) mode |

### (2) PWS0 – Power status register 0

This register shows the stand-by mode status of the Isolated-Area-0.

**Access** This register can be read in 32-bit units.

**Address** FF42 $0004_H$

**Initial Value** $0000\ 0001_H$. This register is initialized by a system reset SYSRES.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PWS0 PSS | 0 | 0 | 0 | 0 | 0 | PWS0 IOHOLD | 1 |
| R | R | R | R | R | R | R | R |

**Table 8-14 PWS0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | PWS0 PSS | Stand-by status:<br>0: microcontroller is in RUN mode<br>1: microcontroller is in the transition to or is in stand-by mode<br>PWS0PSS is set to 1 immediately after stand-by mode trigger was issued (PSC0.PSC0STP = 1). Thus PWS0PSS = 1 indicates the state of transition to stand-by mode and the stand-by mode status.<br>If the stand-by mode was entered, the CPU is also stopped and this register can not be read any more. |
| 1 | PWS0 IOHOLD | I/O buffer hold status of Isolated-Area-0<br>0: I/O buffers of Isolated-Area-0 are not in hold mode<br>1: /O buffers of Isolated-Area-0 are in hold mode |

**Note** This register can not be read in stand-by mode.

### 8.4.4 Wake-up factor controller registers details

#### (1) WUF[L,M,H]0 – Wake-up factor registers

This register informs about the wake-up factor of Isolated-Area-m.

**Notes**

1. For the assignment of the wake-up factors to wake-up factor register bits refer to the key word “Wake-up factors” in the first section of this chapter.
2. Wake-up factor bits WUF0n, which are not assigned to a valid wake-up factor, are fixed to 0.
3. To clear an asserted bit in this register use the WUFC[L,M,H]0 registers.

**Access** This register can be read in 32-bit units.

**Address** WUFL0: FF42 $0100_H$
WUFM0: FF42 $0110_H$,
WUFH0: FF42 $0120_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by a system reset SYSRES.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WUF[L,M,H]0[31:24] | | | | | | | |
| R | R | R | R | R | R | R | R |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WUF[L,M,H]0[23:16] | | | | | | | |
| R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WUF[L,M,H]0[15:8] | | | | | | | |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| WUF[L,M,H]0[7:0] | | | | | | | |
| R | R | R | R | R | R | R | R |

**Table 8-15 WUF[L,M,H]0 registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 0 | WUF [L,M,H]0n | Indicates occurrence of wake-up factor WUF[L,M,H]0n (n = 0 to 31)<br>0: wake-up factor WUF[L,M,H]0n did not occur<br>1: wake-up factor WUF[L,M,H]0n occurred |

### (2) WUFMSK[L,M,H]0 – Wake-up factor mask registers

This register enables wake-up factors.

**Notes**
1. For the assignment of the wake-up factors to wake-up factor register bits refer to the key word “Wake-up factors” in the first section of this chapter.
2. Wake-up mask bits WUFMSK[L,M,H]0n, which are not assigned to a valid wake-up factor, are fixed to 1.

**Access** This register can be read/written in 32-bit units.

**Address** WUFMSKL0: FF42 0104$_H$
WUFMSKM0: FF42 0114$_H$,
WUFMSKH0: FF42 0124$_H$

**Initial Value** FFFF FFFF$_H$. This register is initialized by a system reset SYSRES.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WUFMSK[L,M,H]0[31:24] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WUFMSK[L,M,H]0[23:16] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WUFMSK[L,M,H]0[15:8] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| WUFMSK[L,M,H]0[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 8-16 WUFMSK[L,M,H]0 registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 0 | WUFMSK [L,M,H]0n | Enable of wake-up factor WUF[L,M,H]0n<br>0: wake-up factor WUF[L,M,H]0n enabled<br>1: wake-up factor WUF[L,M,H]0n disabled |

### (3) WUFC[L,M,H]0 – Wake-up factor clear registers

This register clears wake-up factor indicated in the WUF[L,M,H]0 registers.

**Notes**

1. For the assignment of the wake-up factors to wake-up factor register bits refer to the key word “Wake-up factors” in the first section of this chapter.
2. Wake-up factor clear bits WUFC[L,M,H]0n, which are not assigned to a valid wake-up factor, must be written with “0”.

**Access** This register can be written in 32-bit units.

**Address** WUFCL0: FF42 $0108_H$
WUFCM0: FF42 $0118_H$,
WUFCH0: FF42 $0128_H$

**Initial Value** Reading these registers returns always 0000 $0000_H$.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| WUFC[L,M,H]0[31:24] | | | | | | | |
| W | W | W | W | W | W | W | W |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| WUFC[L,M,H]0[23:16] | | | | | | | |
| W | W | W | W | W | W | W | W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WUFC[L,M,H]0[15:8] | | | | | | | |
| W | W | W | W | W | W | W | W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| WUFC[L,M,H]0[7:0] | | | | | | | |
| W | W | W | W | W | W | W | W |

**Table 8-17 WUFC[L,M,H]0 registers contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 0 | WUFC [L,M,H]0n | Clear of wake-up factor WUF[L,M,H]0n of the wake-up factor registers<br>0: no function<br>1: clear WUF[L,M,H]0n |

## 8.4.5 Oscillator wake-up registers details

### (1) OSCWUFMSK – Oscillator wake-up mask register

This register controls the start of the MainOsc upon a wake-up factor.

**Access** This register can be read/written in 32-bit units.

**Address** FF42 01A4$_H$

**Initial Value** 0000 0001$_H$. This register is initialized by a Power-up reset PURES.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | OSCWUFMSK00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 8-18 OSCWUFMSK register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | OSCWUFMSK00 | Enables wake-up factor for starting the MainOsc<br>0: MainOsc starts by wake-up factor<br>1: MainOsc does not start by wake-up factor |

**Note** If OSCWUFMSK00 = 0, the MainOsc is always started upon wake-up, even if the MainOsc was stopped before the stand-by mode was entered.

# Chapter 9 Wake-up Sequencer (SEQ)

This chapter contains a generic description of the Wake-up Sequencer (SEQ).

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 9.1 V850E2/Fx4-G SEQ Features

**Instances**

This microcontroller has the following number of instances of the Wake-up Sequencers.

**Table 9-1 Instances of SEQ**

| Wake-up Sequencer | |
|---|---|
| Instances | 1 |
| Names | SEQ0 |

**Instances index n**

Throughout this chapter, the individual instances of a Wake-up Sequencers are identified by the index “n” (n = 0), for example, SEQnSCTR for the SEQn control register.

**Digital input signal index m**

Throughout this chapter, the digital input signals are identified by the index “m”. Thus the digital signal inputs are named DPINm.
The valid index “m” for each V850E2/Fx4-G device is given in the table below.

**Table 9-2 Digital input signal index m**

| Index | V850E2/FF4-G<br>V850E2/FG4-G |
|---|---|
| m = | 0 to 13 |

**Register addresses**

All SEQn register addresses are given as address offsets from the individual base address <SEQn_base>.
The base address <SEQn_base> of each SEQn is listed in the following table:

**Table 9-3 Register base addresses <SEQn_base>**

| SEQn instance | <SEQn_base> address |
|---|---|
| SEQ0 | FF41 F000$_H$ |

**Clock supply** All Wake-up Sequencers provide one clock input.

**Table 9-4 SEQn clock supply**

| SEQn instance | SEQn clock | Connected to |
|---|---|---|
| SEQ0 | PCLK | Low Speed IntOsc $f_{RL}$ |

**SEQ H/W reset** The Wake-up Sequencers and their registers are initialized by the following reset signal:

**Table 9-5 SEQn reset signal**

| SEQn | Reset signal |
|---|---|
| SEQ0 | • Reset Controller: SYSRES |

### I/O signals

The I/O signals of the Wake-up Sequencers are listed in the following table.

**Table 9-6 SEQn I/O signals**

| SEQn signal | Function | Connected to |
|---|---|---|
| **SEQ0:** | | |
| DPIN0 | Digital input 0 | Port P0_0 |
| DPIN1 | Digital input 1 | Port P0_1 |
| DPIN2 | Digital input 2 | Port P0_2 |
| DPIN3 | Digital input 3 | Port P0_3 |
| DPIN4 | Digital input 4 | Port P0_4 |
| DPIN5 | Digital input 5 | Port P0_5 |
| DPIN6 | Digital input 6 | Port P0_6 |
| DPIN7 | Digital input 7 | Port P0_7 |
| DPIN8 | Digital input 8 | Port P0_8 |
| DPIN9 | Digital input 9 | Port P0_9 |
| DPIN10 | Digital input 10 | Port P0_10 |
| DPIN11 | Digital input 11 | Port P0_11 |
| DPIN12 | Digital input 12 | Port P0_12 |
| DPIN13 | Digital input 13 | Port P0_13 |
| DPO | Digital signal sources activation | Port P0_14 |
| APO | Analog signal sources activation | Port P0_15 |

For using the P0_m ports as Wake-up Sequencer I/O signals, refer to *9.2 “I/O signals port configuration” on page 358* .

### Internal signals

The internal signal connections of the Wake-up Sequencer is shown in the following table.

**Table 9-7 SEQn internal signal connections**

| SEQn signal | Function | Connected to |
|---|---|---|
| **SEQ0:** | | |
| INTTAUJ0I0 | Timer interrupt 0 | TAUJ0 INTTAUJ0I0 |
| INTTAUJ0I1 | Timer interrupt 1 | TAUJ0 INTTAUJ0I1 |
| INTTAUJ0I2 | Timer interrupt 2 | TAUJ0 INTTAUJ0I2 |
| SEQ0WUR | Wake-up request | Stand-by Controller SEQ0WUR |

## 9.2 I/O signals port configuration

The Wake-up Sequencer I/O signals use alternative functions of the respective ports.
Beside the correct setting of the concerned port control registers, the function of some ports require also to enable the Wake-up Sequencer (via the sequencer control register bit SEQnSCTRL.SEQnSEQEN = 1) and to enable the respective DPINm signal to be evaluated by the Wake-up Sequencer (via the sequencer DPIN selection register bits SEQnDPINSR.SEQnDPINSRm = 1).

The table below shows the correct setting of these registers in order to use the respective ports as Wake-up Sequencer I/O signals:

All Wake-up Sequencer ports are using the ports in alternative mode 1.
Thus following setting of the port control registers is common to all Wake-up Sequecner ports:

PMC0_m = 1, PFC0_m = 0, PFCE0_m = 0

**Table 9-8 SEQn port configuration**

| Port | PMn_m setting | SEQnSEQEN | SEQnDPINSRm | Port function |
|---|---|---|---|---|
| P0_0 | PM0_0 = 1 (ALT_IN1) | x | x | DPIN0 |
| P0_1 | PM0_1 = 1 (ALT_IN1) | x | x | DPIN1 |
| P0_2 | PM0_2 = 1 (ALT_IN1) | x | x | DPIN2 |
| P0_3 | PM0_3 = 1 (ALT_IN1) | x | x | DPIN3 |
| P0_4 | PM0_4 = 1 (ALT_IN1) | x | x | DPIN4 |
| P0_5 | PM0_5 = 1 (ALT_IN1) | 1 | 1 | DPIN5 |
| | | else | | FCN0RX |
| P0_6 | PM0_6 = 1 (ALT_IN1) | 1 | 1 | DPIN6 |
| | | else | | FCN1RX |
| P0_7 | PM0_7 = 1 (ALT_IN1) | 1 | 1 | DPIN7 |
| | | else | | URTE11RX |
| P0_8 | PM0_8 = 1 (ALT_IN1) | x | x | DPIN8 |
| P0_9 | PM0_9 = 1 (ALT_IN1) | 1 | 1 | DPIN9 |
| | | else | | URTE10RX |
| P0_10 | PM0_10 = 1 (ALT_IN1) | x | x | DPIN10 |
| P0_11 | PM0_11 = 1 (ALT_IN1) | 1 | 1 | DPIN11 |
| | | else | | URTE11RX |
| P0_12 | PM0_12 = 1 (ALT_IN1) | 1 | 1 | DPIN12 |
| | | else | | TAUJ0I0 |
| P0_13 | PM0_13 = 1 (ALT_IN1) | 1 | 1 | DPIN13 |
| | | else | | TAUJ0I1 |
| P0_14 | PM0_14 = 0 (ALT_OUT1) | 1 | x | DPO |
| | | else | | TAUJ0O2 |
| P0_15 | PM0_15 = 0 (ALT_OUT1) | 1 | x | APO[a] |
| | | else | | TAUJ0O3 |

a) Output of the Analog signal sources activation signal (APO) only in mixed input mode, i.e. if SEQnSCTRL.SEQnSEQMD = 1.

## 9.3 Functional Overview

The Wake-Up Sequencer supports minimization of the standby-modes power consumption in cyclic wake-up applications.

**Features summary**

- periodical check of up to 14 external events
- automatic stand-by mode wake-up upon selectable external events
- automated signals generation for activation of external signal sources
- two operation modes selectable:
  - digital input mode
  - mixed (digital and analog) input mode

The following diagram shows the Wake-up Sequencer and its connections to microcontroller internal and external circuits.

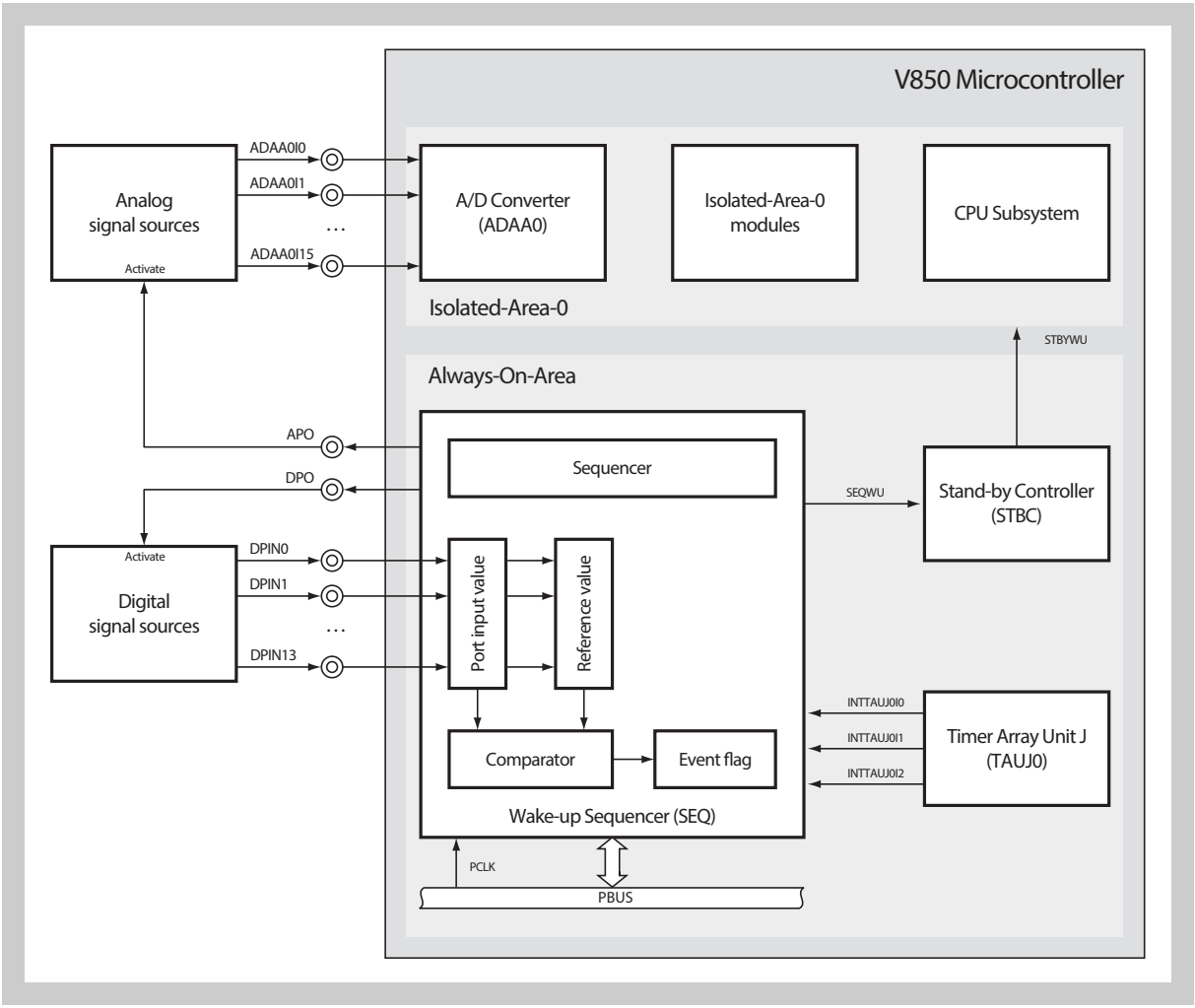



**Figure 9-1 Wake-up Sequencer environment**

## 9.4 Functional description

**Digital input mode**

The Wake-up Sequencer periodically evaluates up to 14 external digital signals and generates a wake-up from DEEPSTOP or STOP stand-by mode, if any of the digital signals has changed its level.

**Mixed input mode**

In mixed input mode the Wake-up Sequencer additionally wakes up the microcontroller to get the CPU and the A/D Converter (ADAA) involved for evaluation of analog input signals.
In this case the Wake-up Sequencer activates the external analog signals sources before the CPU and A/D Converter are in operation. This allows concurrent stabilization of the external analog signals sources and CPU startup, and thus minimization of the power-on time of the CPU and A/D Converter.

**Timing**

All required time instances, as interval and stabilization times, are generated by the Timer Array Unit TAUJ0.
TAUJ0 is operated in a specific mode for generating the Wake-up Sequencer timings. Refer to *9.6 “Wake-up Sequencer specific operation of TAUJ0” on page 381* for a description of this specific mode.

**Power domains**

The Wake-up Sequencer, as well as all other necessary modules, are located on the Always-On-Area, since the Isolated-Area-0 power supply is switched off in DEEPSTOP mode.

### 9.4.1 Mode selection

The mode, the Wake-up Sequencer shall operate in, is selected via a bit of the sequencer control register SEQnSCTLR:

- SEQnSEQMD = 0: digital input mode
- SEQnSEQMD = 0: mixed input mode

### 9.4.2 Starting and stopping the sequencer

The Wake-up Sequencer is started and stopped via a bit of the sequencer control register SEQnSCTLR:

- SEQnSEQEN = 1: the sequencer is started
- SEQnSEQEN = 1: the sequencer is stopped

### 9.4.3 Digital input signals selection

The selection which of the 14 digital input signals shall take part in the evaluation, i.e. the comparison with the reference values, is done via control bits in the DPIN selection register SEQnDPINSR:

- SEQnDmEN = 0: signal DPINm is not enabled for evaluation
- SEQnDmEN = 1: signal DPINm is enabled for evaluation

**DPINm timing requirements**

In order to detect safely any level change of a DPINm signal, the signal must fulfil following timing conditions:

- minimum high level width $t_{DPH}$: 2 $T_{PCLK}$
- minimum low level width $t_{DPL}$: 2 $T_{PCLK}$

$T_{PCLK}$: cycle duration of the PCLK clock

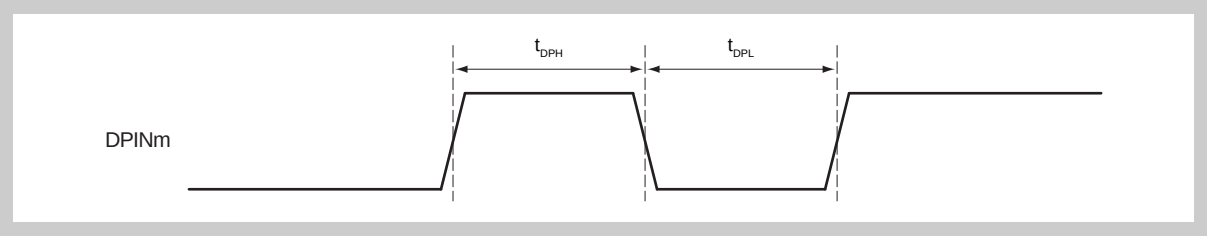



**Figure 9-2 DPINm timing**

### 9.4.4 Event flag

If the comparison between the currently read in digital values and their reference values show a mismatch, the event flag is set in the event flag register SEQnEFR:

- SEQnDPINEVF = 0: current DPINm values equals their reference values
- SEQnDPINEVF = 1: current DPINm values differ from their reference values

Note that the SEQnDPINEVF can be cleared only via the sequencer clear register SEQnSCR.

### 9.4.5 CPU processing acceleration

If the CPU Subsystem is woken up by the Wake-up Sequencer, it operates with following clocks:

- After wake-up from DEEPSTOP
  The CPU clock CKSCLK_000 operates with the
  - High Speed IntOsc clock $f_{RH}$, if it was enabled before DEEPSTOP mode entry (ROSCS.ROSCSCLKEN = 1).
  - Low Speed IntOsc clock $f_{RL}$, if the High Speed IntOsc clock $f_{RH}$ was disabled before DEEPSTOP entry (ROSCS.ROSCSCLKEN = 0).
- After wake-up from STOP
  The CPU clock CKSCLK_000 operates with the clock used before STOP mode entry.

**Note** It is recommended to select the High Speed IntOsc clock as the CPU clock CKSCLK_000 before stand-by entry and to stop the High Speed IntOsc during stand-by unmasking its stop request, i.e. ROSCE.ROSCESTPMSK = 0.

In order to accelerate the CPU processing in the evaluation intervals the High Speed IntOsc frequency of nominal 8 MHz can be set to 13 MHz by setting SEQnSCTRL.SEQnACT13M = 1.

### 9.4.6 Operation clocks

The operation clocks of the involved modules can be selected from different clock sources, whose availability depend on the stand-by mode.

Though the Timer Array Unit TAUJ0, the CPU and the A/D Converter (used in mixed input mode) could be supplied by various clock sources, it is recommended to setup the clock controller as follows before stand-by entry in order to minimize the overall power consumption:

- Stop the MainOsc (MOSCEDISTRG = 1) and the PLLs (PLLEkDISTRG = 1).
- Enable the High Speed IntOsc (ROSCE.ROSCEENTRG = 1) and unmask its stop request for stopping the High Speed IntOsc during stand-by (ROSCE.ROSCESTPMSK = 0).
- Set the High Speed IntOsc frequency $f_{RH}$ to nominal 13 MHz (SEQnSCTRL.SEQnACT13M = 1) for minimizing the CPU Subsystem and A/D Converters operation time after wake-up.
- Select the High Speed IntOsc clock via the respective clock selector register as the
  - CPU clock CKSCLK_000 (CKSCID_000 = $003A_H$),
  - A/D Converter clock CKSCLK_012 (CSKCID_012 = $0007_H$).
- Select the Low Speed IntOsc clock as the TAUJ0 clock CKSCLK_A03 (CKSC_A03.CKSCID_A03 = $0001_H$).
  Since the Timer Array Unit TAUJ0 is used during stand-by mode, mask the stop request of the TAUJ0 PCLK clock CKSCLK_A03.STPMSK_A03 = 1.

**Cautions**

1. If the High Speed IntOsc nominal frequency is set to 13 MHz (SEQnSCTRL.SEQnACT13M = 1), it must not be selected as the clock source for the TAUJ0 (CKSC_A03.CKSCID_A03 $\neq 0007_H$).
2. If the High Speed IntOsc nominal frequency is set to 13 MHz (SEQnSCTRL.SEQnACT13M = 1), it must be stopped in DEEPSTOP mode (ROSCE.ROSCESTPMSK = 0).
3. The High Speed IntOsc nominal frequency must be set back to 8 MHz (SEQnSCTRL.SEQnACT13M = 0), when normal operation of the device is resumed.

### 9.4.7 Wake-up Sequencer setup

Prior starting the Wake-up Sequencer and entering stand-by, a reference value of the digital input signals has to be acquired. This reference value is used for comparison by the sequencer later on.

Several control bits in the sequencer control register SEQnSCTLR are used to generate the reference values by software:

- SEQnDPOSET = 1 sets the DPO output signal to high level, thus activates the external digital signals sources.
- SEQnAPOSET = 1 sets the APO output signal to high level, thus activates the external analog signals sources.
- SEQnDPITRG = 1 triggers the read in of the reference values from the enabled (SEQnSCTLR.SEQnDPmEN = 1) digital input signal ports.

Note that the SEQnDPOSET and SEQnAPOSET can be cleared only via the sequencer clear register SEQnSCR.

**Analog references**

If the Wake-up Sequencer shall operate in mixed input mode during DEEPSTOP stand-by mode and the analog input signals shall be compared against analog reference values, the respective analog signals are converted by use of the A/D Converter and these reference values must be stored in the Back-up RAM or data flash memory.
Note that the CPU RAM content is undefined after wake-up due to Isolated-Area-0 power off in DEEPSTOP mode.

**Clocks setup**

Refer to the section *9.4.6 “Operation clocks” on page 362* .

**DPINm port configuration**

For using ports as Wake-up Sequencer I/O signals, refer to the section *“I/O signals port configuration”* above in this chapter.

**TAUJ0 initialization**

TAUJ0 is operated in a specific mode for generating the Wake-up Sequencer timings. Refer to *9.6 “Wake-up Sequencer specific operation of TAUJ0” on page 381* for a description of this specific mode.

The figure below shows a recommended setup flow.

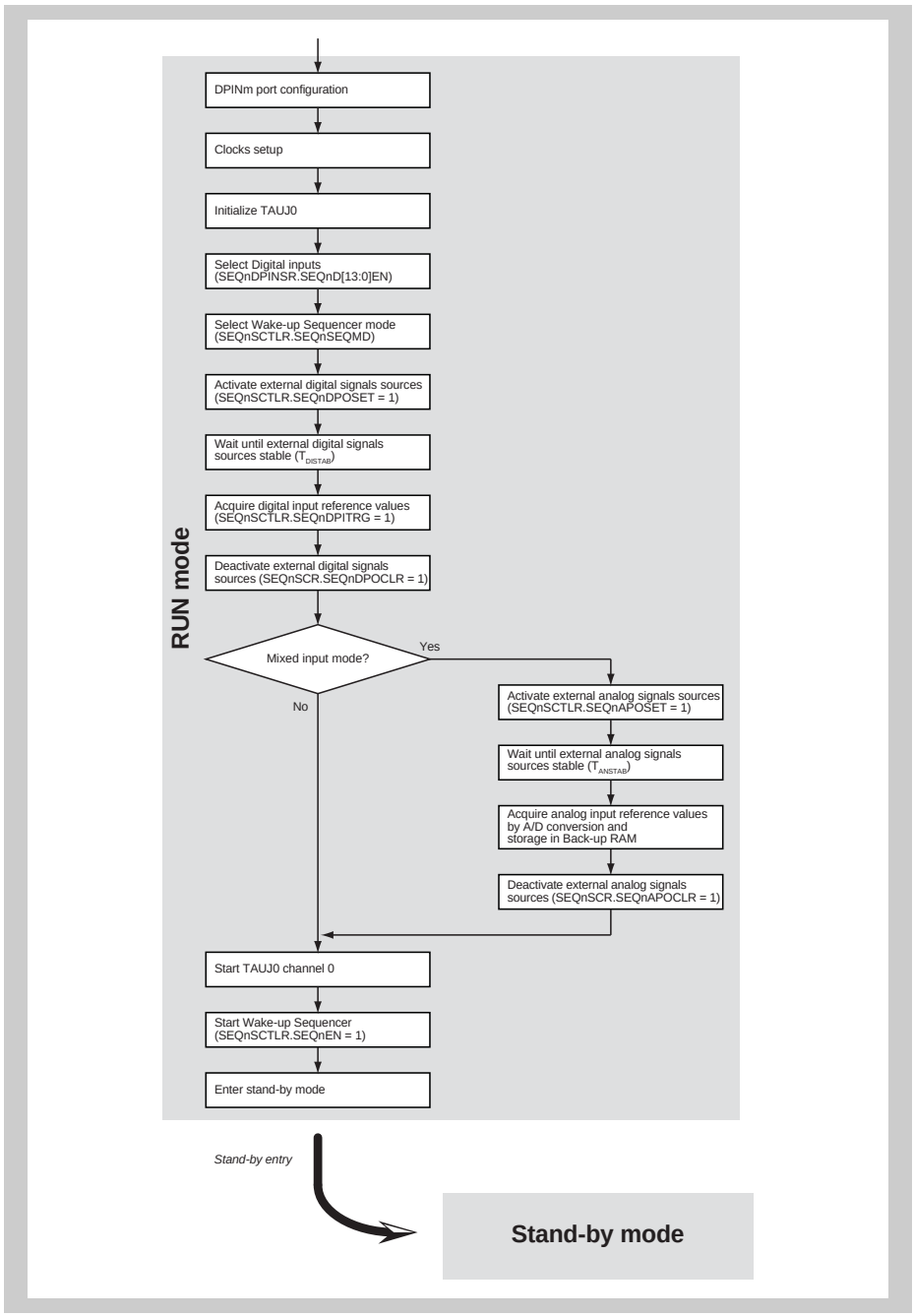



**Figure 9-3 Wake-up Sequencer setup**

## 9.4.8 Digital input mode

### (1) Basic operation in digital input mode

In digital input mode (SEQnSCTLR.SEQMD = 0), the Wake-up Sequencer compares at regular intervals the state of the selected digital inputs DPINm with their reference values, which were stored during the sequencer setup.

Periodically with the cycle duration $T_{EVAL}$ the Wake-up Sequencer

- sets the output signal DPO to high level, that can be used to activate the sources e.g. sensors or switches) of the digital inputs signals to check
- waits a certain time $T_{DISTAB}$ to stabilize the external digital input sources
- reads in the digital signals
- performs a compare of the current signals against their reference values
  - If at least one currently read digital input value does not match its reference value, the event flag SEQnEFR.SEQnDPINEVF is set and the stand-by wake-up request STBYWU is asserted towards the Stand-by Controller, which wakes up the Isolated-Area-0. After the stand-by wake-up the CPU is taking over the control.
  - If the currently read digital input values match the reference values, the output signal DPO is reset to low level (for deactivating the digital input signal sources) and the Wake-up Sequencer waits for the next evaluation period.

The following figure shows the basic operation of the Wake-up Sequencer in digital input mode:

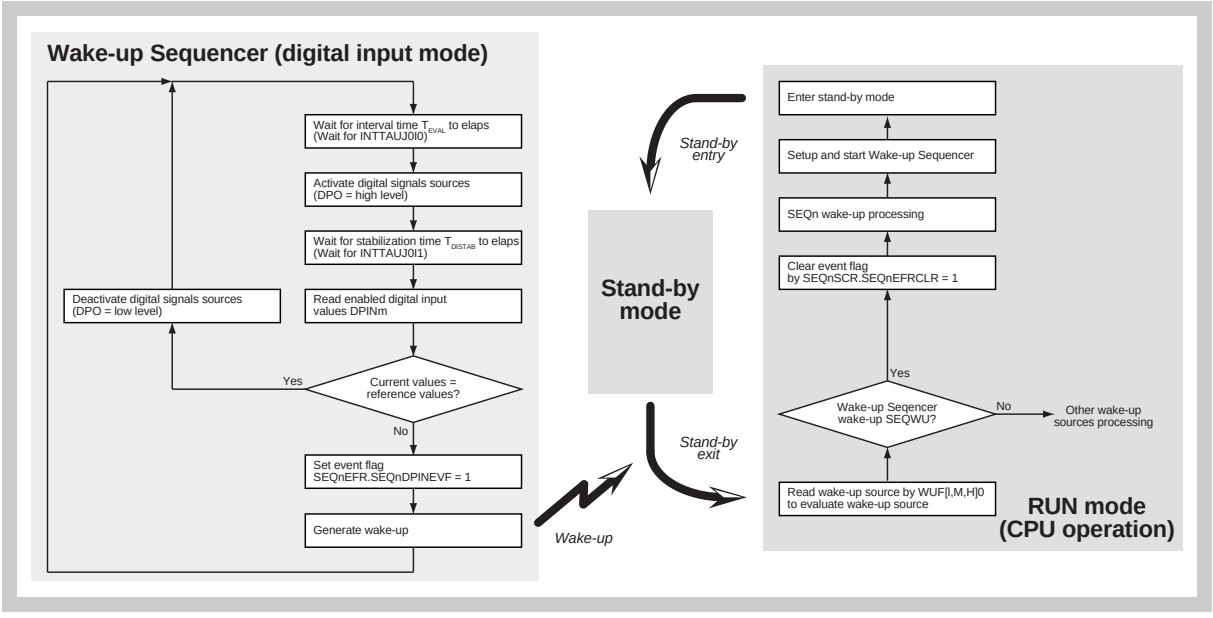



**Figure 9-4 Digital input mode flow**

### (2) CPU operation

After wake-up, the CPU is able to detect the wake-up source by evaluating the wake-up factor registers WUF[L,M,H]0 and the event flag SEQnEFR.SEQnDPINEVF to identify a change of the state of the digital inputs as wake-up source.
The CPU then deactivates the external sources of the digital input signals by

setting SEQnSCR.SEQnDPOCLR = 1 and clears the event flag by SEQnSCR.SEQnEFRCLR = 1.

For details about wake-up factors refer to the chapter *"Stand-by Controller"*.

**(3) Timing**

All required timing instances are realized by Timer Array Unit TAUJ0 interrupts.

TAUJ0 is operated in a specific mode for generating the Wake-up Sequencer timings. Refer to *9.6 "Wake-up Sequencer specific operation of TAUJ0" on page 381* for a description of this specific mode.

The time $T_{CKx}$ in the formulas below refer to cycle duration of one of the CK0 to CK3 clocks, selected by TAUJ0 clock selector.

**Interval time**

The interval time for the cyclic start of the evaluation of the input signals is determined by the Timer Array Unit TAUJ0 channel 0, that generates periodically its interrupt INTTAUJ0I0 with the cycle duration $T_{EVAL}$:

$$T_{EVAL} = T_{CKx} \cdot (TAUJ0CDR0 + 1)$$

**Digital stabilization time**

The digital signal sources stabilization time $T_{DISTAB}$ is counted by the TAUJ0 channel 1, that generates its interrupt INTTAUJ0I1, when the stabilization time has elapsed.

$$T_{DISTAB} = T_{CKx} \cdot (TAUJ0CDR1 + 1)$$

**Master/slave channels**

The interval timer TAUJ0 channels 0 acts as the master channel, while channel 1 is configured as a slave channel.

**(4) Digital input mode: digital input signals DPINm do not change**

The diagram below depicts the situation, where none of the enabled (SEQnDPINSR.DmEN = 1) digital input signals DPINm has changed its level after the reference values were stored during the setup phase.

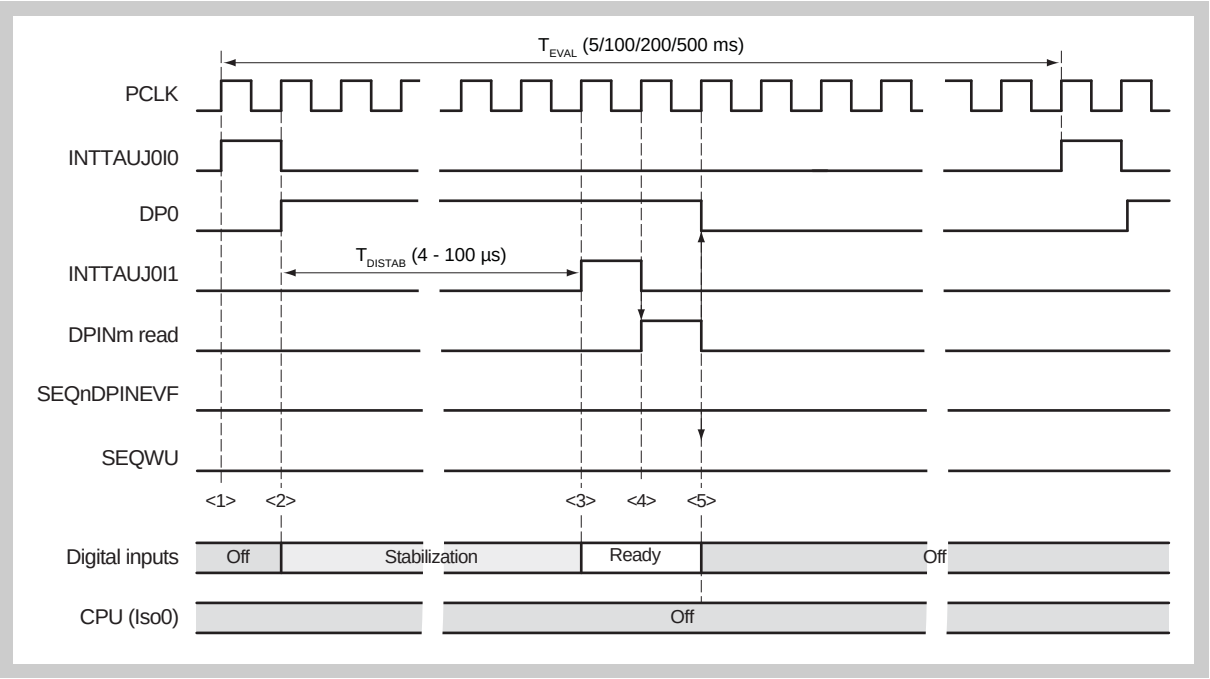



**Figure 9-5 Wake-up Sequencer operation in digital input mode - no DPINm change**

**<1>** After the configured interval time $T_{EVAL}$ has passed, channel 0 of TAUJ0 generates its interrupt request INTTAUJ0I0.

**<2>** The DPO output is set to high level in order to activate the external digital input signals DPINm sources.
Channel 1 of TAUJ0 starts counting down the digital input sources stabilization time $T_{DISTAB}$ to ensure the digital input signals to settle.

**<3>** $T_{DISTAB}$ has elapsed and the digital inputs signals DPINm are stable.

**<4>** The digital input signals DPINm are read and the read values are compared against their reference values.

**<5>** Since all read values and their reference values match, the Wake-up Sequencer resets the DPO output signal to low level (deactivation of external digital input sources) and waits for the next INTTAUJ0I0 interrupt, that starts the next evaluation.

**(5) Digital input mode: digital input signals DPINm change**

The diagram below depicts the situation, where at least one of the enabled (SEQnDPINSR.DmEN = 1) digital input signals DPINm has changed its level after the reference values were stored during the setup phase.

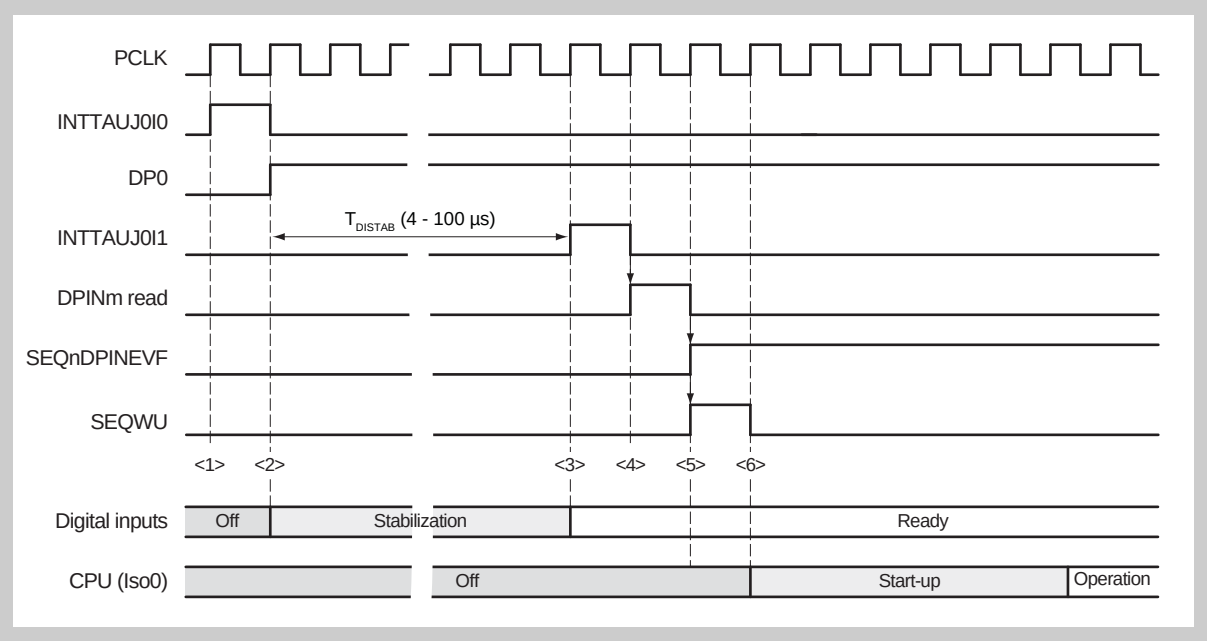



**Figure 9-6 Wake-up Sequencer operation in digital input mode - DPINm change**

**<1>** After the configured interval time $T_{EVAL}$ has passed, channel 0 of TAUJ0 generates its interrupt request INTTAUJ0I0.

**<2>** The DPO output is set to high level in order to activate the external digital input signals DPINm sources.
Channel 1 of TAUJ0 starts counting down the digital input sources stabilization time $T_{DISTAB}$ to ensure the digital input signals to settle.

**<3>** $T_{DISTAB}$ has elapsed and the digital inputs signals DPINm are stable.

**<4>** The digital input signals DPINm are read and the read values are compared against their reference values.

**<5>** Since at least one read value and its reference value does not match, the evaluation flag SEQnEFR.SEQnDPINEVF is set and the Isolated-Area-0 wake-up signal SEQWU is generated towards the Stand-by Controller.

**<6>** The Isolated-Area-0 is woken up and the CPU starts operation.

### 9.4.9 Mixed input mode

#### (1) Basic operation in mixed input mode

In mixed input mode (SEQnSCTLR.SEQMD = 0), the Wake-up Sequencer compares the state of the selected digital inputs DPINm with their reference at regular intervals, as in digital input mode.
Additionally the microcontroller is also prepared to evaluate the level of analog signals, acquired via an A/D conversion, with a minimum of power-on time, thus minimizing the overall power consumption.

Periodically with the cycle duration $T_{EVAL}$ the Wake-up Sequencer

- generates the stand-by mode wake-up request SEQWU towards the Stand-by Controller, which wakes up the Isolated-Area-0 and starts the CPU Subsystem,
- sets the output signals DPO and APO to high level, which can be used to activate the sources of the digital and analog inputs signals to evaluate,
- waits a certain time $T_{ADJUST}$ to adjust the startup time of the CPU Subsystem on Isolated-Area-0 with the stabilization time of the external analog input sources,
- waits a certain time $T_{DISTAB}$ to stabilize the external digital input sources,
- reads in the digital input signals
- performs a compare of the current digital signals against their reference values
  - If the currently read digital input values do not match the reference values, the event flag SEQnEFR.SEQnDPINEVF is set.
  - If the currently read digital input values match the reference values, the event flag SEQnEFR.SEQnDPINEVF remains reset.
- resets the output signal DPO to low level (for deactivating the digital input signal sources)

The following figure shows the basic operation of the Wake-up Sequencer in mixed input mode:

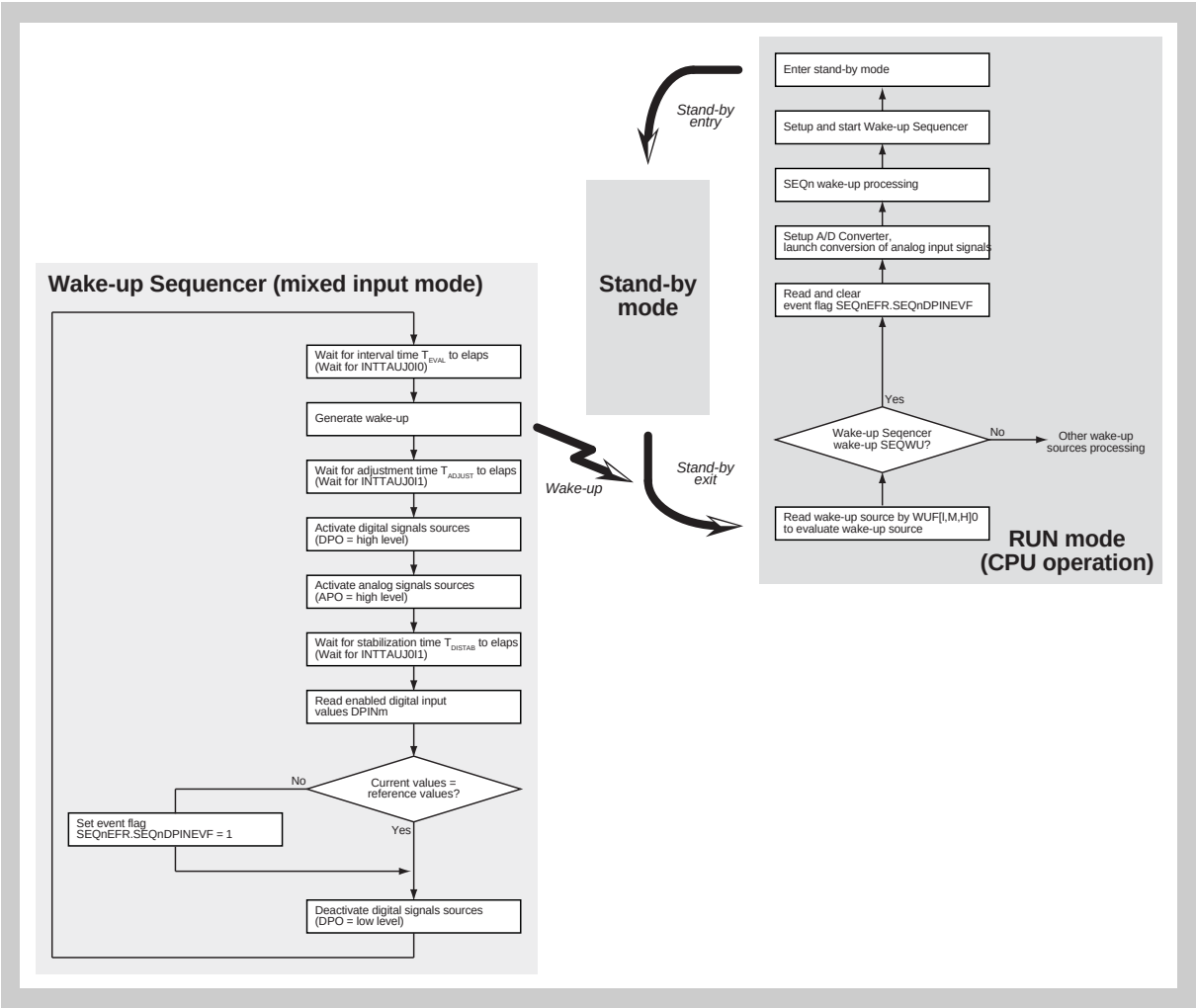



**Figure 9-7 Mixed input mode flow**

### (2) CPU operation

After wake-up, the CPU is able to detect the Wake-up Sequencer as the wake-up source by evaluating the wake-up factor registers WUF[L,M,H]0. For details about wake-up factors refer to the chapter *"Stand-by Controller"*.

The event flag SEQnEFR.SEQnDPINEVF = 1 notifies a change of the state of the digital inputs DPINm.

For evaluation of analog input signals the CPU configures A/D Converter (ADAA) and triggers conversion of the analog signals.

**Note** To reduce processing time for evaluating the analog inputs values, the *"Conversion result upper/lower limit compare function"* of A/D Converter can be used. This will automatically generate an interrupt if the conversion result is not within certain limits. The limits applied could be analog values, taken just before entering stand-by.
Refer to the section *"Result check functions"* in chapter *"A/D Converter (ADAA)"* for details.

After evaluation of the analog conversion result, the CPU can deactivate the analog input signals sources by setting APO to low level by SEQnSCR.APOCLR = 1.

If there was a change in the digital inputs during this interval, the CPU needs to clear the event flag as well by SEQnSCR.EFRCLR = 1.

**(3) Timing**

All required time instances, as interval, adjustment and stabilization times, are generated by the Timer Array Unit TAUJ0.

TAUJ0 is operated in a specific mode for generating the Wake-up Sequencer timings. Refer to *9.6 "Wake-up Sequencer specific operation of TAUJ0" on page 381* for a description of this specific mode.

The time $T_{CKx}$ in the formulas below refer to cycle duration of one of the CK0 to CK3 clocks, selected by TAUJ0 clock selector.

**Interval time**

The interval time for the cyclic start of the evaluation of the input signals is determined by the Timer Array Unit TAUJ0 channel 0, that generates periodically its interrupt INTTAUJ0I0 with the cycle duration $T_{EVAL}$:

$$T_{EVAL} = T_{CKx} \cdot (TAUJ0CDR0 + 1)$$

**Analog stabilization and CPU start-up time**

The analog signal sources stabilization and CPU startup time adjustment $T_{ADJUST}$ is set by the TAUJ0 channel 1 interrupt INTTAUJI1.

$$T_{ADJUST} = T_{CKx} \cdot (TAUJ0CDR1 + 1)$$

The adjustment time is calculated as follows:

$$T_{ADJUST} = T_{CPUSU} - T_{ANSTAB}$$

| | |
|---|---|
| $T_{CPUSU}$: | CPU startup time, i.e. the time from Isolated-Area-0 wake-up to the first CPU instruction fetch |
| $T_{ANSTAB}$: | Stabilization time of the analog signals sources |

For the value of CPU start-up time $T_{CPUSU}$ refer to the Data Sheet.

**Digital stabilization time**

The digital signal sources stabilization time $T_{DISTAB}$ is counted by the TAUJ0 channel 2, that generates its interrupt INTTAUJ0I2, when the stabilization time has elapsed.

$$T_{DISTAB} = T_{CKx} \cdot (TAUJ0CDR2 + 1)$$

**Master/slave channels**

The interval timer TAUJ0 channels 0 acts as the master channel, while channels 1 and 2 are configured as slave channels.

**(4) Mixed input mode: digital input signals DPINm do not change**

The diagram below depicts the situation, where none of the enabled (SEQnDPINSR.DmEN = 1) digital input signals DPINm has changed its level after the reference values were stored during the setup phase.

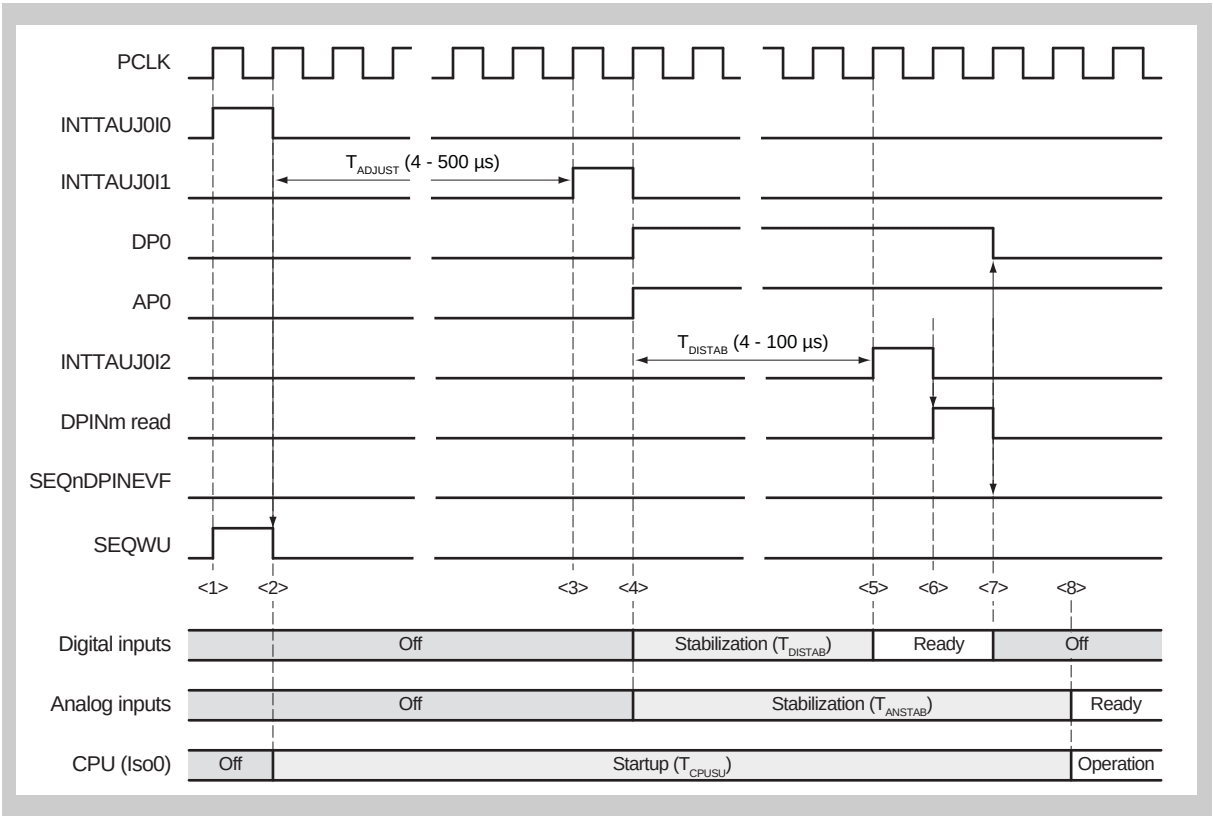



**Figure 9-8 Wake-up Sequencer operation in mixed input mode - no DPINm change**

**<1>** After the configured interval time $T_{EVAL}$ has passed, channel 0 of TAUJ0 generates its interrupt request INTTAUJ0I0.

**<2>** The Isolated-Area-0 wake-up signal SEQWU is generated towards the Stand-by Controller, which launches Isolated-Area-0 wake-up, and thus the startup of the CPU.
Channel 1 of TAUJ0 starts counting down the adjustment time $T_{ADJUST}$ to adjust the wake-up time of the CPU Subsystem on Isolated-Area-0 with the stabilization time of the external analog input sources.

**<3>** The adjustment time $T_{ADJUST}$ has elapsed.

**<4>** The DPO and APO outputs are set to high level in order to activate the external digital and analog input signals sources.
Stabilization of the analog input signals sources and CPU startup are running concurrently.
Channel 1 of TAUJ0 starts counting down the digital input sources stabilization time $T_{DISTAB}$ to ensure the digital input signals to settle.

**<5>** $T_{DISTAB}$ has elapsed and the digital inputs signals DPINm are stable.

**<6>** The digital input signals DPINm are read and the read values are compared against their reference values.

**<7>** Since all read values and their reference values match, the event flag SEQnDPINEVF remains reset.

The Wake-up Sequencer resets the DPO output signal to low level (deactivation of external digital input sources).

<8> The CPU starts operation and the analog inputs signals are stable.

**(5) Mixed input mode: digital input signals DPINm change**

The diagram below depicts the situation, where at least one of the enabled (SEQnDPINSR.DmEN = 1) digital input signals DPINm has changed its level after the reference values were stored during the setup phase.

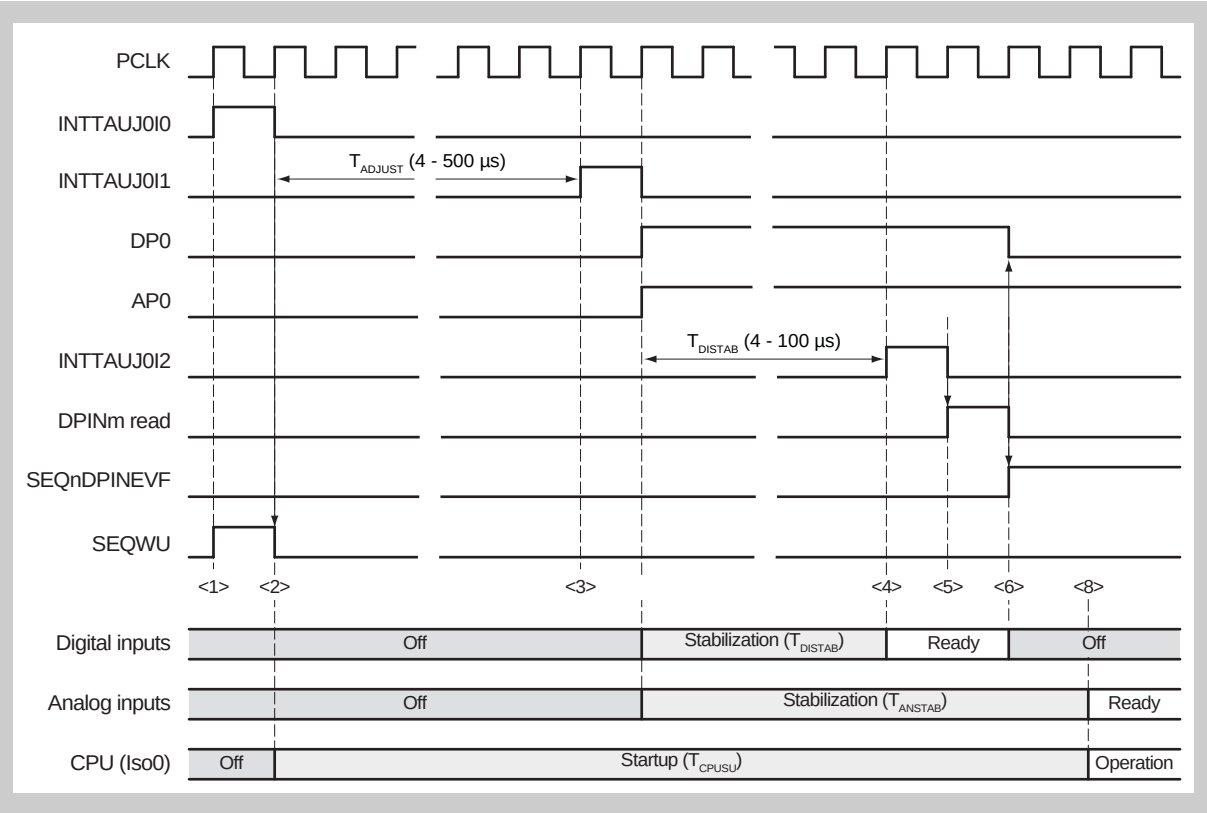



**Figure 9-9 Wake-up Sequencer operation in mixed input mode - DPINm change**

**<1>** After the configured interval time $T_{EVAL}$ has passed, channel 0 of TAUJ0 generates its interrupt request INTTAUJ0I0.

**<2>** The Isolated-Area-0 wake-up signal SEQWU is generated towards the Stand-by Controller, which launches Isolated-Area-0 wake-up, and thus the startup of the CPU.
Channel 1 of TAUJ0 starts counting down the adjustment time $T_{ADJUST}$ to adjust the wake-up time of the CPU Subsystem on Isolated-Area-0 with the stabilization time of the external analog input sources.

**<3>** The adjustment time $T_{ADJUST}$ has elapsed.

**<4>** The DPO and APO outputs are set to high level in order to activate the external digital and analog input signals sources.
Stabilization of the analog input signals sources and CPU startup are running concurrently.
Channel 1 of TAUJ0 starts counting down the digital input sources stabilization time $T_{DISTAB}$ to ensure the digital input signals to settle.

**<5>** $T_{DISTAB}$ has elapsed and the digital inputs signals DPINm are stable.

**<6>** The digital input signals DPINm are read and the read values are compared against their reference values.

**<7>** Since at least one read value and its reference value does not match, the evaluation flag SEQnEFR.SEQnDPINEVF is set.

The Wake-up Sequencer resets the DPO output signal to low level (deactivation of external digital input sources).

<8> The CPU starts operation and the analog inputs signals are stable.

## 9.5 Wake-up Sequencer registers

This section contains a description of all registers of the Wake-up Sequencer.

### 9.5.1 Wake-up Sequencer registers overview

The Wake-up Sequencer is controlled and operated by the following registers:

**Table 9-9 Wake-up Sequencer registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| DPINm selection register | SEQnDPINSR | <SEQn_base> |
| Control register | SEQnSCTLR | <SEQn_base> + $4_H$ |
| Event flag register | SEQnEFR | <SEQn_base> + $8_H$ |
| Clear register | SEQnSCR | <SEQn_base> + $C_H$ |

**<SEQn_base>** The base addresses <SEQn_base> of the SEQn is defined in the first section of this chapter under the key word “Register addresses”.

### 9.5.2 Wake-up Sequencer registers details

#### (1) SEQnDPINSR - DPIN selection register

This register selects the digital inputs signals that will be evaluated by the Wake-up Sequencer.

**Access** This register can be read/written in16-bit units.

**Address** <SEQn_base>

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | SEQn D13EN | SEQn D12EN | SEQn D11EN | SEQn D10EN | SEQn D9EN | SEQn D8EN | SEQn D7EN | SEQn D6EN | SEQn D5EN | SEQn D4EN | SEQn D3EN | SEQn D2EN | SEQn D1EN | SEQn D0EN |
| R | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 9-10 SEQnDPINSR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 13 to 0 | SEQnDmEN | Enable of related digital inputs DPINm<br>0: DPINm is not enabled for evaluation<br>1: DPINm is enabled for evaluation |

### (2) SEQnSCTLR - Control register

This register contains various control bits.

**Access** This register can be read/written in 16-bit units.

**Address** <SEQn_base> + $4_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SEQn APOSET | SEQn DPOSET | 0 | 0 | 0 | 0 | 0 | 0 | SEQn ACT13M | 0 | 0 | 0 | 0 | SEQn DPITRG | SEQn SEQMD | SEQn SEQEN |
| W | W | R | R | R | R | R | R | R/W | R | R | R | R | W | R/W | R/W |

**Table 9-11 SEQnSCTLR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | SEQn APOSET | Set the output signal APO to high level for activating the analog input signal sources<br>0: no function<br>1: sets APO to high level<br>This bit can only be cleared via the clear register SEQnSCR.SEQnAPOCLR = 1.<br>Reading of this bit returns always 0. |
| 14 | SEQn DPOSET | Set the output signal DPO to high level for activating the digital input signal sources<br>0: no function<br>1: sets DPO to high level<br>This bit can only be cleared via the clear register SEQnSCR.SEQnDPOCLR = 1.<br>Reading of this bit returns always 0. |
| 7 | SEQn ACT13M | Change nominal frequency of the High Speed IntOsc<br>0: High Speed IntOsc runs at nominal frequency $f_{RH}$ = 8 MHz<br>1: High Speed IntOsc runs at nominal frequency $f_{RH}$ = 13 MHz |
| 2 | SEQn DPITRG | Trigger acquisition of digital reference values<br>0: no function<br>1: current state of enabled digital input signals DPINm is stored as the reference value<br>Reading of this bit returns always 0. |
| 1 | SEQn SEQMD | Wake-up sequencer mode selection<br>0: digital input mode<br>1: mixed input mode |
| 0 | SEQn SEQEN | General enable of the Wake-up Sequencer<br>0: sequencer operation disabled<br>1: sequencer operation enabled |

### (3) SEQnEFR - Event flag register

This register indicates the result of the last comparison between the latest read in values of the digital input signals and their reference values.

**Access** This register can be read in 16-bit units.

**Address** <SEQn_base> + $8_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SEQn DPIN EVF |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 9-12 SEQnEFR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | SEQn DPINEVF | Result of current and reference values comparison:<br>0: current values match reference values, i.e. no DPINm change detected<br>1: current values do not match reference values, i.e. at least one DPINm change detected<br>This bit can only be cleared via the clear register SEQnSCR.SEQnEFRCLR = 1. |

### (4) SEQnSCR - Clear register

This register clears the active state of the SEQnAPOSET and SEQnDPOSET bits of the SEQnSCTLR register and the SEQnDPINEVF of the SEQnEFR register.

**Access** This register can be written in 16-bit units.

**Address** <SEQn_base> + $C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SEQn APO CLR | SEQn DPO CLR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SEQn EFR CLR |
| W | W | R | R | R | R | R | R | R | R | R | R | R | R | R | W |

**Table 9-13 SEQnSCR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | SEQn APOCLR | Clear of SEQnSCTRL.SEQnAPOSET<br>0: no function<br>1: clear SEQnSCTRL.SEQnAPOSET, i.e. APO output signal is set to low level |
| 14 | SEQn DPOCLR | Clear of SEQnSCTRL.SEQnDPOSET<br>0: no function<br>1: clear SEQnSCTRL.SEQnDPOSET, i.e. DPO output signal is set to low level |
| 0 | SEQn EFRCLR | Clear of SEQnEFR.SEQnDPINEVF<br>0: no function<br>1: SEQnEFR.DPINEVF event flag is cleared |

## 9.6 Wake-up Sequencer specific operation of TAUJ0

This section describes the operation modes and setup of the Timer Array Unit TAUJ0, as needed to interact with the Wake-up Sequencer.

### 9.6.1 TAUJ0 operation overview

The Wake-up Sequencer is used in combination with TAUJ0 to generate the required timings for interval time and stabilization times of external input signals.
For a general description of TAUJ refer to chapter *"Timer Array Unit J (TAUJ)"*.

**TAUJ0 operation clock**

Though the TAUJ0 can be supplied by various clock sources, it is recommend to select the Low Speed IntOsc as its clock source.
Thus throughout this chapter it is assumed that TAUJ0 is supplied with the Low Speed IntOsc clock $f_{RL}$ during RUN and STOP/DEEPSTOP stand-by modes.

**(1) Digital input mode**

The following figure shows the general timing diagram of TAUJ0 used for the Wake-up Sequencer in digital input mode:

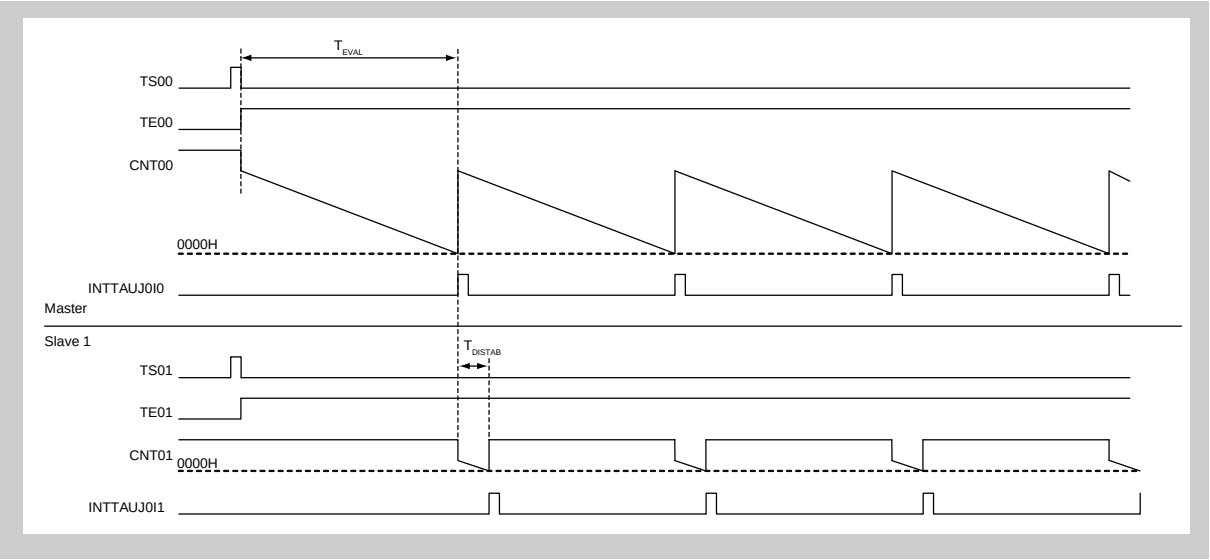



**Figure 9-10 Timing diagram of TAUJ0 in SEQn digital input mode**

Channel 0 operates as the master channel in interval timer mode and counts the interval time $T_{EVAL}$ of the wake-up cycle.

When the master channel counter reaches 0000 $0000_H$, it asserts its interrupt INTTAUJ0I0 and generates an internal signal which in turn starts the count operation of slave channel 1 and reloads the channel 0 counter value.

Slave channel 1 operates in One Count Mode and counts the stabilization time $T_{DISTAB}$ needed by the external digital input signals to settle. The required value is depending on the electrical characteristics of the application circuitry.

When slave channel 1 counter reaches 0000 $0000_H$, it asserts its interrupt INTTAUJ0I1, that is used to trigger the read in of the digital input signals DPINm and the comparison of the current state of the selected digital input signals with their reference value.

**Note** The typical minimum stabilization time that can be configured is 1 count of TAUJ0 channel 1:

$$\text{min. } T_{DISTAB} = 1 / f_{RL} = 4.2\ \mu s$$

with nominal $f_{RL}$ = 240 kHz.
For characteristics of Low Speed IntOsc refer to the Data Sheet.

### (2) Mixed input mode

The following figure shows the general timing diagram of TAUJ0 used for the Wake-up Sequencer in mixed input mode:

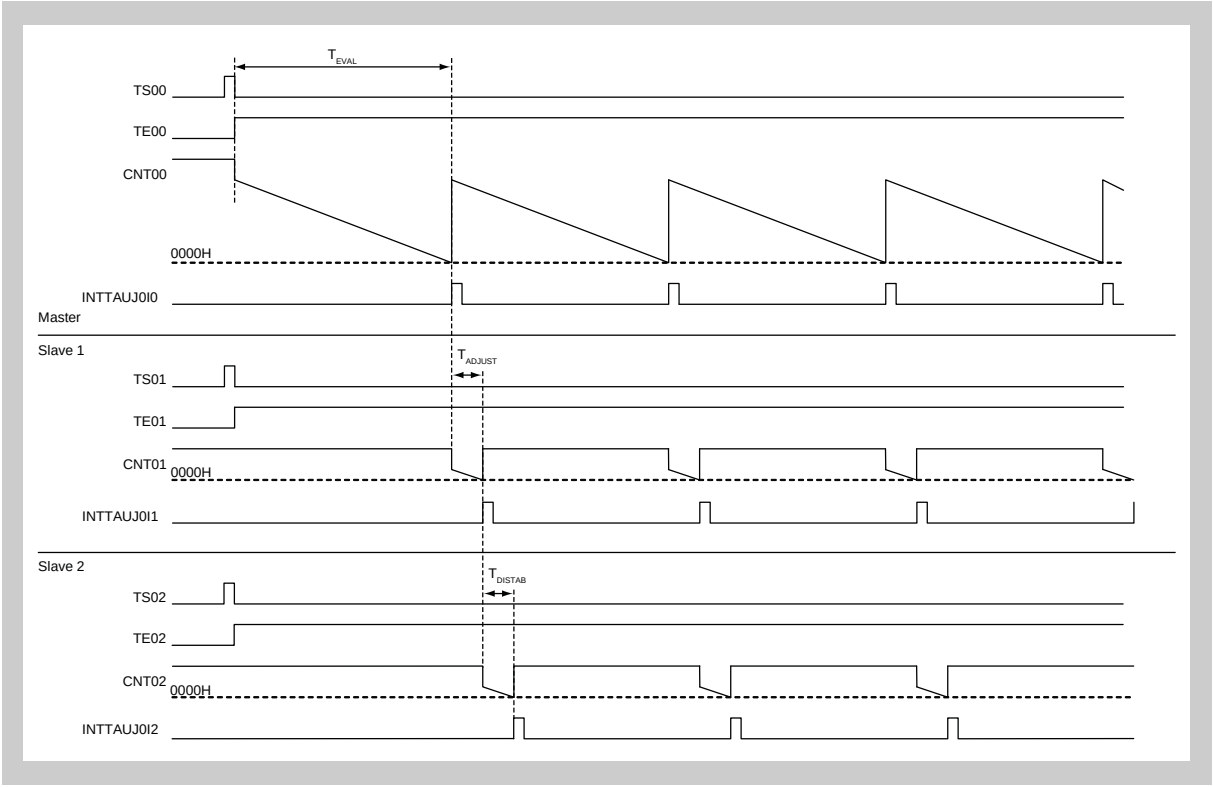



**Figure 9-11 Timing diagram of TAUJ0 in SEQn mixed input mode**

Channel 0 operates as the master channel in interval timer mode and counts the interval time $T_{EVAL}$ of the wake-up cycle.

When the master channel counter reaches 0000 $0000_H$, it asserts its interrupt INTTAUJ0I0 and generates an internal signal which in turn starts the count operation of slave channel 1 and reloads the channel 0 counter value.

Slave channel 1 operates in One Count Mode and counts the adjustment time $T_{ADJUST}$, i.e. the difference between the CPU startup time and the stabilization time of the external analog signals sources.

The adjustment time is calculated as follows:

$$T_{ADJUST} = T_{CPUSU} - T_{ANSTAB}$$

| | |
|---|---|
| $T_{CPUSU}$: | CPU startup time, i.e. the time from Isolated-Area-0 wake-up to the first CPU instruction fetch |
| $T_{ANSTAB}$: | Stabilization time of the analog signals sources |

For the value of CPU start-up time $T_{CPUSU}$ refer to the Data Sheet.
$T_{ANSTAB}$ is depending on the electrical characteristics of the application circuitry.

When the slave channel 1 counter reaches 0000 $0000_H$ it asserts its interrupt INTTAUJ0I1 and generates an internal that starts the count operation of slave channel 2.

Slave channel 2 operates in One Count Mode and counts the stabilization time $T_{DISTAB}$ needed by the external digital input signals to settle. The required value is depending on the electrical characteristics of the application circuitry.

When slave channel 2 counter reaches 0000 $0000_H$, it asserts its interrupt INTTAUJ0I2, that is used to trigger the read in of the digital input signals DPINm and the comparison of the current state of the selected digital input signals with their reference value.

**Note** The typical minimum stabilization time that can be configured is 1 count of TAUJ0 channel 2:

$$\text{min. } T_{DISTAB} = 1 / f_{RL} = 4.2\ \mu s$$

with nominal $f_{RL}$ = 240 kHz.
For characteristics of Low Speed IntOsc refer to the Data Sheet.

### 9.6.2 TAUJ0 register settings

In the following, only the register settings to support SEQn operation will be described for TAUJ0.

Only these specific register settings are allowed to be used to operate TAUJ0 in conjunction with the Wake-up Sequencer SEQn.

**(1) Register settings for the master channel 0**

**(a) TAUJ0CMOR0 for the master channel 0**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

Table 9-14 TAUJ0CMOR0 settings for the master channel of the PWM Output Function

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 1: Channel is master channel |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | Not used, so set to 00. |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: Does not generate INTTAUJ0I0 at operation start |

**(b) TAUJ0CMUR0 for the master channel**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

Table 9-15 TAUJ0CMUR0 settings for the master channel of the PWM Output Function

| Bit name | Setting |
|---|---|
| TIS[1:0] | Not used, so set to 00. |

**(c) Channel output mode for the master channel**

The channel output mode is not used by this function. However, it can be used by other functions or in Independent Channel Output Mode Controlled by Software.

**(d) Simultaneous rewrite for the master channel**

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 9-16 Simultaneous rewrite settings for the master channel**

| Bit name | Setting |
|---|---|
| TAUJ0RDE.RDE00 | 0: Disables simultaneous rewrite |
| TAUJ0RDM.RDM00 | Not used, so set to 0. |

### (2) Register settings for the slave channel 1

#### (a) TAUJ0CMOR1 for the slave channel 1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

Table 9-17 TAUJ0CMOR1 settings for the slave channel 1

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 100: INTTAUJ0I0 of the master channel is the start trigger |
| COS[1:0] | Not used, so set to 00. |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 1: Generates INTTAUJ0I1 |

#### (b) TAUJ0CMUR1 for the slave channel 1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

Table 9-18 TAUJ0CMUR1 settings for the slave channel 1

| Bit name | Setting |
|---|---|
| TIS[1:0] | Not used, so set to 00. |

#### (c) Channel output mode for the slave channel 1

Table 9-19 Control bit settings for Independent Channel Output Mode 1

| Bit name | Setting |
|---|---|
| TAUJ0TOE.TOE01 | 0: Disables Independent Channel Output |
| TAUJ0TOM.TOM01 | Not used, so set to 0. |
| TAUJ0TOC.TOC01 | Not used, so set to 0. |
| TAUJ0TOL.TOL01 | Not used, so set to 0. |

**(d) Simultaneous rewrite for the slave channel 1**

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 9-20 Simultaneous rewrite settings for the slave channel 1**

| Bit name | Setting |
|---|---|
| TAUJ0RDE.RDE01 | 0: Disables simultaneous rewrite |
| TAUJ0RDM.RDM01 | Not used, so set to 0. |

### (3) Register settings for the slave channel 2

#### (a) TAUJ0CMOR2 for the slave channel 2

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

Table 9-21 TAUJ0CMOR2 settings for the slave channel 2

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 101: INTTAUJ0I1 of the upper channel is the start trigger |
| COS[1:0] | Not used, so set to 00. |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 1: Generates INTTAUJ0I2 |

#### (b) TAUJ0CMUR2 for the slave channel 2

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

Table 9-22 TAUJ0CMUR2 settings for the slave channel 2

| Bit name | Setting |
|---|---|
| TIS[1:0] | Not used, so set to 00. |

#### (c) Channel output mode for the slave channel 2

Table 9-23 Control bit settings for Independent Channel Output Mode 1

| Bit name | Setting |
|---|---|
| TAUJ0TOE.TOE02 | 0: Disables Independent Channel Output |
| TAUJ0TO02.TO0202 | Not used, so set to 0. |
| TAUJ0TOC.TOC02 | Not used, so set to 0. |
| TAUJ0TOL.TOL02 | Not used, so set to 0. |

**(d) Simultaneous rewrite for the slave channel 2**

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 9-24 Simultaneous rewrite settings for the slave channel 2**

| Bit name | Setting |
|---|---|
| TAUJ0RDE.RDE02 | 0: Disables simultaneous rewrite |
| TAUJ0RDM.RDM02 | Not used, so set to 0. |

# Chapter 10 Code Protection and Security

## 10.1 Overview

The microcontroller supports various methods for protecting the program code in the flash memory from undesired access, such as illegal read-out or illegal reprogramming.

Some interfaces offer in general access to the internal flash memory: Nexus debug interface, external flash programmer interface, Self-Programming facilities and test interfaces.

In the following the security relevant items are listed. The features to protect the internal flash memory data from being read by unauthorized persons are described.

For more information about the flash memory, see chapter *“Flash Memory”*.

The following sections give an overview about supported code protection methods.

## 10.2 Flash Programmer and Self-Programming Protection

In general, illegal read-out and re-programming of the flash memory contents is possible via the flash programmer interface and the Self-Programming feature. For protection of the flash memory, the following flags provide various protection levels.

The flags can be set by flash programmers. For a description of flash memory programming see chapter *"Flash Memory"*.

**(1) Block erase protection flag (Block erase protection function)**

Set this flag to disable the feature to erase single blocks via the flash programmer interface. This flag does not affect the Self-Programming interface.

The flag is valid for the whole flash memory.

**(2) Program protection flag (Program protection function)**

Set this flag to disable the programming function via the flash programmer interface. This flag does not affect the Self-Programming interface.

The flag is valid for the whole flash memory.

**(3) Read-out protection flag (Read-out protection function)**

Set this flag to disable the feature that allows reading back the flash memory via the flash programmer interface. This flag does not affect the Self-Programming interface.

This flag is valid for the whole flash memory.

**(4) Boot block cluster protection flag**

Set this flag to disable erasure and rewrite of the boot block cluster.
The boot block cluster can not be manipulated in any way (no erase/write).

This applies in Serial- and Self-Programming mode.

Once this flag is set, it is impossible to reset this flag. Thus the boot block cluster content can not be changed any more.

**(5) Flash shielding**

Flash shielding allows to specify a random number of consecutive code flash memory blocks, the flash shield window, which can be erased and written. Erasure of and writing to all flash memory blocks outside the flash shield window is impossible.

## 10.3 On-Chip Debug Interface Protection

In general, illegal read-out of the flash memory contents is possible via the Nexus On-Chip Debug interface. For protection of the flash memory, the usage of the debug interface can be disabled.

The debug interface is protected via a 95-bit ID code and an internal control flag (On-Chip Debug enable control).

When the debugger is started, the status of the control flag in the internal flash memory is queried. Setting this flag to zero disables the On-Chip Debugger.

When debugging is enabled (On-Chip Debug enable flag is set), you have to enter the 95-bit ID code via the debugger. The code is compared with the ID code stored in the internal flash memory. If the codes do not match, debugging is not possible.

It is also possible to temporarily enable the On-Chip Debug interface by setting the On-Chip Debug enable flag via the user program, stored in the internal flash memory.
However after the device receives an external $\overline{\text{RESET}}$ or a Power-On-Clear reset POCRES, the On-Chip Debug enable flag is reset again and thus the On-Chip Debug interface is disabled, provided the On-Chip Debug enable flag in the flash memory is 0.

The On-Chip Debug enable flag and the ID code are stored in the flash memory configuration area and can be accessed via the OCDIDL, OCDIDM, OCDIDH registers.

### 10.3.1 On-Chip Debug enable flag

On-Chip Debugging is controlled by the On-Chip Debug enable flag OCDIDH.OCDID[95].

**(1) Permanent On-Chip Debug enable/disable**

The On-Chip Debug enable flag can be permanently set or reset

- while reprogramming the flash memory by an external flash programmer
- by the user's program with the Self-Programming feature.

Both methods change the content of the internal flash memory.

Any modification of the On-Chip Debug control flag OCDIDH.OCDID[95] by re-programming of the flash memory becomes effective after next release from an external $\overline{\text{RESET}}$ or a Power-On-Clear reset POCRES.

**(2) Temporary On-Chip Debug enable/disable**

Temporarily enabling respectively disabling the On-Chip Debugging is done by the On-Chip Debug control register IDMODI:

- IDMODI.IDEN = 1 and IDMODI.IDDATA = 0 sets OCDIDH.OCDID[95] = 0 and disables On-Chip Debugging
- IDMODI.IDEN = 1 and IDMODI.IDDATA = 1 sets OCDIDH.OCDID[95] = 1 and enables On-Chip Debugging

**Notes**

1. After release from the next external $\overline{\text{RESET}}$ or Power-On-Clear reset POCRES, the On-Chip Debug control flag OCDIDH.OCDID[95] takes on the value from the internal flash memory.
2. Writing to IDMODI with IDMODI.IDEN = 0 does not modify the On-Chip Debug control flag.

### 10.3.2 On-Chip Debug ID code

The 95-bit ID code is accessible via the register bits OCDIDH.OCDID[94:64], OCDIDM.OCDID[63:32] and OCDIDL.OCDID[31:0].

The ID code can be specified

- while reprogramming the flash memory by an external flash programmer
- by the user's program with the Self-Programming feature.

Both methods change the content of the internal flash memory.

Any modification of the ID code OCDIDH.OCDID[94:64], OCDIDM.OCDID[63:32] and OCDIDL.OCDID[31:0] by re-programming of the flash memory becomes effective after next release from an external $\overline{\text{RESET}}$ or a Power-On-Clear reset POCRES.

### 10.3.3 On-Chip Debug protection levels summary

The following table summarizes the protection levels of the On-Chip Debug interface:

**Table 10-1 On-Chip Debug protection levels**

| On-Chip Debug enable flag | ID code | Protection Level |
|---|---|---|
| 0 | X[a] | Level 2: Full protection<br>On-Chip debug interface can not be used. |
| 1 | user-specific ID code | Level 1: ID code protection<br>On-Chip Debug interface can only be used if the user enters the correct ID code. |
| | ID code is all ones[b] | Level 0: No protection<br>On-Chip Debug interface can be used. |

a) ID codes are not compared
b) This is the default state after the flash memory has been erased.

**Note** Once the On-Chip Debug interface has been set as "use-prohibited", it cannot be used until the On-Chip Debug enable flag is set to 1 by the user's program or by Self-Programming.

### 10.3.4 On-Chip Debug control registers

Following registers are dedicated to the On-Chip Debugger:

**Table 10-2 On-Chip Debug control registers overview**

| Register Name | Shortcut | Address |
|---|---|---|
| On-Chip Debug ID register L | OCDIDL | FF47 $0000_H$ |
| On-Chip Debug ID register M | OCDIDM | FF47 $0004_H$ |
| On-Chip Debug ID register H | OCDIDH | FF47 $0008_H$ |
| On-Chip Debug control register | IDMODI | FF47 $0000_H$ |

### (1) OCDIDL/M/H - On-Chip Debug ID registers

These registers hold the 95-bit ID code, the user is requested to enter upon start of a debug session.
By use of the OCDID[95] bit On-Chip Debugging can generally be disable respectively enabled.

**Access** In normal operation mode these registers can be read in 32-bit units.
Writing to this register is only possible in flash programming and Self-Programming mode.
On-chip Debug control bit OCDID[95] can be temporarily modified via the IDMODI register also in normal operation mode.

**Address** OCDIDL: FF47 $0000_H$, OCDIDM: FF47 $0004_H$, OCDIDH: FF47 $0008_H$

**Initial Value** User defined

OCDIDH:

| 31 | 30 | ... | | 0 |
|---|---|---|---|---|
| OCDID [95] | OCDID [94] | ... | OCDID [65] | OCDID [64] |
| R | R | ... | R | R |

OCDIDM:

| 31 | 30 | ... | | 0 |
|---|---|---|---|---|
| OCDID [63] | OCDID [62] | ... | OCDID [33] | OCDID [32] |
| R | R | ... | R | R |

OCDIDL:

| 31 | 30 | ... | | 0 |
|---|---|---|---|---|
| OCDID [31] | OCDID [30] | ... | OCDID [1] | OCDID [0] |
| R | R | ... | R | R |

**Table 10-3 OCDIDH/M/L registers contents**

| Register | Bit position | Bit name | Function |
|---|---|---|---|
| OCDIDH | 31 | OCDID[95] | Enable/disable On-Chip Debug:<br>0: On-Chip Debug disabled<br>1: On-Chip Debug enabled |
| OCDIDH | 30 - 0 | OCDID[94:64] | 95-bit On-Chip Debug ID code |
| OCDIDM | 31 - 0 | OCDID[63:32] | |
| OCDIDL | 31 - 0 | OCDID[31:0] | |

### (2) IDMODI - On-Chip Debug control register

This register allows to temporarily enable/disable On-Chip Debugging in normal operation mode, i.e. by the user's software.
This is performed by modifying the OCDIDH.OCDID[95] On-Chip Debug control bit.

**Note** Any modification of the On-Chip Debug control bit OCDIDH.OCDID[95] by the user's software becomes invalid after the next release from an external RESET or a Power-On-Clear reset POCRES.

**Protection** Writing to this register is protected by a special sequence of instructions by using a protection command register.
Refer to the section *"Write protected Registers"* in chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be written in 32-bit units.
Reading this register returns the value of OCDIDL.

**Address** FF47 $0000_H$

**Initial Value** User defined

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | IDEN | IDDATA |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 10-4 IDMODI register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1<br>0 | IDEN<br>IDDATA | Modify enable/disable On-Chip Debug control bit OCDIH.OCDI[95] |

| IDEN | IDDATA | OCDIH.OCDI[95] modification |
|---|---|---|
| 0 | X | not modified |
| 1 | 0 | set to 0 (On-Chip Debug disabled) |
| | 1 | set to 1 (On-Chip Debug enabled) |

# Chapter 11 Reset Controller

## 11.1 Functional Overview

Several system reset functions are provided in order to initialize the microcontroller hardware and its registers.

**Features summary**

A reset can be caused by the following events:

- External reset signal $\overline{\text{RESET}}$
  Noise in the external reset signal is eliminated by an analog filter.
- Power-On-Clear (POCRES)
- Overflow of the Watchdog Timers (WDTA0RES, WDTA1RES)
- Clock Monitors reset ($\overline{\text{CLMA0RES}}$, $\overline{\text{CLMA2RES}}$, $\overline{\text{CLMA3RES}}$)
- Low-Voltage Indicator reset (LVIRES)
- Software reset (SWRES)
- Debugger reset (DBRES)

The following block diagram shows the main components of the Reset Controller.

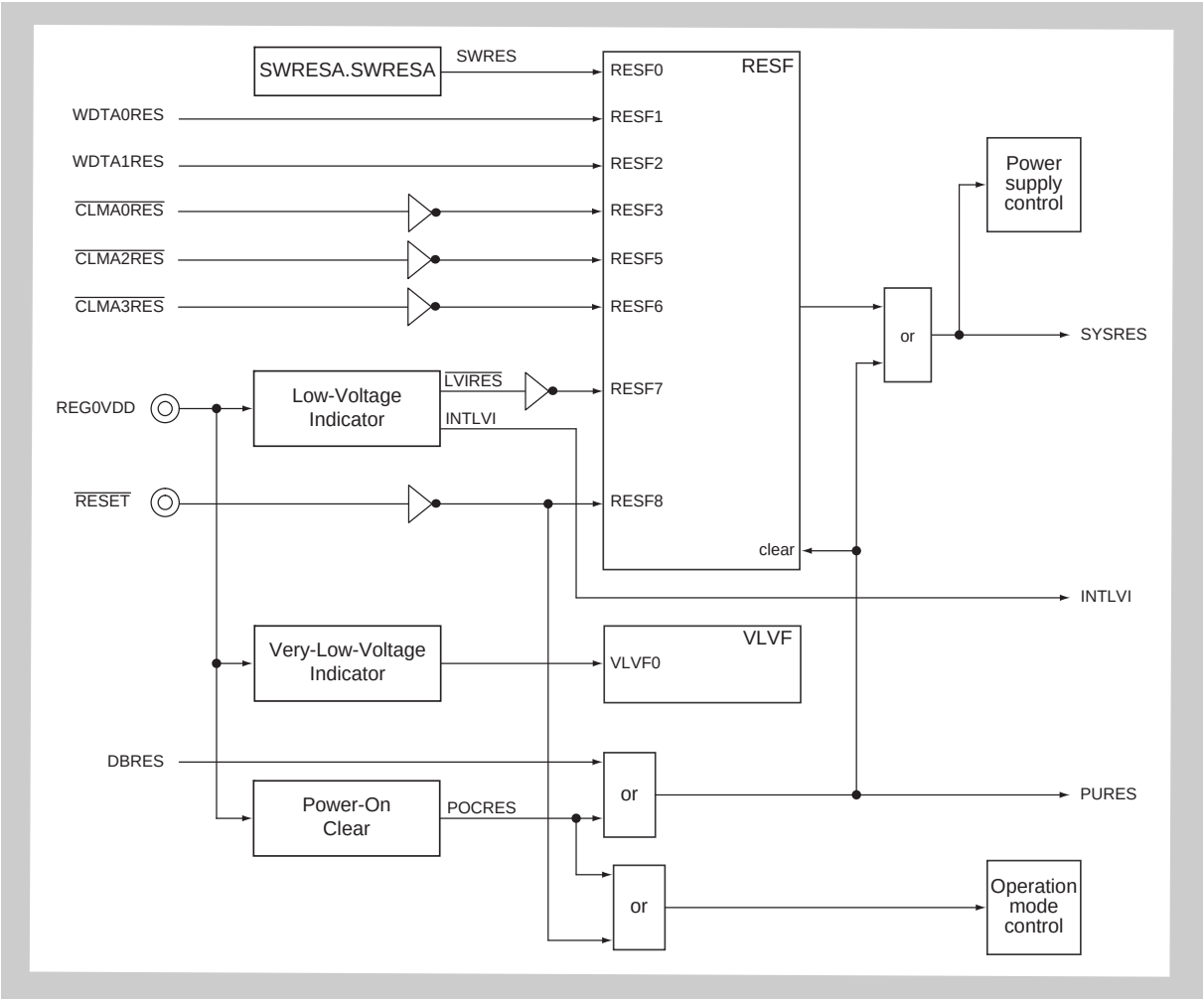



**Figure 11-1 Block diagram of the Reset Controller**

**(1) Internal reset signals**

The Reset Controller manages the generation of two reset signals upon occurrence of reset signals from various reset sources:

- system reset SYSRES
  The system reset is generated by all reset sources. SYSRES is applied to all microcontroller components, except the clock generator circuits. Consequently all clock generators continue operation, provided they were operating before the SYSRES.
- power-up reset PURES
  The power-up reset PURES is activated by the Power-On-Clear reset POCRES and the debugger reset DBRES. It resets some clock generators. Hence these clock generators stop operation and must be restarted. Refer to chapter *7 “Clock Controller” on page 254* detailed information, which clock generators are stopped by PURES.
  Assertion of PURES generates also a SYSRES, thus all microcontroller components are reset as well.

**(2) Operation mode control**

The microcontroller’s operation mode is determined upon release of the Power-On-Clear reset or the external RESET.
Refer to the section *“Operation modes”* in the *“CPU System Functions”* chapter for details about the operation modes.

**(3) Isolated areas power supply**

Any reset switches off the power supplies of the Isolated-Area-0.
Thus after a reset the Isolated-Area-0 has the same status as after initial power-up or after wake-up from DEEPSTOP mode, i.e. it has to be completely re-initialized.

**Note** Be aware that also after any reset the internal Data RAM content is undefined.

**(4) Reset flags**

The reset source flag register RESF holds a flag for each reset source, that is set, when the respective reset was asserted.
All reset flags are only cleared by a power-up reset PURES or by software.
Refer to section *11.2.1 “Reset flags” on page 402* for details.

**Note** In case of concurrently asserted power-up reset PURES and external RESET, setting of the RESF.RESF8 flag, that indicates the occurrence of an external RESET, depends on the time relation between PURES and the RESET. Refer to the Data Sheet for details.

**(5) Interrupts**

The Low-Voltage Indicator generates the interrupt INTLVI. Refer to section *11.2.3 “Low-Voltage Indicator (LVI)” on page 404* .

**(6) On-chip modules resets**

**Watchdog Timers reset**

The Watchdog Timers can generate the resets WDTA0RES and WDTA1RES.
Refer to section *11.2.6 “Watchdog Timers reset (WDTAnRES)” on page 410* .

**Clock Monitor resets**

The Clock Monitors can generate the resets $\overline{\text{CLMA0RES}}$, $\overline{\text{CLMA2RES}}$ and $\overline{\text{CLMA3RES}}$.
Refer to section *11.2.8 “Clock Monitors reset (CLMAnRES)” on page 410* for details.

**Debugger reset**

In case a debugger is connected, it can generate the reset DBRES, which leads to a power-up reset PURES>.
Refer to chapter *“ 11.2.9 “Debugger reset (DBRES)” on page 410* for details.

**(7) Software controlled resets**

**SWRES**

A software reset can be generated by use of the software reset control register SWRESA.
Refer to section *11.2.7 “Software reset (SWRES)” on page 410* for details.

**(8) Power supply supervision**

Several circuits observe the level of the external power supply REG0VDD and generate different actions, depending on REG0VDD level.

**Power-On Clear**

The Power-On-Clear circuit (POC) permanently compares the power supply voltage VDD with an internal reference voltage. It ensures that the microcontroller only operates as long as the power supply exceeds a well-defined limit.
Refer to section *11.2.2 “Power-On Clear (POC)” on page 403* for details.

**Low-Voltage Indicator**

The Low-Voltage Indicator (LVI) generates the LVIRES reset, if the voltage level of REG0VDD drops below a certain level. The level can be adjusted and the LVIRES can be masked.
Refer to section *11.2.3 “Low-Voltage Indicator (LVI)” on page 404* for details.

**Very-Low-Voltage Indicator**

The Very-Low-Voltage Indicator (VLVI) flag VLVF.VLVF0 indicates, that REG0VDD dropped below a certain level and retention of the microcontroller’s on-chip Back-up RAMs (BURAM) content is not guaranteed.
Refer to section *11.3.6 “Very Low-Voltage flag control registers” on page 417* for details.

## 11.2 Functional Description

### 11.2.1 Reset flags

The reset flag register RESF provides reset flags for each reset source.

If a reset has occurred, the assigned flag is set. This way the source of the reset can be evaluated.

All flags in RESF are not cleared, except by a power-up reset PURES (POCRES or DBRES). Thus this register behaves cumulative: each reset source sets its own flag, independent of all others.

### 11.2.2 Power-On Clear (POC)

The Power-On-Clear circuit (POC) permanently compares the power supply voltage REG0VDD with the internal reference voltage $V_{POC}$. It ensures that the microcontroller only operates as long as the power supply exceeds a well-defined limit.

If REG0VDD falls below the internal reference voltage (REG0VDD < $V_{POC}$), the internal reset signal POCRES, and thus a power-up reset PURES and the system reset SYSRES are generated.

Refer to the Data Sheet for the specification of the internal voltage reference level $V_{POC}$.

**Clock generator reset**

The Power-On-Clear reset POCRES resets also some clock generators via PURES. Hence these clock generators stop operation and must be restarted. Refer to chapter *"Clock Controller"* for detailed information, which clock generators are stopped by POCRES, respectively PURES.

At Power-On Clear reset, the reset status flag register RESF is cleared.

The Power-On-Clear function holds the microcontroller in reset state as long as the power supply voltage REG0VDD does not exceed the threshold level $V_{POC}$.

The following figure illustrates the timing of a POCRES.

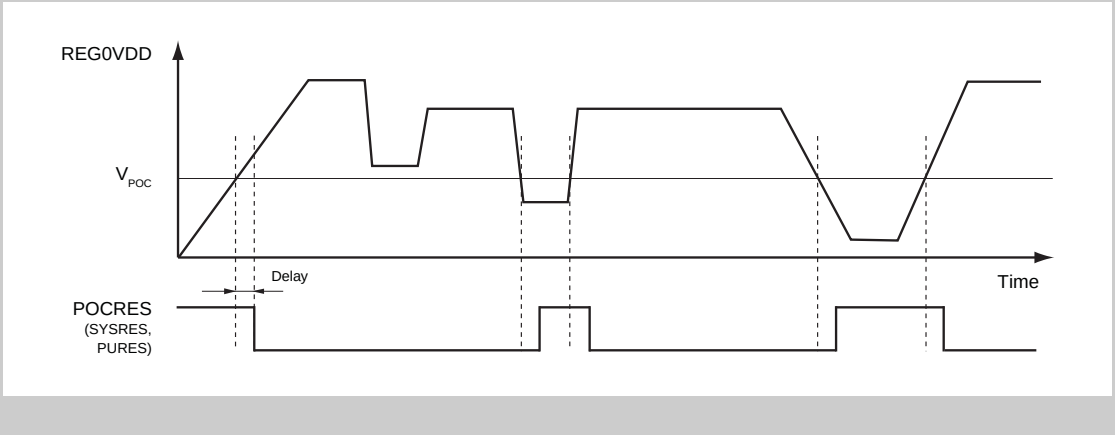



**Figure 11-2 POC reset timing**

**Delay**

A delay time is induced between REG0VDD crossing the $V_{POC}$ level and assertion of POCRES.
Refer to the Data Sheet for the delay time specification.

### 11.2.3 Low-Voltage Indicator (LVI)

The Low-Voltage Indicator circuit (LVI) permanently compares the power supply voltage REG0VDD with the LVI internal reference voltage $V_{LVI}$.

If REG0VDD falls below the internal reference voltage (REG0VDD < $V_{LVI}$), the internal reset signal $\overline{\text{LVIRES}}$, the interrupt INTLVI, and the system reset SYSRES are generated.

Additionally the $\overline{\text{LVIRES}}$ flag RESF.RESF7 is set.
RESF.RESF7 is not cleared automatically, if REG0VDD exceeds $V_{LVI}$. It is cleared by

- setting RESFC.RESFC7 = 1
- power-up reset PURES (POCRES or DBRES)

**LVI reference voltage**

The LVI reference voltage $V_{LVI}$ can be selected from 6 different levels by LVICNT.LVICNT[2:0].
If LVICNT.LVICNT[2:0] is set to $00_B$, the LVI is disabled.
Refer to the Data Sheet for the specification of the internal voltage reference levels $V_{LVI}$.

**LVI interrupt**

The LVI interrupt INTLVI is asserted if

- REG0VDD falls below the reference voltage (REG0VDD < $V_{LVI}$)
- REG0VDD rises above the reference voltage (REG0VDD > $V_{LVI}$)

The INTLVI interrupt can be used as a wake-up source from any stand-by mode. Refer to the chapter *8 “Stand-by Controller (STBC)” on page 322* for details.

**$\overline{\text{LVIRES}}$ mask**

Generation of the LVIRES can be disabled:

- LVICNT.LVIRESMSK = 0: LVIRES not masked (enabled)
- LVICNT.LVIRESMSK = 1: LVIRES masked (disabled)

The following figure illustrates the timing of a $\overline{\text{LVIRES}}$ and RESF.RESF7.

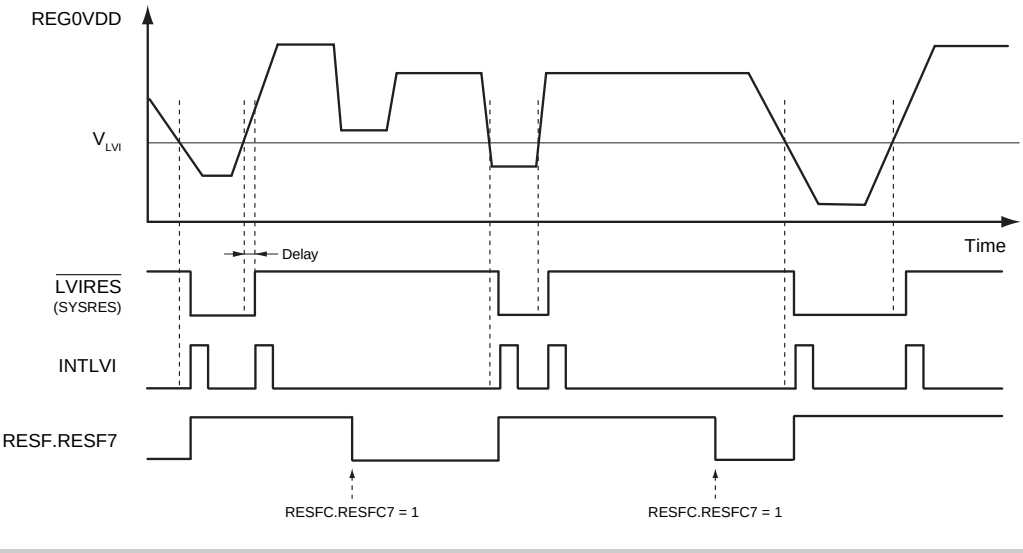



**Figure 11-3 LVI reset and interrupt timing**

**Delay** A delay time is induced between REG0VDD crossing the $V_{LVI}$ level and assertion of $\overline{\text{LVIRES}}$ respectively setting of RESF.RESF7.
Refer to the Data Sheet for the delay time specification.

### 11.2.4 Very-Low-Voltage Indicator (VLVI)

The Very-Low-Voltage Indicator circuit (VLVI) permanently compares the power supply voltage REG0VDD with the VLVI internal reference voltage $V_{VLVI}$.

Refer to the Data Sheet for the specification of the internal voltage reference level $V_{VLVI}$.

**BURAM content retention**

If the power supply voltage REG0VDD does not fall below $V_{VLVI}$, the content of the on-chip Back-up RAMs (BURAM) is retained and must not be restored.
If REG0VDD falls below $V_{VLVI}$ the BURAM content must be assumed to have altered. Thus the entire BURAM must be restored before continuing.

If REG0VDD falls below the internal reference voltage (REG0VDD < $V_{VLVI}$), the flag VLVF.VLVF0 is set.
VLVF.VLVF0 is not cleared automatically, if REG0VDD exceeds $V_{VLVI}$. It is cleared by

- setting VLVFC.VLVFC0 = 1

**Note** VLVF.VLVF0 is no affected by any reset.

The following figure illustrates the timing of VLVF.VLVF0.

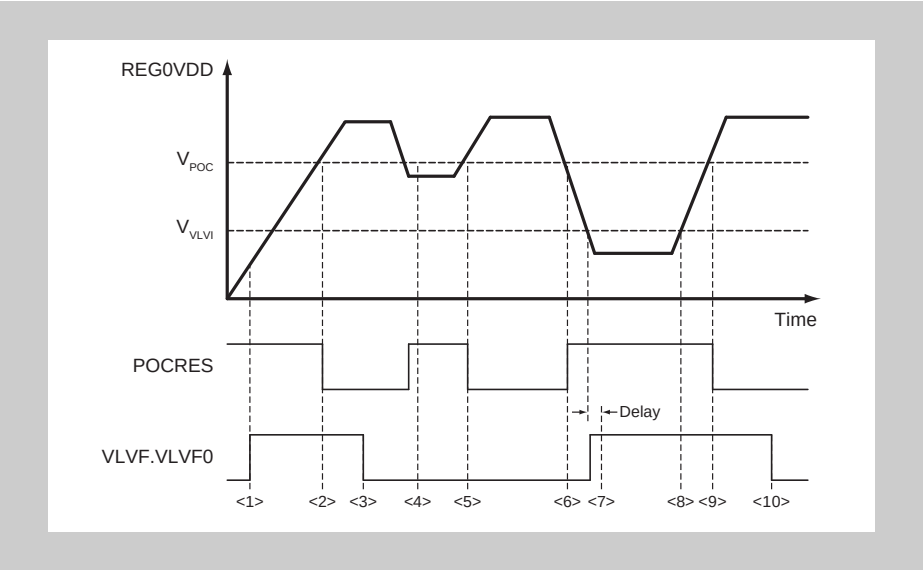



**Figure 11-4 VLVI timing**

**<1>** REG0VDD exceeds: $V_{VLVI}$ -> VLVF.VLVF0 = 1

**<2>** POCRES is released, because REG0VDD exceeds $V_{POC}$,
VLVF.VLVF0 = 1 indicates, that the BURAM is not initialized

**<3>** VLVF.VLVF0 = 0 must be set by software

**<4> to <5>** While REG0VDD remains below Power-On-Clear voltage $V_{POC}$, POCRES is asserted

**<6>** REG0VDD drops below Power-On-Clear voltage $V_{POC}$, POCRES is asserted

**<7>** REG0VDD drops below $V_{VLVI}$ -> VLVF.VLVF0 = 1

**<8>** Though REG0VDD rises above $V_{VLVI}$, VLVF.VLVF0 stays 1

**<9>** REG0VDD rises above $V_{POC}$, POCRES is de-asserted

**<10>** VLVF.VLVF0 = 0 must be set by software

**Delay** A delay time is induced between REG0VDD crossing the $V_{VLVI}$ level setting of VLVF.VLVF0.
Refer to the Data Sheet for the delay time specification.

### 11.2.5 External $\overline{\text{RESET}}$

Reset is performed when a low level signal is applied to the $\overline{\text{RESET}}$ pin.

Additionally the $\overline{\text{RESET}}$ flag RESF.RESF8 is set.
RESF.RESF8 is not cleared automatically, if $\overline{\text{RESET}}$ is de-asserted. It is cleared by

- setting RESFC.RESFC8 = 1
- power-up reset PURES (POCRES or DBRES)

The $\overline{\text{RESET}}$ is passed through an analog noise filter to prevent erroneous resets due to noise.

The following figure shows the timing when an external reset is performed. It explains the effect of the noise eliminator.

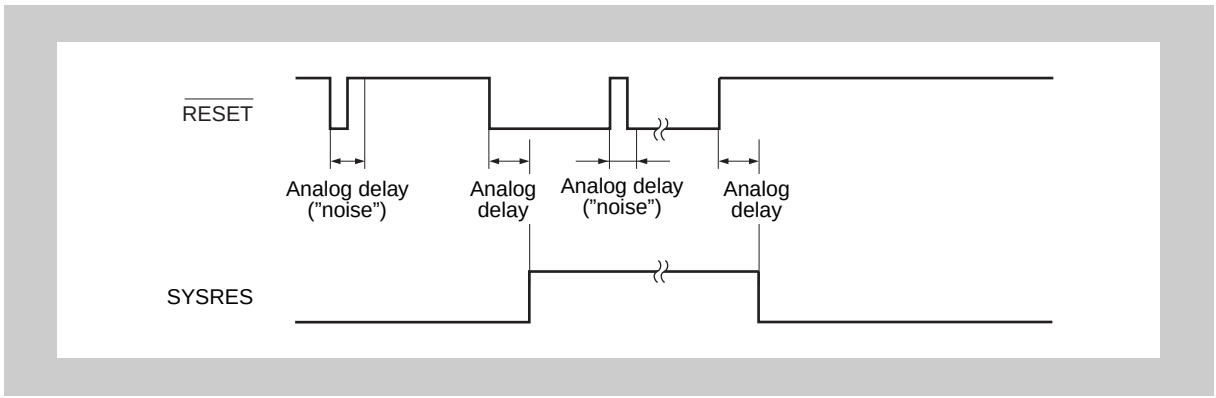



**Figure 11-5 External $\overline{\text{RESET}}$ timing**

The analog delay is caused by the analog filter. The filter regards pulses up to a certain width as noise and suppresses them.
For the minimum $\overline{\text{RESET}}$ pulse width refer to the Data Sheet.

**(1) MainOsc during $\overline{\text{RESET}}$**

During assertion of the external $\overline{\text{RESET}}$ the MainOsc is operating with its maximum amplification gain, provided its power supply is active.

The figure below shows the behaviour of the MainOsc under different circumstances, in particular in relation to the power-up reset PURES.

**PURES** The power-up reset PURES is activated by the Power-On-Clear reset POCRES or the debugger reset DBRES.
Note that the MainOsc is disabled after a PURES.

**MainOsc status** The figure shows the MainOsc status as follows:

- max = maximum amplification gain, which means short stabilization mode (MainOsc control register MOSCC.MOSCCSHTSTBY = 1)
- norm = normal amplification gain, which means normal stabilization mode (MainOsc control register MOSCC.MOSCCSHTSTBY = 0)
- off = MainOsc is stopped

**MainOsc start/stop** The MainOsc is started respectively stop by software via the MainOsc enable register MOSCE as follows:

- S/W start: MOSCEENTRG = 1
- S/W stop: MOSCEDISTRG = 1

For details concerning the MainOsc control and amplification gain refer to the description of the MainOsc and its control registers in the *"Clock Controller"* chapter.

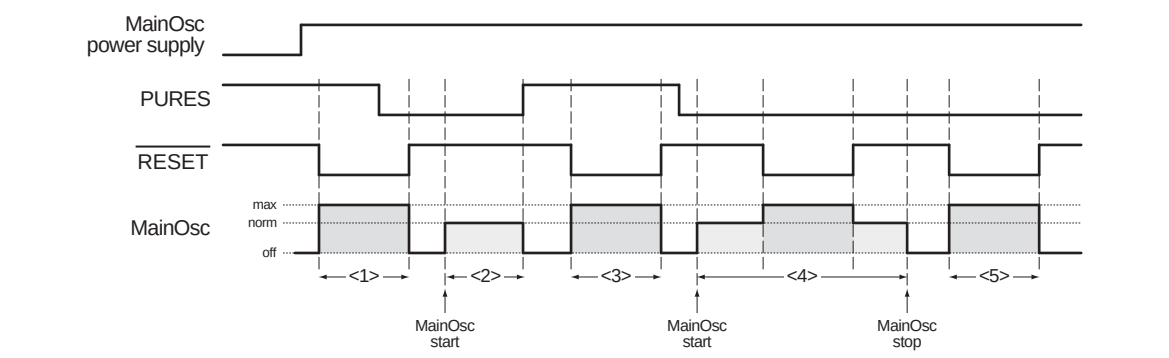



**Figure 11-6 MainOsc behaviour during $\overline{\text{RESET}}$**

**<1>** The MainOsc operates, as long as $\overline{\text{RESET}}$ stays active, even if PURES is deactivated before $\overline{\text{RESET}}$. However the MainOsc stops upon de-assertion of $\overline{\text{RESET}}$.

**<2>** The MainOsc is started by S/W and stopped by PURES.

**<3>** MainOsc is operating with maximum amplification gain during active $\overline{\text{RESET}}$.

**<4>** The MainOsc is operating with normal amplification gain and it is set to maximum during active $\overline{\text{RESET}}$. After de-assertion of $\overline{\text{RESET}}$ the amplification gain returns to its previous level.

**<5>** If the MainOsc was disabled, it operates with maximum amplification gain during active $\overline{\text{RESET}}$ and is stopped after de-assertion of $\overline{\text{RESET}}$.

### 11.2.6 Watchdog Timers reset (WDTAnRES)

The Watchdog Timers can be configured to generate a reset if the watchdog time expires. After watchdog reset, the Watchdog Timer reset flags RESF.RESF1 for WDTA0RES respectively RESF.RESF2 for WDTA1RES are set and the system reset SYSRES is asserted.

RESF.RESF1 (RESF.RESF2) is not cleared automatically, if WDTA0RES (WDTA1RES) is de-asserted. It is cleared by

- setting RESFC.RESFC1 = 1 (RESFC.RESFC2 = 1)
- power-up reset PURES (POCRES or DBRES)

### 11.2.7 Software reset (SWRES)

The software reset SWRES can be asserted by setting SWRESA.SWRESA0 = 1.

This generates a system reset SYSRES and sets the reset flag RESF.RESF0 = 1.

RESF.RESF0 is not cleared automatically. It is cleared by

- setting RESFC.RESFC0 = 1
- power-up reset PURES (POCRES or DBRES)

### 11.2.8 Clock Monitors reset ($\overline{\text{CLMAnRES}}$)

The Clock Monitors can generate the resets:

- $\overline{\text{CLMA0RES}}$, if a MainOsc fail is detected
- $\overline{\text{CLMA2RES}}$, if a High Speed IntOsc fail is detected
- $\overline{\text{CLMA3RES}}$, if a PLL0 fail is detected

Upon a Clock Monitor reset, the system reset SYSRES is generated and the respective reset flag in the RESF register is set.
These flags are not cleared automatically. They are cleared by

- setting RESFC.RESFC3 = 1 for $\overline{\text{CLMA0RES}}$, (RESFC.RESFC5 = 1 for $\overline{\text{CLMA2RES}}$, RESFC.RESFC6 = 1 for $\overline{\text{CLMA3RES}}$)
- power-up reset PURES (POCRES or DBRES)

### 11.2.9 Debugger reset (DBRES)

In case a debugger is connected, it can generate the reset DBRES.

DBRES activates the power-up reset PURES, thus operates in the same way like the Power-On-Clear reset POCRES:

- It resets some clock generators. Hence these clock generators stop operation and must be restarted.
- The reset status flag register RESF is cleared.

## 11.3 Reset Controller Registers

This section contains a description of all registers of the Reset Controller.

### 11.3.1 Writing to protected registers

Write protected registers are protected from inadvertent write access due to erroneous program execution, etc.

Following Reset Controller registers feature this special write protection:

- Software reset register SWRESA

Refer to the section *3.11 “Write protected Registers” on page 135* for a detailed description how to write to write protected registers.

### 11.3.2 Reset Controller registers overview

The Reset Controller is controlled and operated by the following registers:

**Table 11-1 Reset Controller registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| **General reset flags registers** | | |
| Reset factor register | RESF | FF42 $0160_H$ |
| Reset factor clear register | RESFC | FF42 $0168_H$ |
| **Software reset control registers** | | |
| Software reset register | SWRESA | FF42 $0204_H$ |
| **Low-Voltage Indicator reset control registers** | | |
| LVI control register | LVICNT | FF42 $0200_H$ |
| **Very-Low-Voltage flag control registers** | | |
| VLVF register | VLVF | FF42 $0180_H$ |
| VLVF clear register | VLVFC | FF42 $0188_H$ |

### 11.3.3 Reset Controller general control registers details

#### (1) RESF- Reset factor register

This register contains information about which type of resets occurred since the last Power-On Clear reset.

Each reset condition sets the corresponding flag in the register.
For example, if a Clock Monitor CLMA0RES occurs after a Watchdog Timer reset WDTA0RES, RESF reads 0000 000A$_H$.

**Access** This register can be read in 32-bit units.

**Address** FF42 0160$_H$

**Initial Value** 0000 0000$_H$. This register is initialized by a power-up reset PURES (POCRES and DBRES).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | RESF8 | RESF7 | RESF6 | RESF5 | 0 | RESF3 | RESF2 | RESF1 | RESF0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 11-2 RESF register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 8 | RESF8 | External reset flag<br>0: no RESET reset occurred<br>1: RESET reset has occurred<br>**Note:** In case of concurrently asserted power-up reset PURES and external RESET, setting of the RESF.RESF8 flag, that indicates the occurrence of an external RESET, depends on the time relation between PURES and the RESET. Refer to the Electrical Target Specification for details. |
| 7 | RESF7 | Low-Voltage Indicator reset flag<br>0: no LVIRES reset occurred<br>1: LVIRES reset has occurred |
| 6 | RESF6 | PLL0 Clock Monitor CLMA3 reset flag<br>0: no CLMA3RES reset occurred<br>1: CLMA3RES reset has occurred |
| 5 | RESF5 | High Speed IntOsc Clock Monitor CLMA2 reset flag<br>0: no CLMA2RES reset occurred<br>1: CLMA2RES reset has occurred |
| 3 | RESF3 | MainOsc Clock Monitor CLMA0 reset flag<br>0: no CLMA0RES reset occurred<br>1: CLMA0RES reset has occurred |

**Table 11-2 RESF register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | RESF2 | Watchdog Timer WDTA1 reset flag<br>0: no WDTA1RES reset occurred<br>1: WDTA1RES reset has occurred |
| 1 | RESF1 | Watchdog Timer WDTA0 reset flag<br>0: no WDTA0RES reset occurred<br>1: WDTA0RES reset has occurred |
| 0 | RESF0 | Software reset flag<br>0: no SWRES reset occurred<br>1: SWRES reset has occurred |

### (2) RESFC - Reset factor clear register

This register clears the reset flags of the RESF register.

**Access** This register can be read/written in 32-bit units.
Reading this register returns always 0000 $0000_H$.

**Address** $FF42\ 0168_H$

**Initial Value** Reading this registers returns always 0000 $0000_H$.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | RESFC8 | RESFC7 | RESFC6 | RESFC5 | 0 | RESFC3 | RESFC2 | RESFC1 | RESFC0 |
| R | R | R | R | R | R | R | R/W | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W |

**Table 11-3 RESFC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 8 | RESFC8 | External reset flag RESF.RESF8 flag clear<br>0: no function<br>1: clear RESF.RESF8 |
| 7 | RESFC7 | Low-Voltage Indicator reset flag RESF.RESF7 flag clear<br>0: no function<br>1: clear RESF.RESF7 |
| 6 | RESFC6 | PLL0 Clock Monitor CLMA3 reset flag RESF.RESF6 flag clear<br>0: no function<br>1: clear RESF.RESF6 |
| 5 | RESFC5 | High-Speed IntOsc Clock Monitor CLMA2 reset flag RESF.RESF5 flag clear<br>0: no function<br>1: clear RESF.RESF5 |
| 3 | RESFC3 | MainOsc Clock Monitor CLMA0 reset flag RESF.RESF3 flag clear<br>0: no function<br>1: clear RESF.RESF3 |
| 2 | RESFC2 | Watchdog Timer WDTA1 reset flag RESF.RESF3 flag clear<br>0: no function<br>1: clear RESF.RESF2 |
| 1 | RESFC1 | Watchdog Timer WDTA0 reset flag RESF.RESF2 flag clear<br>0: no function<br>1: clear RESF.RESF1 |
| 0 | RESFC0 | Software reset flag RESF.RESF0 flag clear<br>0: no function<br>1: clear RESF.RESF0 |

### 11.3.4 Software reset control registers details

#### (1) SWRESA - Software reset register

This register is used to generate a software reset SWRES.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register PROTCMD2.
Refer to the section *"Write protected Registers"* in chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 32-bit units.
Reading this register returns always 0000 $0000_H$.

**Address** FF42 $0204_H$

**Initial Value** Reading this registers returns always 0000 $0000_H$.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SWRESA0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R/W |

**Table 11-4 SWRESA register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | SWRESA0 | Software reset control<br>0: no function<br>1: generate software reset SWRES |

### 11.3.5 Low-Voltage Indicator reset control registers

#### (1) LVICNT - LVI control register

This register is used to control the Low-Voltage Indicator and to select the LVI detection level.

**Protection** Writing to this register is protected by a special sequence of instructions by using the protection command register PROTCMD2.
Refer to the section *"Write protected Registers"* in chapter *"CPU System Functions"* for a detailed description how to write to write protected registers.

**Access** This register can be read/written in 32-bit units.

**Address** FF42 $0200_H$

**Initial Value** $0000\ 0000_H$. This register is initialized by a power-up reset PURES (POCRES and DBRES).

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | LVIRESMK | LVICNT[2:0] | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R/W | R/W | R/W | R/W |

**Table 11-5 LVICNT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 | LVIRESMK | LVI reset LVIRES mask<br>0: LVIRES unmasked<br>1: LVIRES masked (LVIRES is not generated) |
| 2 to 0 | LVICNT[2:0] | LVI detection level<br>**LVICNT[2:0] / Detection level**<br>$000_B$ — LVI is deactivated<br>$001_B$ — LVI level 1 ($VLVIL_1$)<br>$010_B$ — LVI level 2 ($VLVIL_2$)<br>$011_B$ — LVI level 3 ($VLVIL_3$)<br>$100_B$ — LVI level 4 ($VLVIL_4$)<br>$101_B$ — LVI level 5 ($VLVIL_5$)<br>$11X_B$ — LVI level 6 ($VLVIL_6$)<br>Refer to the Data Sheet for the specification of the LVI detection levels. |

**Note** If the selected LVI detection level is close to the Power-On-Clear detection level, both may detect low voltage at the same time.
In this case the Power-On-Clear reset POCRES is performed and the reset factor register RESF is cleared. Thus the LVI reset flag RESF.RESF7 does not indicate the LVI detection.

### 11.3.6 Very Low-Voltage flag control registers

#### (1) VLVF - Very-Low-Voltage flag register

This register shows the status of the VLVF detection.

**Access** This register can be read in 32-bit units.

**Address** FF42 $0180_H$

**Initial Value** 0000 $0001_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VLVF0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 11-6 VLVF register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | VLVF0 | VLVI status<br>0: Very low voltage not detected<br>1: Very low voltage detected |

#### (2) VLVFC - Very-Low-Voltage flag clear register

This register clears the VLVF0 flag of the VLVF register.

**Access** This register can be read/written in 32-bit units.
Reading this register returns always 0000 $0000_H$.

**Address** FF42 $0188_H$

**Initial Value** Reading this registers returns always 0000 $0000_H$.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VLV FC0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 11-7 VLVFC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | VLVFC0 | VLVI flag VLVF.VLVF0 flag clear<br>0: no function<br>1: clear VLVF.VLVF0 |

# Chapter 12 OS Timer (OSTM)

This chapter contains a generic description of the OS Timer.

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 12.1 V850E2/Fx4-G OSTM Features

**Instances**

This microcontroller has the following number of instances of the OS Timer.

**Table 12-1 Instances of OS Timer**

| OS Timer | |
|---|---|
| Instances | 1 |
| Names | OSTM0 |

**Instances index n**

Throughout this chapter, the individual instances of the OS Timer are identified by the index “n” (n = 0), for example OSTMnCTL for the OS Timer n control register.

**Register addresses**

All OS Timer register addresses are given as address offsets from the individual base addresses <OSTMn_base>.
The <OSTMn_base> addresses of each OSTMn are listed in the following table:

**Table 12-2 Register base addresses <OSTMn_base>**

| OSTMn instance | <OSTMn_base> address |
|---|---|
| OSTM0 | FF80 $0000_H$ |

**Clock supply** All OS Timers provide one clock input.

**Table 12-3 OSTM clock supply**

| OSTMn instance | OSTMn clock | Connected to |
|---|---|---|
| OSTM0 | PCLK | Clock Controller CKSCLK_032 |

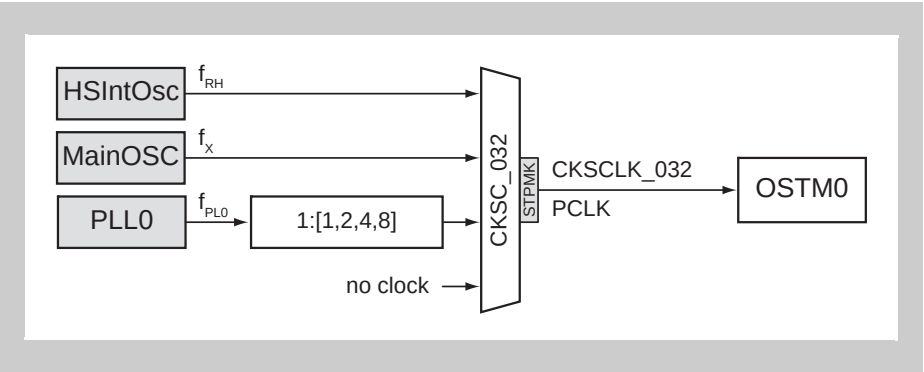



**Figure 12-1 OSTM clock supply**

**Interrupts** The OS Timers can generate the following interrupt requests:

**Table 12-4 OSTMn interrupt requests**

| OSTMn signals | Function | Connected to |
|---|---|---|
| OSTM0TINT | OSTMn interrupt | Interrupt Controller INTOSTM0 |

**OSTM H/W reset** The OS Timers and their registers are initialized by the following reset signal:

**Table 12-5 OSTMn reset signal**

| OSTMn | Reset signal |
|---|---|
| OSTM0 | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

## 12.2 Functional Overview

**Features summary** The OS Timer has two operation modes

– Interval timer mode

– Free-run compare mode

The following block diagram shows the main components of the OS Timer.

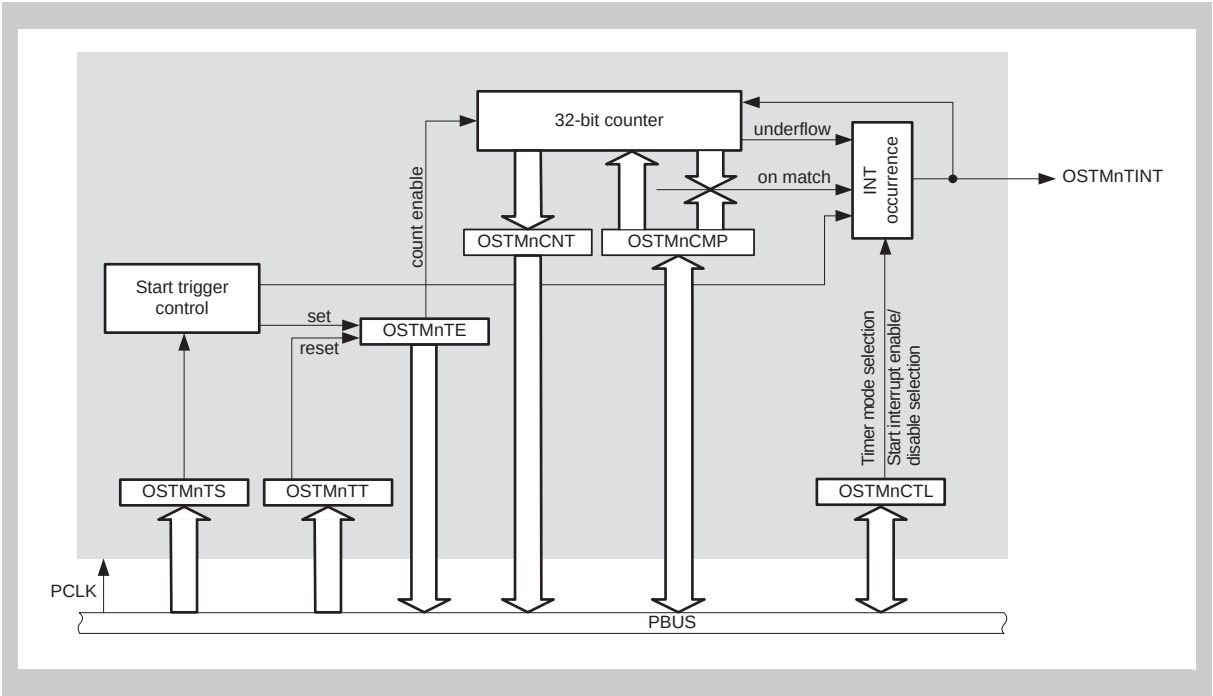



**Figure 12-2 Block diagram of the OS Timer**

## 12.3 Functional Description

The OS Timer is a 32-bit timer/counter.

It can be used as an interval timer or in free-run compare mode. The selected operation mode specifies the count direction and controls the interrupt request generation.

### 12.3.1 Count clock

The count clock of the OS Timer is defined by PCLK clock input.

This is illustrated in the following figures.

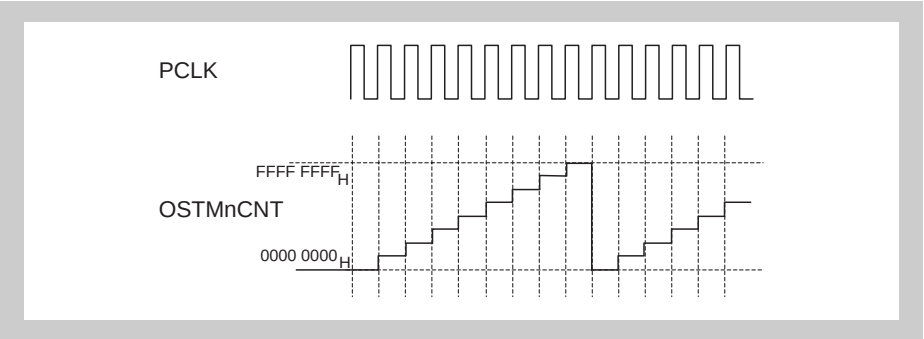



**Figure 12-3 Counter operation**

### 12.3.2 Interrupt request generation

By default, interrupt request OSTMnTINT is generated on counter underflow (interval timer mode) or when the counter matches the compare value (free-run compare mode).

Additionally, an interrupt request can be generated at counter start or counter restart. This is controlled by bit OSTMnCTL.OSTMnMD0.

This is illustrated in the following figure.

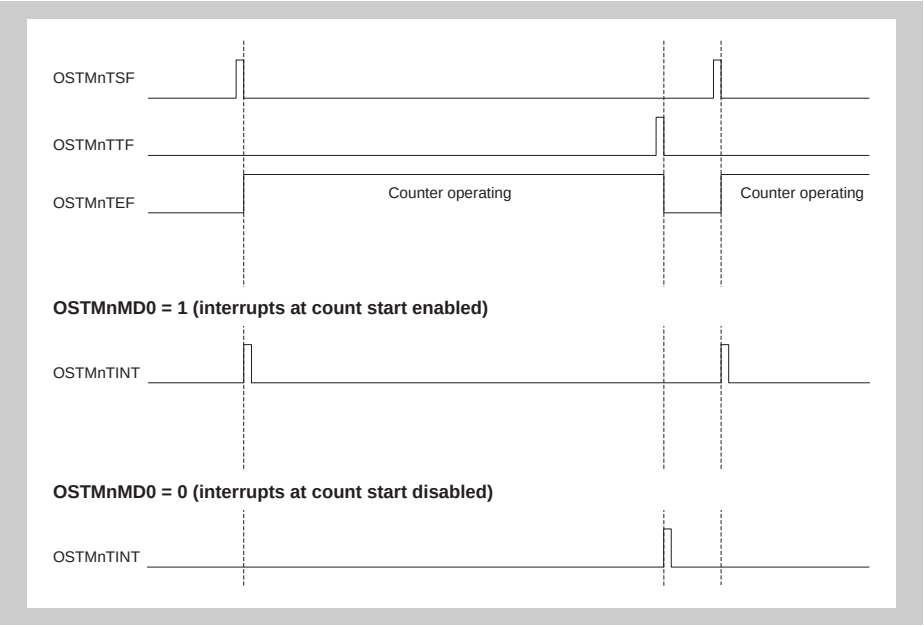



**Figure 12-4 Interrupt generation at count start**

### 12.3.3 Starting and stopping the timer

The OS Timer is started and stopped as follows:

**Start** The timer is started by setting the bit OSTMnTS.OSTMnTSF = 1 or

Status bit OSTMnTE.OSTMnTEF is set to 1 and bit OSTMnTS.OSTMnTSF returns to zero.

Depending on the operation mode, the counter starts to count down or to count up. Refer to *12.3.4 “Interval timer mode” on page 422* and *12.3.5 “Free-run compare mode” on page 426* for details.

**Stop** The timer is stopped by setting the bit OSTMnTT.OSTMnTTF = 1.

Status bit OSTMnTE.OSTMnTEF is cleared and bit OSTMnTT.OSTMnTTF immediately returns to zero.

When the counter is stopped, the register OSTMnCNT holds its current value until a new count operation starts.

### 12.3.4 Interval timer mode

In interval timer mode, the OSTM can be used as a reference timer generating interrupt requests at fixed intervals.

**(1) Basic operation in interval timer mode**

In interval timer mode, the timer counts down, starting from the value specified in the OSTMnCMP register. When the counter underflows (0000 $0000_H$ is reached), an interrupt request OSTMnTINT is generated.

The interval timer mode is set by OSTMnCTL.OSTMnMD1 = 0.

The OSTMnCMP register can be rewritten at any time. If it is rewritten during count operation, the counter loads the new OSTMnCMP value when the next 0000 $0000_H$ is reached. Then the counter continues with the new value.

**OSTMnTINT period**

The period of OSTMnTINT is:

- OSTMnTINT occurrence period = count clock period * (OSTMnCMP + 1)

The following figure shows the basic operation of the OS Timer in interval timer mode with counter start interrupt enabled (OSTMnCTL.OSTMnMD0 = 1):

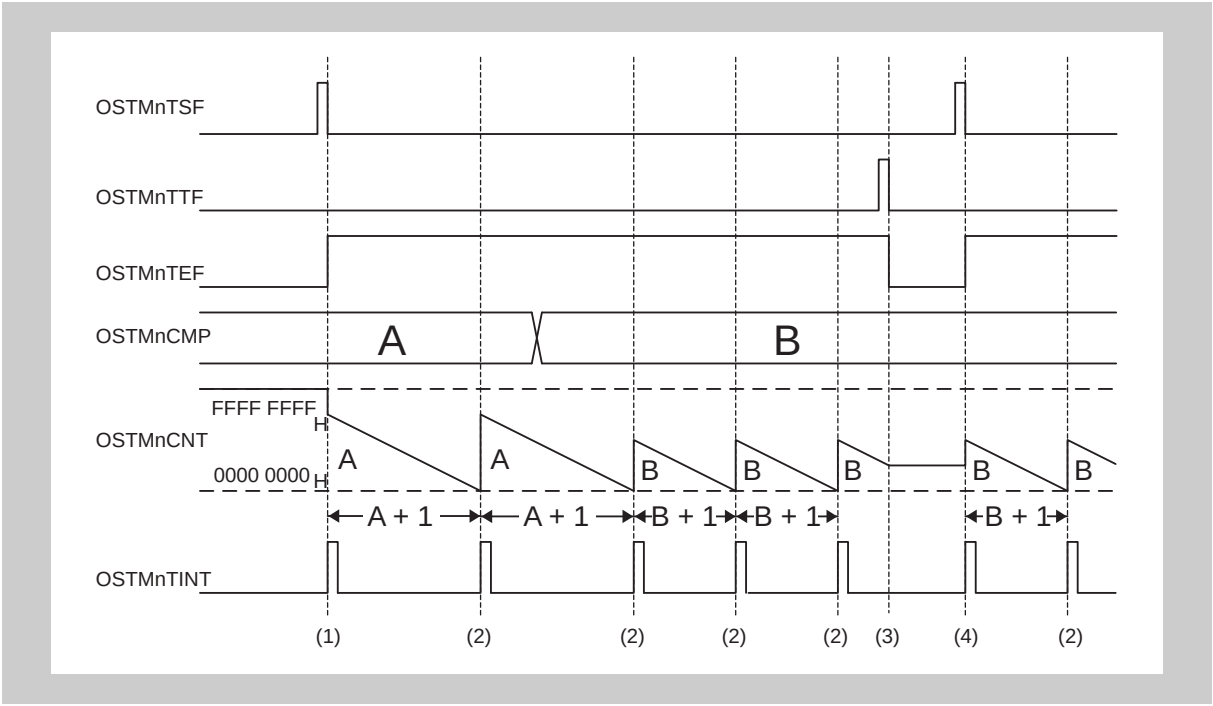



**Figure 12-5 Timing diagram of OSTM in interval timer mode**

**(1)** After counter start (OSTMnTS.OSTMnTSF = 1), the OSTMnTE.OSTMnTEF bit is set to indicate that the counter is enabled. OSTMnTS.OSTMnTSF returns to 0 in order to start the timer.
The counter counts down starting from the value of OSTMnCMP. The counter value is indicated in register OSTMnCNT. OSTMnTINT is asserted.

**(2)** When the counter reaches 0000 $0000_H$, the interrupt request OSTMnTINT is asserted. The counter loads the new start value from OSTMnCMP and continues to count down.

**(3)** At counter stop (OSTMnTT.OSTMnTTF = 1), the OSTMnTE.OSTMnTEF bit is cleared to indicate that the counter is disabled. OSTMnTT.OSTMnTTF returns to 0.The counter holds its current value until the counter is restarted.

**(4)** If the counter is restarted (OSTMnTS.OSTMnTSF = 1), the counter loads the new start value from OSTMnCMP and continues to count down.

**Forced restart**

A forced restart of the counter is performed by setting OSTMnTS.OSTMnTSF = 1 during the count operation.

The counter loads the start value from the OSTMnCMP register and continues to count down.

The following figure shows the forced restart of the OS Timer in interval timer mode, with counter start interrupt enabled (OSTMnCTL.OSTMnMD0 = 1):

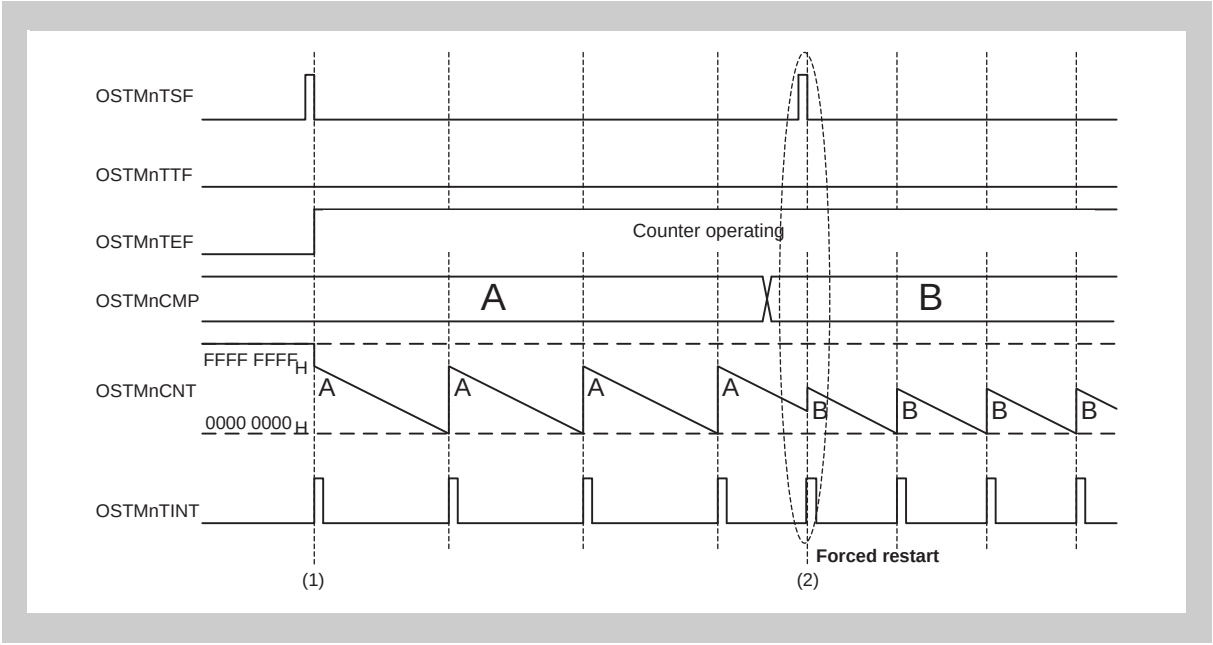



**Figure 12-6 Timing diagram of forced restart in interval timer mode**

The timing diagram above shows the following:

**(1)** The counter is started and stopped as shown and explained in *Figure 12-5 “Timing diagram of OSTM in interval timer mode” on page 423* .

**(2)** The counter is started again (OSTMnTS.OSTMnTSF = 1), while it is operating (OSTMnTE.OSTMnTEF = 1).
The counter immediately restarts counting down, starting with the current value of OSTMnCMP.
The interrupt request OSTMnTINT is asserted.

**(2) Operation when OSTMnCMP = 0000 0000$_H$**

When OSTMnCMP = 0000 0000$_H$ the OSTMnTINT interrupt request is set to 1, while the counter is enabled.

The following figure shows the operation of the OS Timer, when OSTMnCMP = 0000 0000$_H$, counter start interrupt is enabled (OSTMnCTL.OSTMnMD0 = 1):

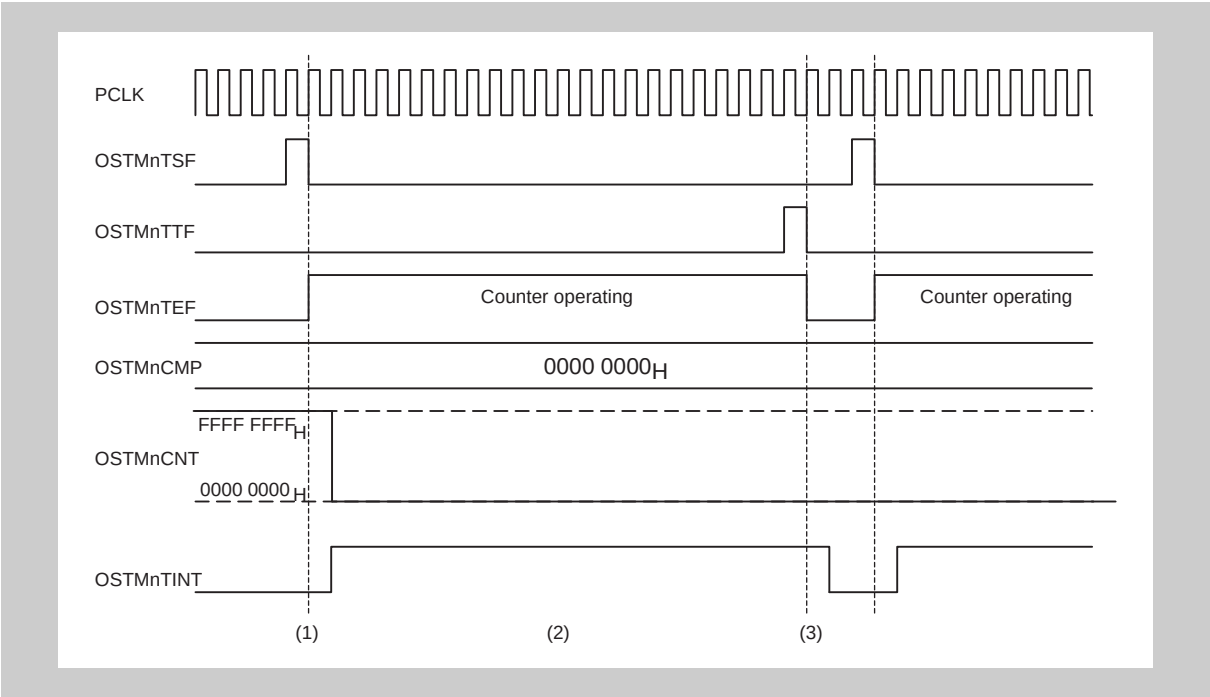



**Figure 12-7 Timing diagram when OSTMnCMP = 0000 0000$_H$ in interval timer mode**

The timing diagram above shows the following:

**(1)** After counter start, the counter starts counting, but will be reloaded with the OSTMnCMP value, and thus remains 0000 0000$_H$.

**(2)** The interrupt request OSTMnTINT is continuously asserted.

**(3)** After counter stop, the interrupt request OSTMnTINT is deasserted.

**(3) Initialization for interval timer mode**

The setting procedure in interval timer mode after a reset release is described below:

**Initialization**

1. Set the start value of the down-counter in the OSTMnCMP register.
2. Select the interval timer mode by clearing bit OSTMnCTL.OSTMnMD1.
3. Select the interrupt mode at counter start (OSTMnCTL.OSTMnMD0).

### 12.3.5 Free-run compare mode

#### (1) Basic operation in free-run compare mode

In free-run compare mode, the counter counts up from 0000 0000$_H$ to FFFF FFFF$_H$. When the value of the OSTMnCMP register matches the current count value, the OSTMnTINT interrupt request is output.

The free-run compare mode is selected by setting OSTMnCTL.OSTMnMD1 = 1.

The OSTMnCMP register can be rewritten at any time.

The following figure shows the basic operation of the OS Timer in free-run compare mode with counter start enabled (OSTMnCTL.OSTMnMD0 = 1):

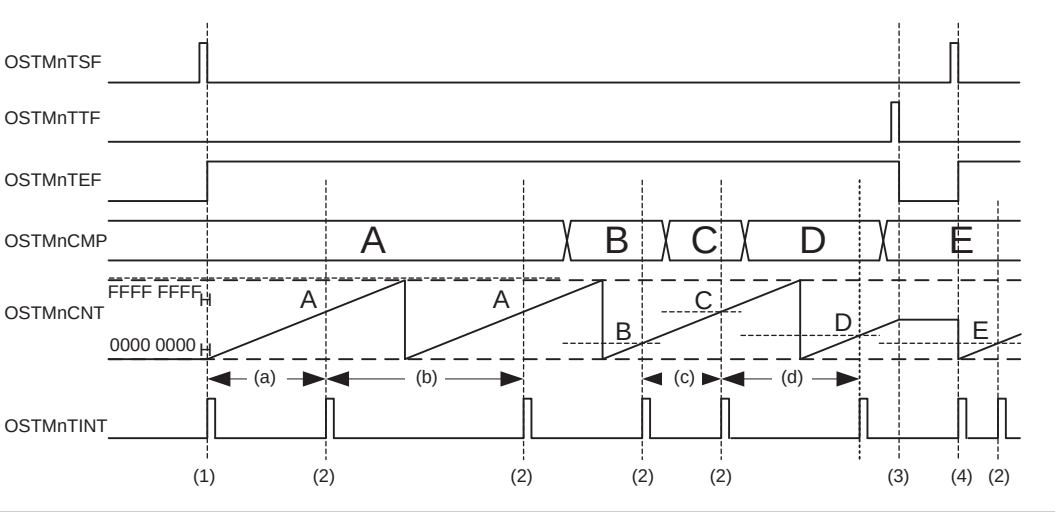



**Figure 12-8 Timing diagram of OSTM in free-run compare mode**

The timing diagram above shows the following:

**(1)** After counter start (OSTMnTS.OSTMnTSF = 1), the OSTMnTE.OSTMnTEF bit is set to indicate that the counter is enabled. OSTMnTS.OSTMnTSF returns to 0 in order to start the counter.
The counter counts up from 0000 0000$_H$ to FFFF FFFF$_H$. The counter value is indicated in register OSTMnCNT.

**(2)** When the value of the OSTMnCMP register matches the current count value, the interrupt request OSTMnTINT is asserted.

**(3)** At counter stop (OSTMnTT.OSTMnTTF = 1), the OSTMnTE.OSTMnTEF bit is cleared to indicate that the counter is disabled. OSTMnTT.OSTMnTTF returns to 0.
The counter holds its current value until the counter is restarted.

**(4)** If the counter is restarted (OSTMnTS.OSTMnTSF = 1), the counter starts counting from 0000 0000$_H$.

The OSTMnTINT occurrence period is different at count start and depends on the old and new compare value if OSTMnCMP is rewritten during operation:

**Table 12-6 OSTMnTINT occurrence timing**

| Old compare | New compare | Counter value at time of rewrite | OSTMnTINT occurrence period | Label in timing diagram |
|---|---|---|---|---|
| Counter start | | | (A + 1) * count clock period | (a) |
| A | A | No rewrite | (FFFF FFFF$_H$ + 1) * count clock period | (b) |
| B | C > B | B < counter value < C | (C - B) * count clock period | (c) |
| C | D < C | Counter value > D,C | (FFFF FFFF$_H$ - C + D + 1) * count clock period | (d) |

**Forced restart** A forced restart operation is not performed even if the bit OSTMnTS.OSTMnTSF is set during the count operation. The counter ignores this setting and continues counting.

**(2) Operation when OSTMnCMP = 0000 0000$_H$**

The following figure shows the operation of the OS Timer when OSTMnCMP = 0000 0000$_H$, counter start interrupt is enabled (OSTMnCTL.OSTMnMD0 = 1).

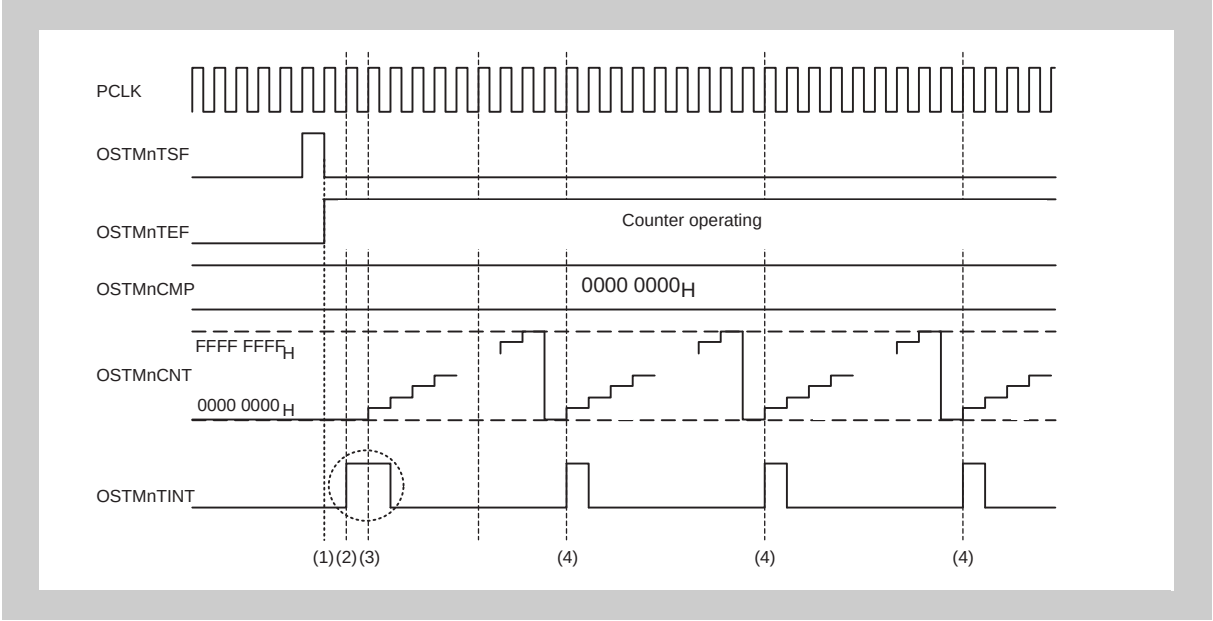



**Figure 12-9 Timing diagram when OSTMnCMP = 0000 0000$_H$ in free-run compare mode**

The timing diagram above shows the following:

**(1)** After counter start, the counter starts counting up from 0000 0000$_H$ to FFFF FFFF$_H$.

**(2)** The interrupt request OSTMnTINT at counter start is generated.

**(3)** If the current count value matches OSTMnCMP, the compare interrupt is generated. In the above case with OSTMnCMP = 0000 0000$_H$, OSTMnTINT stays active for 2 PCLK periods.

**(4)** Every FFFF FFFF$_H$ clock cycles the interrupt request OSTMnTINT is asserted.

**(3) Initialization for free-run compare mode**

The setting procedure in free-run compare mode after a reset release is described below:

**Initialization**

1. Set the compare value in the OSTMnCMP register.
2. Select the free-run compare mode by setting the bit OSTMnCTL.OSTMnMD1.
3. Select the interrupt mode at counter start by the bit OSTMnCTL.OSTMnMD0.

## 12.4 OS Timer Registers

This section contains a description of all registers of the OS Timer.

### 12.4.1 OS Timer registers overview

The OS Timer is controlled and operated by the following registers:

**Table 12-7 OS Timer registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| OSTM compare register | OSTMnCMP | <OSTMn_base> |
| OSTM counter register | OSTMnCNT | <OSTMn_base> + $4_H$ |
| OSTM count enable status register | OSTMnTE | <OSTMn_base> + $10_H$ |
| OSTM count start trigger register | OSTMnTS | <OSTMn_base> + $14_H$ |
| OSTM count stop trigger register | OSTMnTT | <OSTMn_base> + $18_H$ |
| OSTM control register | OSTMnCTL | <OSTMn_base> + $20_H$ |
| OSTM emulation register | OSTMnEMU | <OSTMn_base> + $24_H$ |

**<OSTMn_base>** The base addresses <OSTMn_base> of the OSTMn is defined in the first section of this chapter under the key word “Register addresses”.

## 12.4.2 OS Timer registers details

### (1) OSTMnCMP - OSTM compare register

This register stores the start value of the down-counter or the value with which the counter is compared, depending on the operation mode.

**Access** This register can be read/written in 32-bit units.

**Address** <OSTMn_base>

**Initial Value** $0000\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OSTMnCMP[31:16] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OSTMnCMP[15:0] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 12-8 OSTMnCMP register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 0 | OSTMnCMP[31:0] | • In interval timer mode: start value of the down-counter<br>• In free-run compare mode: compare value |

### (2) OSTMnCNT - OSTM counter register

This register indicates the count value of the timer.

**Access** This register can be read in 32-bit units.

**Address** <OSTMn_base> + $4_H$

**Initial Value** The initial value depends on the operation mode of the OS Timer, see *Table 12-10 “Correlation between operation mode, counting direction and initial value” on page 430* .

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OSTMnCNT[31:16] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OSTMnCNT[15:0] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 12-9 OSTMnCNT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 0 | OSTMnCNT[31:0] | 32-bit counter value |

The following table shows the correlation between operation mode, counting direction and initial value. The initial value is the value that is read after the operating mode has been changed.

**Table 12-10 Correlation between operation mode, counting direction and initial value**

| Timer operation mode | OSTMnCTL.OSTMnMD1 | Counting direction | Initial value |
|---|---|---|---|
| Interval Timer Mode | 0[a] | Down | FFFF FFFF$_H$ |
| Free Running Compare Mode | 1 | Up | 0000 0000$_H$ |

a) Value after reset.

### (3) OSTMnTE - OSTM count enable status register

This register indicates whether the counter is enabled or disabled.

**Access** This register can be read in 8-bit units.

**Address** <OSTMn_base> + $10_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | OSTMn TEF |
| R | R | R | R | R | R | R | R |

**Table 12-11 OSTMnTE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | OSTMn TEF | Indicates, whether the counter is enabled or disabled:<br>0: Counter disabled<br>1: Counter enabled<br>Setting OSTMnTS.OSTMnTSF to 1 sets this bit to 1.<br>Setting OSTMnTT.OSTMnTTF to 1 resets this bit to 0. |

**Note** If the counter is disabled, the counter value OSTMnCNT remains its value.

If the counter is restarted again, it

- continues with this value in interval timer mode.
- restarts with count value 0000 $0000_H$ in free-run compare mode.

### (4) OSTMnTS - OSTM count start trigger register

This register starts the counter.

**Access** This register can be written in 8-bit units. It is always read as $00_H$.

**Address** <OSTMn_base> + $14_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | OSTMn TSF |
| R | R | R | R | R | R | R | W |

**Table 12-12 OSTMnTS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | OSTMn TSF | Starts the counter:<br>0: No function<br>1: Starts the counter and sets OSTMnTE.OSTMnTEF = 1. When the counter is enabled, this bit returns to 0.<br>• In interval timer mode, forced restart is executed when this bit is set while OSTMnTE.OSTMnTEF = 1.<br>• In free-run compare mode, setting this bit is ignored while OSTMnTE.OSTMnTEF = 1. |

### (5) OSTMnTT - OSTM count stop trigger register

This register stops the counter.

**Access** This register can be written in 8-bit units. It is always read as $00_H$.

**Address** <OSTMn_base> + $18_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | OSTMn TTF |
| R | R | R | R | R | R | R | W |

**Table 12-13 OSTMnTT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | OSTMn TTF | Stops the counter:<br>0: No function<br>1: Stops the counter and clears the bit OSTMnTE.OSTMnTEF. When the counter has stopped, this bit returns to 0. |

### (6) OSTMnCTL - OSTM control register

This register specifies the counter operation mode and controls the generation of the interrupt request OSTMnTINT at counter start.

**Access** This register can be read/written in 8-bit units. It can only be written when the counter is disabled (OSTMnTE.OSTMnTEF = 0).

**Address** <OSTMn_base> + $20_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | OSTMn MD1 | OSTMn MD0 |
| R | R | R | R | R | R | R/W | R/W |

**Table 12-14 OSTMnCTL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | OSTMnMD1 | Specifies the counter operation mode:<br>0: Interval timer mode<br>1: Free-run compare mode |
| 0 | OSTMnMD0 | Controls the OSTMnTINT interrupt request at counter start:<br>0: Disables interrupt at counter start<br>1: Enables interrupt at counter start |

### (7) OSTMnEMU - OSTMn emulation register

This register controls whether the OSTMn can be stopped during emulation, for instance upon a breakpoint hit.

**Access** This register can be read/written in 8-bit units.

**Address** <OSTMn_base> + $24_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OSTMn SVSDIS | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 12-15 OSTMnEMU register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | OSTMn SVSDIS | Emulation control<br>0: OSTMn can be stopped during emulation<br>1: OSTMn continuous operating during emulation |

# Chapter 13 Window Watchdog Timer A (WDTA)

This chapter contains a generic description of the Window Watchdog Timer A (WDTA).

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 13.1 V850E2/Fx4-G WDTA Features

**Instances**

This microcontroller has the following number of instances of the Window Watchdog Timer A.

**Table 13-1 Instances of WDTA**

| Window Watchdog Timer A | |
|---|---|
| Instances | 2 |
| Names | WDTA0, WDTA1 |

**Instances index n**

Throughout this chapter, the individual instances of a Window Watchdog Timer A are identified by the index “n” (n = 0 to 1), for example, WDTAnWDTE for the WDTAn Watchdog Timer enable register.

**Register addresses**

All WDTAn register addresses are given as address offsets from the individual base address <WDTAn_base>.
The base address <WDTAn_base> of each WDTAn is listed in the following table:

**Table 13-2 Register base addresses <WDTAn_base>**

| WDTAn instance | <WDTAn_base> address |
|---|---|
| WDTA0 | FF80 $6000_H$ |
| WDTA1 | FF80 $7000_H$ |

**Clock supply** All Window Watchdog Timers A provide two clock inputs.

**Table 13-3 WDTAn clock supply**

| WDTAn instance | WDTAn clock | Connected to |
|---|---|---|
| WDTA0 | PCLK | Clock Generator CKSCLK_A02 |
| | WDTATCKI | Clock Generator CKSCLK_A07 |
| WDTA1 | PCLK | Clock Generator CKSCLK_001 |
| | WDTATCKI | Clock Generator CKSCLK_007 |

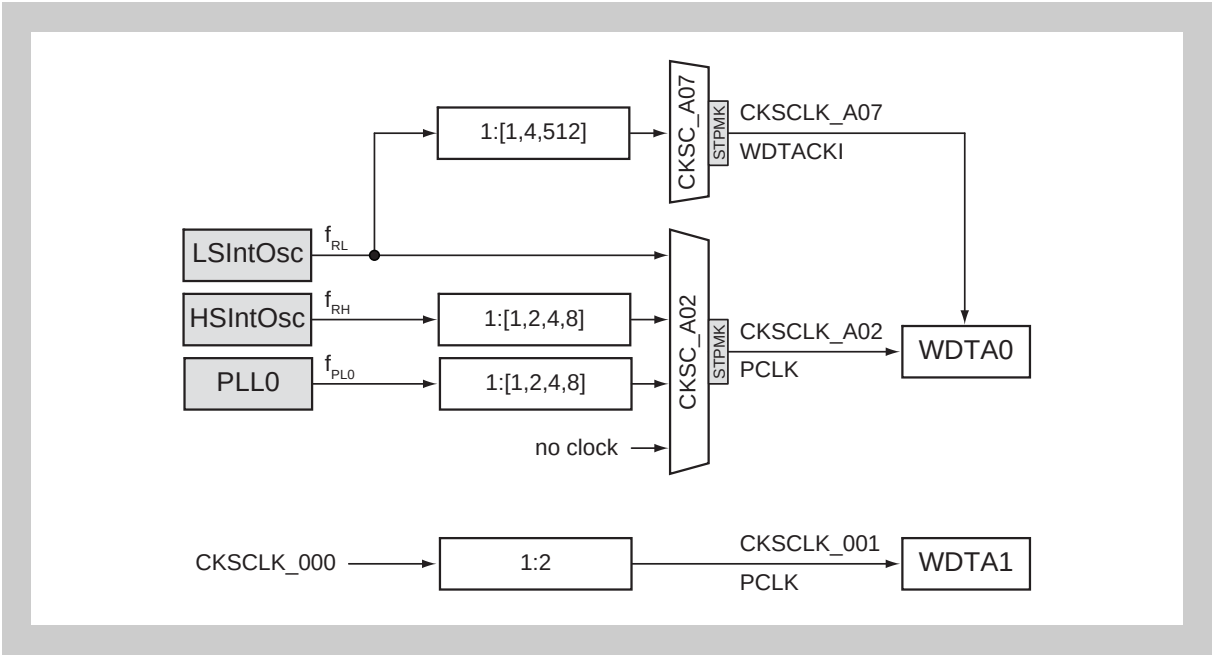



**Figure 13-1 WDTA clock supply**

**Interrupts and reset outputs** The interrupts and reset outputs of the WDTAn are listed in the table below.

**Table 13-4 WDTA interrupts and reset outputs**

| WDTAn signals | Function | Connected to |
|---|---|---|
| **WDTA0:** | | |
| WDTA0RES | WDTA0 error reset | Reset Controller WDTA0RES |
| WDTA0TNMI | WDTA0 error NMI | Interrupt Controller WDTA0TNMI[a] |
| INTWDT0 | WDTA0 75% interrupt | Interrupt Controller INTWDT0[a] |
| **WDTA1:** | | |
| WDTA1RES | WDTA1 error reset | Reset Controller WDTA1RES |
| WDTA1TNMI | WDTA1 error NMI | Interrupt Controller WDTA1TNMI[a] |
| INTWDT1 | WDTA1 75% interrupt | Interrupt Controller INTWDT1[a] |

a) These interrupts can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

**WDTA H/W reset**

The Window Watchdog Timers A and their registers are initialized by the following reset signal:

**Table 13-5 WDTAn reset signal**

| WDTAn | Reset signal |
|---|---|
| WDTA0 | • Reset Controller SYSRES |
| WDTA1 | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

**WDTA during DEEPSTOP mode**

If the V850E2/Fx4-G enters the DEEPSTOP stand-by mode, WDTA0 remains in operation and would need to be retriggered, although the CPU is not operating.
In order to stop also WDTA0 during DEEPSTOP, its operation clock CKSCLK_A07 can be stopped by setting CKSC_A07.STPMK_A07 = 0.

**WDTA during debugging**

When operating the V850E2/Fx4-G microcontroller under control of a debugger and the microcontroller is stopped, e.g. because of a breakpoint hit, the WDTA is stopped and all WDTA registers can not be written. In particular it is not possible to re-trigger the WDTA by the debugger's single-step execution.

## 13.2 WDTA Start-up Options

The start-up options determine the start-up configuration of the WDTA after reset release. They are described in the following table.

**Table 13-6 WDTA start-up options**

| Start-up option | Function | Description | Connected to |
|---|---|---|---|
| OPWDEN | WDTA enabler/disabler | Enables/disables the WDTA:<br>0: WDTA is disabled<br>1: WDTA is enabled | • WDTA0: Flash configuration option OPBT0.OPBT0[19]<br>• WDTA1: Flash configuration option OPBT0.OPBT0[23] |
| OPWDOVF[2:0] | Count clock setting | Specifies the reset value of the count clock control bits WDTAnMD.WDTAnOVF[2:0]. | WDTA0/WDTA1: Flash configuration option OPBT0.OPBT0[18:16] |
| OPWDTPR | Start mode signal selector | Specifies the signal that sets the start mode:<br>0: OPWDRUN start-up option<br>1: WDTATRTYP input signal<br>If WDTATRTYP is selected (OPWDTPR = 1), the start mode depends on the reset type.<br><br>Refer to section *"V850E2/Fx4-G WDTAn start modes"* below and *"WDTA after reset release"* in the section *"Functional Description"* for details. | fixed to 0 |
| OPWDRUN | Automatic start enabler | Specifies the start mode:<br>0: Software trigger start mode<br>1: Automatic start mode<br><br>Refer to section *"V850E2/Fx4-G WDTAn start modes"* below and *"WDTA after reset release"* in the section *"Functional Description"* for details. | • WDTA0: Flash configuration option OPBT0.OPBT0[20]<br>• WDTA1: Flash configuration option OPBT0.OPBT0[24] |
| OPWDWS[1:0] | Initial open window size setting | Specifies the reset value of the open window size control bits WDTAnMD.WDTAnWS[1:0].<br><br>The open window size control bits only apply after the first WDTA trigger and not after reset release.<br>After reset release the open window size is 100%.<br><br>Refer to *"Window function"* in the section *"Functional Description"* for details. | fixed to $11_B$ |
| OPWDINT | INTWDTn (75% interrupt) request generation | Specifies the reset value of control bit WDTAnMD.WDTAnWIE. This bit enables/disables the output of the 75% interrupt request INTWDTn.<br><br>Refer to *"75% interrupt output"* in the section *"Functional Description"* for details. | fixed to 0 |

### 13.2.1 V850E2/Fx4-G WDTAn start modes

If a Watchdog Timer is enabled (OPWDEN = 1), its start mode depends on OPWDRUN:

- OPWDRUN = 0: software trigger mode
- OPWDRUN = 0: automatic start mode

**Caution** If the Watchdog Timer is disabled (OPWDEN = 0), it can not be enabled afterwards.

## 13.3 Functional Overview

**Features summary**

The Window Watchdog Timer A has the following functions:

- Operation mode after reset selectable by using start-up options
- Fixed activation code and variable activation code (VAC) selectable
- Two start modes available:
  - Automatic start mode
  - Software trigger mode
- Reset-dependent start mode selection
- Operation upon error detection selectable:
  - Generation of NMI request WDTAnTNMI on error detection
  - Generation of reset WDTAnTRES on error detection
- Interrupt request generation at 75% of the counter overflow value
- Window function

The following figure shows the main components of the Window Watchdog Timer A:

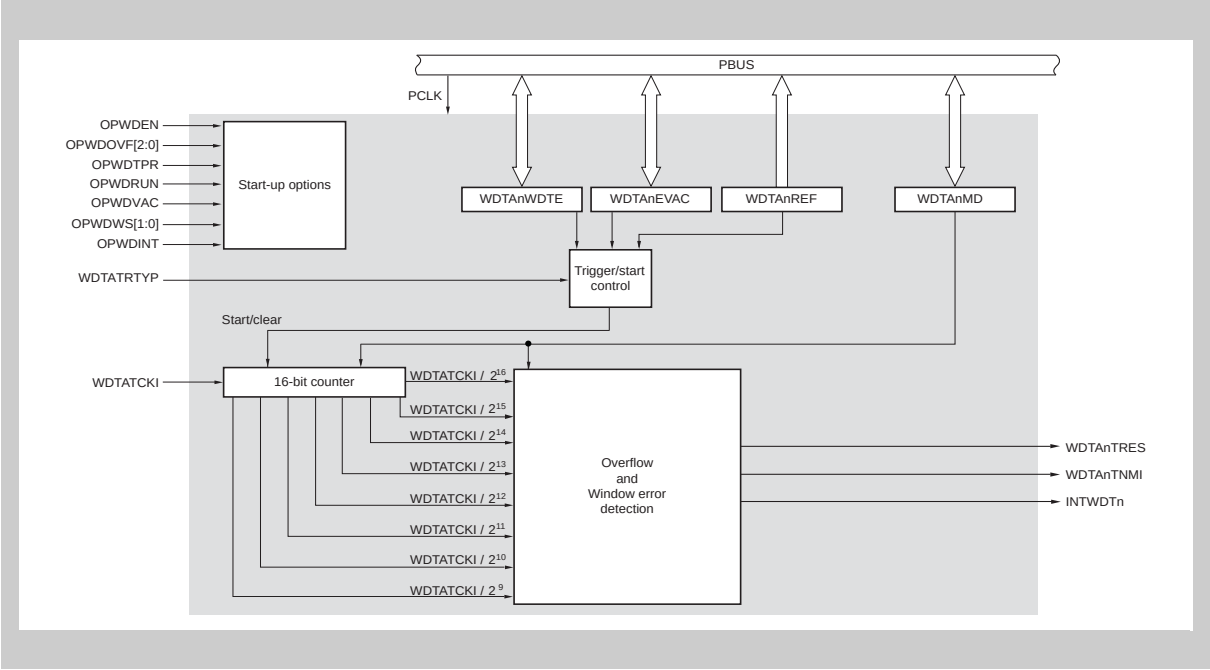



**Figure 13-2 Block diagram of the Window Watchdog Timer A**

## 13.4 Functional Description

The Window Watchdog Timer A generates a reset or a non-maskable interrupt if the 16-bit counter overflows or if any other error condition is fulfilled.
For a description of all error conditions, refer to the section *"Error detection"*.

The counter is cleared and restarted every time a WDTA trigger occurs in the open window period.
Refer to the sections *"WDTA trigger"* and *"Window function"* for details.

At 75% of the maximum counter value, the WDTA can generate an interrupt request INTWDTn.
Refer to the section *"75% interrupt output"* for details.

After reset release, the start-up options specify the start mode and the WDTA settings. The settings can be modified by writing the Watchdog Timer mode register WDTAnMD.
Refer to the section *"WDTA after reset release"* for details.

### 13.4.1 WDTA after reset release

#### (1) Start modes

The WDTA provides two modes for the counter start after reset release:

- Software trigger start mode

  The counter value remains $0000_H$ after reset release.

  The counter is started with the first WDTA trigger.
  The first trigger can occur any time after reset release.

- Automatic start mode

  The counter starts automatically after reset release.
  The first trigger must occur before the counter overflows.

#### (2) Start mode selection

The start mode can be selected as follows:

- By start-up options
- By the WDTATRTYP input signal

  This signal indicates the reset type. Thus, the selected start mode after reset release depends on the reset type.

The start mode selection is listed in the following table.

**Table 13-7 Start mode selection**

| Start-up options | | Input signal | Reset type | Start mode |
|---|---|---|---|---|
| OPWDTPR | OPWDRUN | WDTATRTYP | | |
| 0 | 0 | Ignored | Ignored | Software trigger |
| | 1 | | | Automatic |
| 1 | 0 | Ignored | Ignored | Software trigger |
| | 1 | 0 | Any apart from automatic start reset source | Software trigger |
| | | 1 | Automatic start reset source | Automatic |

### (3) WDTA settings after reset release

The WDTA settings are as follows between reset release and the first trigger:

| Function | Setting | Remark |
|---|---|---|
| Start mode | Specified by start-up options | For a description of the start modes, refer to the section *"WDTA after reset release"*. |
| Count clock | | |
| 75% interrupt mode | | |
| Error mode | Reset mode | Any error condition before the first trigger generates a reset. |
| Open window size | 100% | If automatic start mode is specified, the first trigger is valid any time before the counter overflows. |

**Change WDTA settings**

After the first trigger, the WDTA continues according to the settings of the Watchdog Timer mode register WDTAnMD.

To change the WDTA settings, WDTAnMD must be written *before* the first trigger. Changing the value of WDTAnMD *after* the first trigger leads to an error.

If WDTAnMD is not changed before the first trigger, the WDTA mode is specified by the initial value of WDTAnMD.

The new or initial value of WDTAnMD applies after the first trigger.

**Automatic start mode timing**

The automatic start mode timing and the changes to the WDTA settings are illustrated in the following figure.

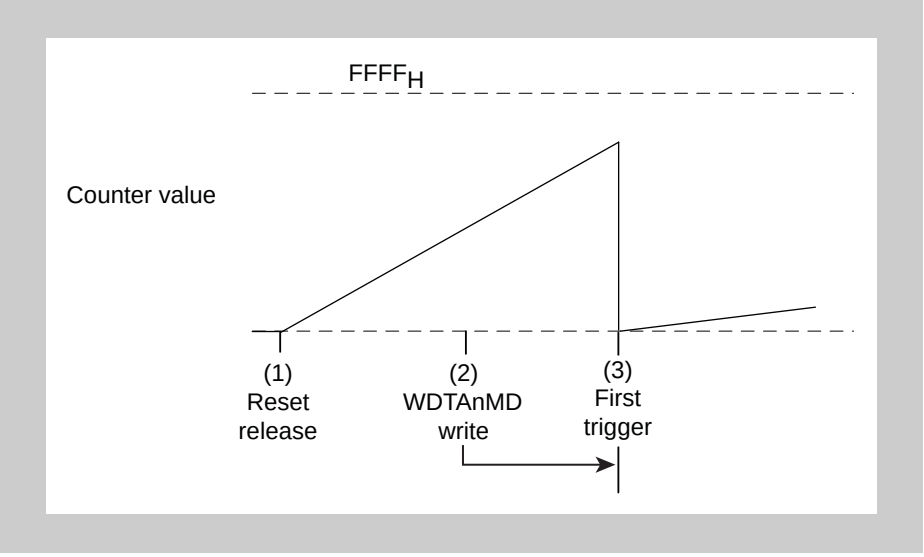



**Figure 13-3 Timing diagram of WDTA start in automatic start mode**

The timing diagram above shows the following:

1. After reset release, the counter starts immediately.
   The count clock is specified by the start-up options, for example:
   – Count clock after reset release = WDTATCKI / $2^{13}$
   (OPWDOVF[2:0] = $100_B$)

2. WDTAnMD is written before the first trigger. However, the settings are not applied immediately.
3. The first trigger must occur before the counter overflows.
   After the first trigger, the settings specified in WDTAnMD are applied, for example a new count clock:
   – Count clock after first trigger = WDTATCKI / $2^{16}$
     (WDTAnMD.WDTAnOVF[2:0] = $111_B$

   With the decreased count clock, the counter value rises more slowly over time.

**Software trigger start mode timing**

The software trigger start mode timing and the changes to the WDTA settings are illustrated in the following figure.

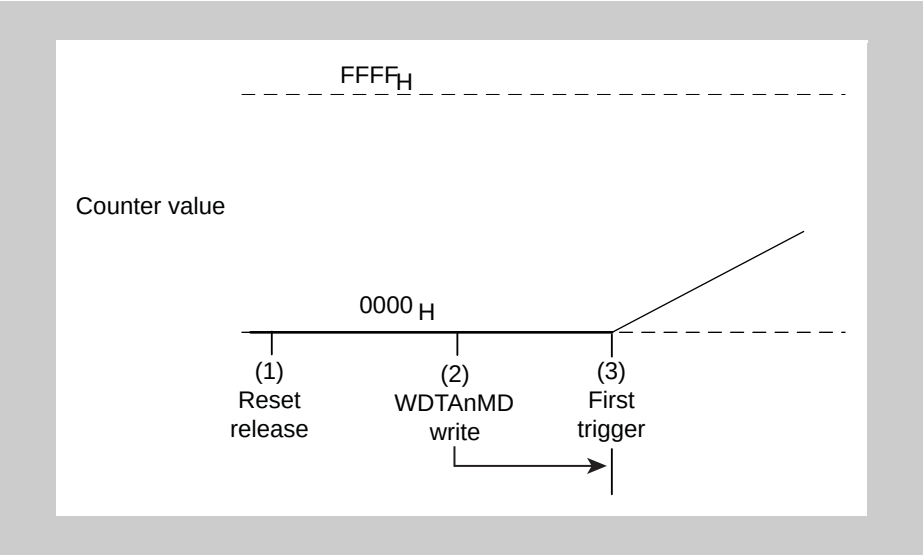



**Figure 13-4 Timing diagram of WDTA start in software trigger start mode**

The timing diagram above shows the following:

1. After reset release, the counter remains $0000_H$ until the first trigger.
   The count clock is specified by the start-up options, but it does not have any effect.
2. WDTAnMD is written before the first trigger. However, the settings are not applied immediately.
3. The counter starts at the first trigger.
   The count clock and other settings specified in WDTAnMD are applied.

### 13.4.2 WDTA trigger

The WDTA trigger has the following functions:

- Counter start trigger in software trigger start mode
- Counter restart trigger to avoid counter overflow

The trigger register to be used differs depending on whether the activation code is fixed or variable. The type of activation code and the associated trigger register are specified by using the start-up option OPWDVAC.

**Table 13-8 Trigger register and activation code**

| Type of activation code | Trigger register | Activation code |
|---|---|---|
| Fixed | WDTAnWDTE | $AC_H$ |
| Variable | WDTAnEVAC | Refer to the section *"Varying Activation Code calculation"* for details. |

#### (1) Variable activation code calculation

The variable activation code (ExpectWDTE) is calculated using a reference value in register WDTAnREF. The reference value in WDTAnREF is updated each time the trigger register WDTAnEVAC is written.

- Use the expression below to calculate the variable activation code (ExpectWDTE):
  ExpectWDTE = $AC_H$ – WDTAnREF (old)
- Use the expression below to calculate how the WDTAnREF value is updated:
  WDTAnREF (new) = rotate left 1 bit (ExpectWDTE)

**Note** WDTAnREF is also updated upon an erroneous WDTA trigger, i.e.:

- When the value written to WDTAnEVAC differs from the expected activation code.
- When writing to WDTAnEVAC outside the open window.

In either case, a WDTAnTRES reset or WDTAnTNMI interrupt request is also generated.

The table below lists the variable activation codes according to the number of triggers.

**Table 13-9 Expected activation code development**

| No[a] | WDTAnREF (old) | | ExpectWDTE ($AC_H$ - WDTAnREF) | | WDTAnREF (new) | |
|---|---|---|---|---|---|---|
| 0 | 0000 0000 | $00_H$ | 1010 1100 | $AC_H$ | 0101 1001 | $59_H$ |
| 1 | 0101 1001 | $59_H$ | 0101 0011 | $53_H$ | 1010 0110 | $A6_H$ |
| 2 | 1010 0110 | $A6_H$ | 0000 0110 | $06_H$ | 0000 1100 | $0C_H$ |
| ... | ... | ... | ... | ... | ... | ... |

a) Number of triggers after reset

This generates a sequence of 122 different numbers that have to be used to re-trigger the Watchdog Timer.

**Note** Bit 7 of the WDTAnEVAC register (WDTAnEVAC7) cannot be cleared to 0 after the WDTA has been started. Thus even if bit 7 of the activation code is 0, the WDTA will not stop.

### 13.4.3 Error detection

The conditions for error detection are:

- Overflow interval time is exceeded (counter overflow)
- Wrong activation code is written to the trigger register
- Writing to the trigger register outside the open window.
- Illegal update of Watchdog Timer mode register WDTAnMD:
  - Writing a *new* value to WDTAnMD after the first trigger leads to an error detection.
  - Writing the same value to WDTAnMD after the first trigger does *not* lead to an error detection.

**Error mode** When an error is detected, either an NMI request (WDTAnTNMI) or a reset (WDTAnTRES) is generated.
WDTAnMD.WDTAnERM selects the error mode:

- WDTAnMD.WDTAnERM = 0: NMI mode
- WDTAnMD.WDTAnERM = 1: reset mode

**Caution** Due to the fact, that any modification of the WDTAnMD register becomes effective with the first WDTA trigger, the initial value of WDTAnMD remains active until the first WDTA trigger.

This means in particular, that a WDTAnTRES is generated in the following case:

- WDTA is changed to NMI mode by WDTAnMD.WDTAnERM = 0
- WDTAnMD is written again before the first WDTA trigger (error condition)

**After error detection**

After detection of an error the Watchdog Timer operation is stopped. For restarting the Watchdog Timer a reset is necessary.

- In NMI mode:
  The NMI service routine has to initiate assertion of a reset, e.g. by applying a software reset.
- In reset mode:
  The Watchdog Timer generates the reset by itself.

The following figure shows the reset or NMI request generation when the counter overflows and automatic start mode is selected.

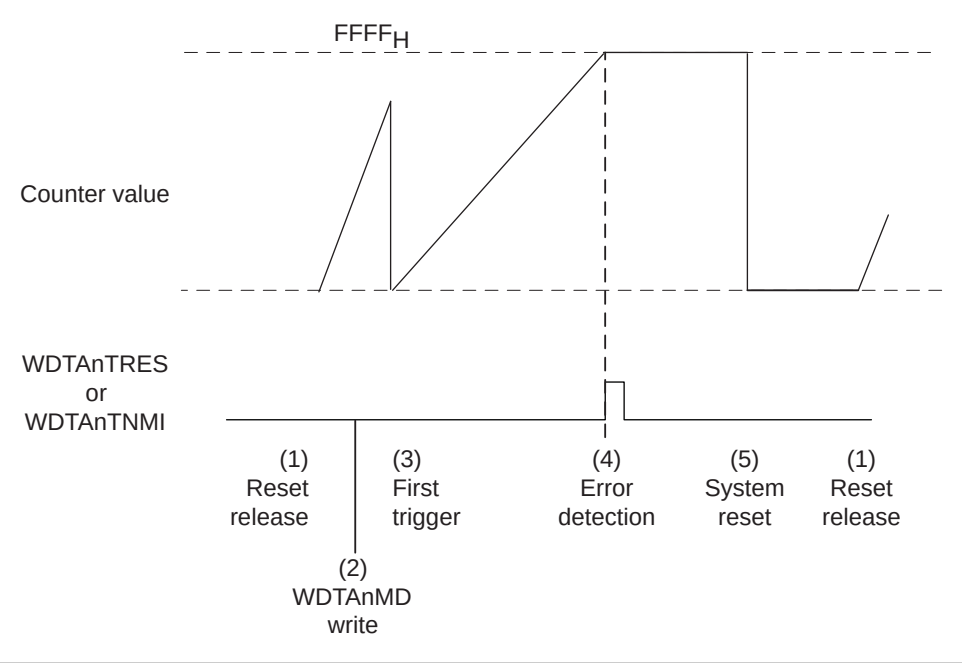



**Figure 13-5 Timing diagram of WDTA NMI request or reset generation**

The timing diagram above shows the following:

1. After reset release, the counter starts (automatic start mode is selected).
2. WDTAnMD is written before the first trigger. However, the settings are not applied immediately.
3. The counter is cleared at the first trigger and the new WDTA settings are applied.
4. When the counter overflows, an error is detected. Depending on the error mode, either interrupt request WDTAnTNMI or reset WDTAnTRES is generated.
   The counter value remains, the Watchdog Timer is stopped and can not be restarted, until a system reset is performed.
5. When the system is reset, the counter is cleared and stopped until reset release.

### 13.4.4 75% interrupt output

When the counter reaches 75% of the maximum counter value, the interrupt request INTWDTn is generated.

This function can be automatically enabled with the start-up option OPWDINT = 1.
By use of WDTAnMD.WDTAnWIE this function can be enabled respectively disabled afterwards.

The following figure shows the 75% interrupt request generation under following conditions:

- Automatic start mode selected
- Count clock changes after first trigger

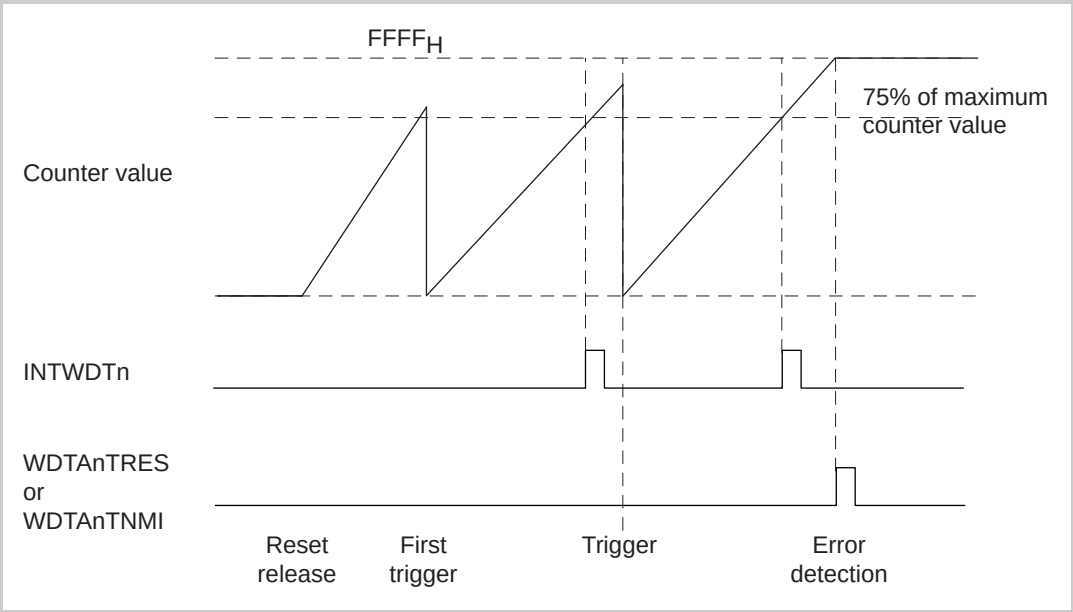



**Figure 13-6 Timing diagram of WDTA 75% interrupt output**

### 13.4.5 Window function

When the open window size is set to less than 100%, an error is detected if the trigger occurs outside the open window.

The definition of the open window size differs before and after the first trigger:

- After reset release, the open window size is 100%.

  OPWDWS[1:0] and bits WDTAnMD.WDTAnWS[1:0] are not applied.
- After the first trigger, the open window size is specified by bits WDTAnMD.WDTAnWS[1:0].

The following figure shows WDTA operation with an open window size of 25% and with automatic start mode selected.

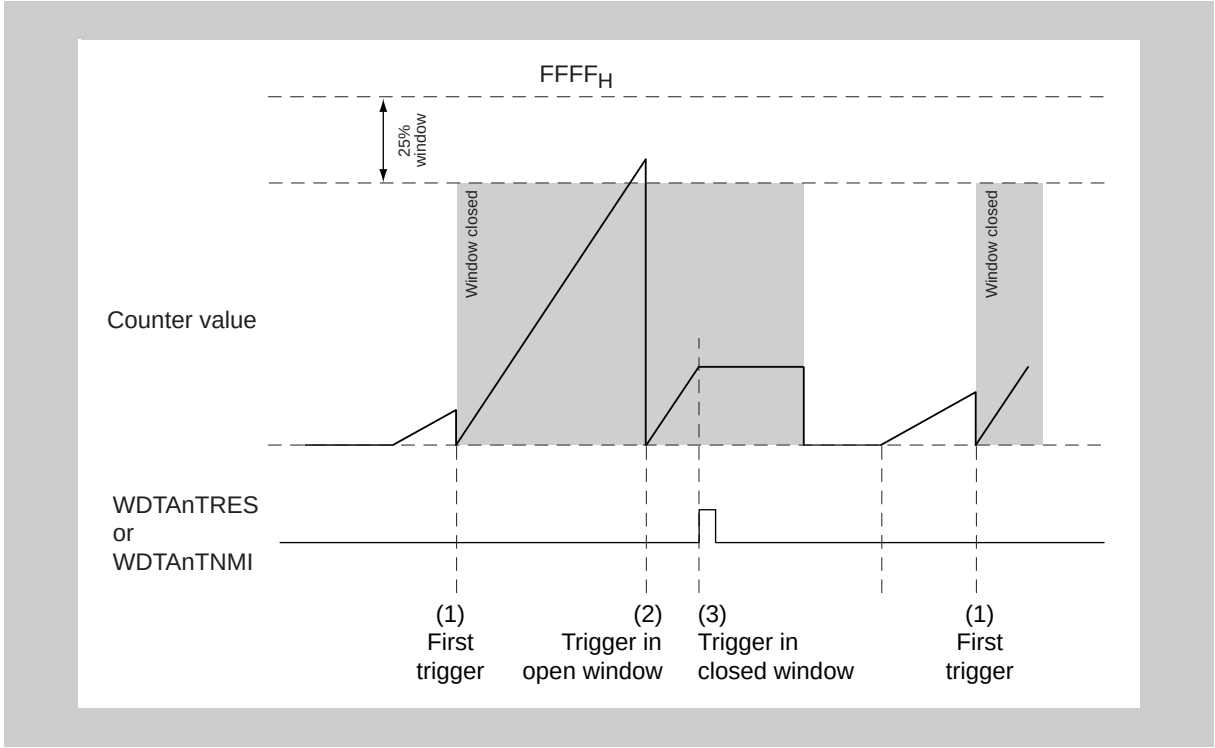



**Figure 13-7 Timing diagram of WDTA window function**

The timing diagram above shows the following:

1. The open window size is fixed to 100% for the first trigger.
2. A trigger that occurs in the open window does not generate an error.
3. A trigger that occurs in the closed window generates a WDTAnTNMI request or a WDTAnTRES reset, depending on the selected error mode.

## 13.5 Application hint: Evaluation of the Watchdog status

In case the Watchdog Timer status needs to be evaluated, the following procedures could for example be used.

**Variable activation code disabled**

- if WDTAnWDTE = $2C_H$: Watchdog was not activated
- if WDTAnWDTE = $AC_H$: Watchdog was activated

**Variable activation code enabled**

- if WDTAnREF ≠ 0: Watchdog was activated
- if WDTAnREF = 0:
  - if WDTAnEVAC = $2C_H$: Watchdog was not activated
  - if WDTAnEVAC ≠ $2C_H$: Watchdog was activated

## 13.6 WDTA registers

This section contains a description of all registers of the WDTA.

### 13.6.1 WDTA registers overview

The WDTA is controlled and operated by the following registers:

**Table 13-10 WDTA register overview**

| Register name | Shortcut | Address |
|---|---|---|
| WDTA enable register | WDTAnWDTE | <WDTAn_base> + $0000_H$ |
| WDTA VAC enable register | WDTAnEVAC | <WDTAn_base> + $0004_H$ |
| WDTA reference value register | WDTAnREF | <WDTAn_base> + $0008_H$ |
| WDTA mode register | WDTAnMD | <WDTAn_base> + $000C_H$ |

**<WDTAn_base>** The base addresses <WDTAn_base> of the WDTAn is defined in the first section of this chapter under the key word “Register addresses”.

### 13.6.2 WDTA registers details

#### (1) WDTAnWDTE – WDTA enable register

This register is the WDTA start control and trigger register if the VAC function is not used (start-up option OPWDVAC = 0).

**WDTA trigger** Writing $AC_H$ to this register restarts the counter.
Refer to the section *"WDTA trigger"* for details.

The behaviour of this register depends on activation of the VAC function, refer to the table *"WDTAnWDTE behaviour"* below.

**Access** This register can be read/written in 8-bit units.

**Address** <WDTAn_base> + $0000_H$

**Initial Value** x010 $1100_B$.The initial value of the "x" bit depend on the start-up options OPWDEN, OPWDTPR, WDTATRTYP, OPWDRUN and OPWDVAC.
Refer to the table *"WDTAnRUN initial value"* below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| WDTAnRUN | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 13-11 WDTAnWDTE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | WDTAnRUN | Enables/disables the WDTAn:<br>0: WDTAn disabled<br>1: WDTAn enabled<br>Since the WDTA can not be stopped once it was started, this bit can only be cleared by a reset. |

**Initial value** WDTAnRUN is only valid if WDTA is enabled (OPWDEN = 1) and VAC is disabled (OPWDVAC = 0). In this case, the initial value of bit WDTAnRUN depends on other start-up options is listed below:

**Table 13-12 WDTAnRUN initial value**

| Start-up options | | Input signal | Start mode | Initial value of WDTAnRUN |
|---|---|---|---|---|
| OPWD TPR | OPWD RUN | WDTATRTYP | | |
| 0 | 0 | Ignored | Software trigger | 0 |
| 0 | 1 | Ignored | Automatic | 1 |
| 1 | Ignored | 0 | Software trigger | 0 |
| 1 | Ignored | 1 | Automatic | 1 |

The behaviour of WDTAnWDTE during read/write accesses depends on activation of the VAC mode, as shown in the table below.

**Table 13-13 WDTAnWDTE behaviour**

| OPWDVAC | WDTAnWDTE | | Remark |
|---|---|---|---|
| | Read | Write | |
| 0 | $AC_H$ | WDTA trigger $AC_H$[a] | VAC disabled: WDTAnWDTE enabled |
| 1 | $2C_H$ | ignored | VAC enabled: WDTAnWDTE disabled |

a) Any other write value will lead to an error detection.

### (2) WDTAnEVAC – WDTA enable VAC register

This register is the start control and trigger register if the VAC function is used (start-up option OPWDVAC = 1).

**WDTA trigger** Writing the correct activation code to this register restarts the counter. Refer to the section *"WDTA trigger"*.

The behaviour of this register depends on activation of the VAC function, refer to the table *"WDTAnEVAC behaviour"* below.

**Access** This register can be read/written in 8-bit units.

**Address** <WDTAn_base> + $0004_H$

**Initial Value** x010 $1100_B$.The initial value of the "x" bit depend on the start-up options OPWDEN, OPWDTPR, WDTATRTYP, OPWDRUN and OPWDVAC. Refer to the table *"WDTAnRUN initial value"* below.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| WDTAnEVAC7 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 13-14 WDTAnEVAC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | WDTAnEVAC7 | Enables/disables the WDTAn:<br>0: WDTAn disabled<br>1: WDTAn enabled<br>Since the WDTA can not be stopped once it was started, this bit can only be cleared by a reset. Thus even if bit 7 of the activation code is 0, the WDTA will not stop. |

**Initial value** WDTAnEVAC7 is only valid if WDTA is enabled (OPWDEN = 1) and VAC is enabled (OPWDVAC = 1). In this case, the initial value of bit WDTAnEVAC7 depends on other start-up options is listed below:

**Table 13-15 WDTAnRUN initial value**

| Start-up options | | Input signal | Start mode | Initial value of WDTAnEVAC7 |
|---|---|---|---|---|
| OPWD TPR | OPWD RUN | WDTATRTYP | | |
| 0 | 0 | Ignored | Software trigger | 0 |
| 0 | 1 | Ignored | Automatic | 1 |
| 1 | Ignored | 0 | Software trigger | 0 |
| 1 | Ignored | 1 | Automatic | 1 |

The behaviour of WDTAnEVAC during read/write accesses depends on activation of the VAC mode, as shown in the table below.

**Table 13-16 WDTAnEVAC behaviour**

| OPWDVAC | WDTAnEVAC | | Remark |
|---|---|---|---|
| | Read | Write | |
| 0 | $2C_H$ | ignored | VAC disabled: WDTAnEVAC disabled |
| 1 | last written VAC | WDTA trigger VAC[a] | VAC enabled: WDTAnEVAC enabled |

a) Any other write value will lead to an error detection.

### (3) WDTAnREF – WDTA reference value register

This register contains the reference value for calculating the activation code of the VAC function. It is automatically updated after every trigger operation. Refer to the section *"WDTA trigger"*.

If VAC is disabled (OPWDVAC = 0), reading this register returns $00_H$.

**Access** This register can be read in 8-bit units.

**Address** <WDTAn_base> + $0008_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| WDTAnREF[7:0] | | | | | | | |
| R | R | R | R | R | R | R | R |

**Table 13-17 WDTAnREF register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | WDTAnREF[7:0] | Reference value for activation code calculation. |

### (4) WDTAnMD – WDTA mode register

This register specifies the overflow interval time, the 75% interrupt output mode, the error mode, and the open window size.

It can be updated only once after reset release and before the first trigger.

**Caution** The updated WDTAnMD value is effective after the next WDTA trigger.

Updating this register after the WDTA has been started leads to error detection, but the read value of this register can be written without generating an error.

**Access** This register can be read/written in 8-bit units.

**Address** <WDTAn_base> + $000C_H$

**Initial Value** $0xxx\ x1xx_B$.The initial value of the “x” bits depend on the start-up options OPWDOVF[2:0], OPWDINT and OPWDWS[1:0].
Refer to *“WDTA Start-up Options”* in the first section of this chapter.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | WDTAnOVF[2:0] | | | WDTAnWIE | WDTAnERM | WDTAnWS[1:0] | |
| R/W[a] | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

a) Writing to this bit is ignored, reading returns 0.

**Table 13-18 WDTAnMD register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 6 to 4 | WDTAnOVF[2:0] | Selects the count clock and thus the overflow interval time:<br>**WDTAnOVF2 / WDTAnOVF1 / WDTAnOVF0 / Count clock**<br>0 / 0 / 0 / WDTATCKI / $2^9$<br>0 / 0 / 1 / WDTATCKI / $2^{10}$<br>0 / 1 / 0 / WDTATCKI / $2^{11}$<br>0 / 1 / 1 / WDTATCKI / $2^{12}$<br>1 / 0 / 0 / WDTATCKI / $2^{13}$<br>1 / 0 / 1 / WDTATCKI / $2^{14}$<br>1 / 1 / 0 / WDTATCKI / $2^{15}$<br>1 / 1 / 1 / WDTATCKI / $2^{16}$<br>The overflow interval time is calculated from the count clock:<br>Overflow interval time = 1 / count clock frequency<br><br>The reset values of WDTAnOVF[2:0] depend on start-up option OPWDOVF[2:0]. |
| 3 | WDTAnWIE | Enables/disables the 75% interrupt request INTWDTn:<br>0: INTWDTn disabled<br>1: INTWDTn enabled<br>The reset value of WDTAnWIE depends on start-up option OPWDINT. |

**Table 13-18 WDTAnMD register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>2</td><td>WDTAnERM</td><td>Specifies the error mode:<br>0: NMI request mode<br>1: Reset mode</td></tr>
<tr><td>1 to 0</td><td>WDTAnWS[1:0]</td><td>Selects the open window size:
<table>
<tr><th>WDTAnWS1</th><th>WDTAnWS0</th><th>Open window size</th></tr>
<tr><td>0</td><td>0</td><td>25%</td></tr>
<tr><td>0</td><td>1</td><td>50%</td></tr>
<tr><td>1</td><td>0</td><td>75%</td></tr>
<tr><td>1</td><td>1</td><td>100%</td></tr>
</table>
The reset values of WDTAnWS[1:0] depend on start-up option OPWDWS[1:0].</td></tr>
</table>

# Chapter 14 Timer Array Unit B (TAUB)

This chapter contains a generic description of the Timer Array Unit B (TAUB).

The first section describes all V850E2/Fx4-G specific properties, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 14.1 V850E2/Fx4-G TAUB Features

**Instances**

This microcontroller has the following number of instances of the Timer Array Unit B.

**Table 14-1 Instances of TAUB**

| Timer Array Unit B | V850E2/FF4-G<br>V850E2/FG4-G |
|---|---|
| Instance | 1 |
| Name | TAUB0 |

**Instances index n**

Throughout this chapter, the individual instances of a Timer Array Unit B is identified by the index “n” (n = 0), for example, TAUBnTOM for the TAUBn channel output mode register.

**Channel index m**

The Timer Array Unit B has 16 channels. Throughout this chapter, the individual channels are identified by the index “m” (m = 0 to 15), thus a certain channel is denoted as CHm.
The even numbered channels (m = 0, 2, 4, 6, 8, 10, 12, 14) are denoted as CHm_even.
The odd numbered channels (m = 1, 3, 5, 7 , 9, 11, 13, 15) are denoted as CHm_odd.

**Register addresses**

All TAUBn register addresses are given as address offsets to the individual base address <TAUBn_base>.
The base address <TAUBn_base> of each TAUBn is listed in the following table:

**Table 14-2 Register base addresses <TAUBn_base>**

| TAUBn instance | <TAUBn_base> address |
|---|---|
| TAUB0 | FF80 $8000_H$ |

**Clock supply** All Timer Array Units B provide one clock input:

**Table 14-3 TAUBn clock supply**

| TAUBn instance | TAUBn clock | Connected to |
|---|---|---|
| TAUB0 | PCLK | Clock Controller CKSCLK_006 |

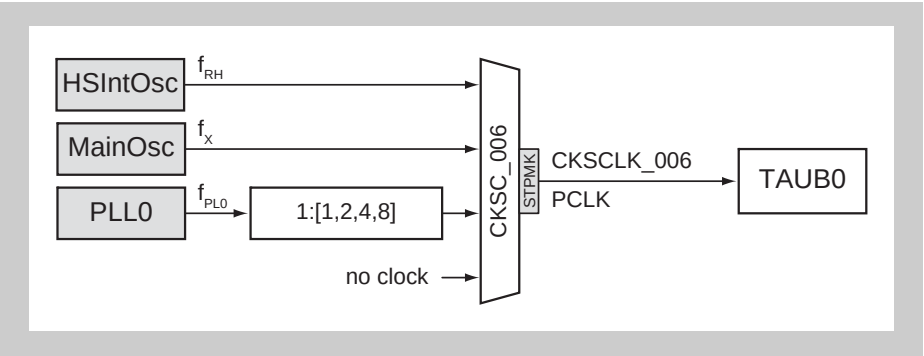



**Figure 14-1 TAUB clock supply**

**Interrupts and DMA** The Timer Array Unit B can generate the following interrupt and DMA requests:

**Table 14-4 TAUBn interrupt and DMA requests (1/2)**

| TAUBn signals | Function | Connected to |
|---|---|---|
| **TAUB0:** | | |
| INTTAUB0I0 | Channel 0 interrupt | Interrupt Controller INTTAUB0I0[a] |
| INTTAUB0I1 | Channel 1 interrupt | Interrupt Controller INTTAUB0I1[a] |
| INTTAUB0I2 to INTTAUB0I3 | Channel 2 to 3 interrupt | Interrupt Controller INTTAUB0I2[a] to INTTAUB0I3[a] |
| INTTAUB0I4 | Channel 4 interrupt | Interrupt Controller INTTAUB0I4[a] |
| INTTAUB0I5 | Channel 5 interrupt | Interrupt Controller INTTAUB0I5[a] |
| INTTAUB0I6 to INTTAUB0I7 | Channel 6 to 7 interrupt | Interrupt Controller INTTAUB0I6[a] to INTTAUB0I7[a] |
| INTTAUB0I8 | Channel 8 interrupt | Interrupt Controller INTTAUB0I8[a] DMA Controller trigger 12 |
| INTTAUB0I9 | Channel 9 interrupt | Interrupt Controller INTTAUB0I9[a] DMA Controller trigger 13 |
| INTTAUB0I10 | Channel 10 interrupt | Interrupt Controller INTTAUB0I10[a] DMA Controller trigger 14 |
| INTTAUB0I11 | Channel 11 interrupt | Interrupt Controller INTTAUB0I10[a] DMA Controller trigger 15 |
| INTTAUB0I12 | Channel 12 interrupt | Interrupt Controller INTTAUB0I12[a] DMA Controller trigger 16 |
| INTTAUB0I13 | Channel 13 interrupt | Interrupt Controller INTTAUB0I13[a] DMA Controller trigger 17 |

**Table 14-4 TAUBn interrupt and DMA requests (2/2)**

| TAUBn signals | Function | Connected to |
|---|---|---|
| INTTAUB0I14 | Channel 14 interrupt | Interrupt Controller INTTAUB0I14[a]<br>DMA Controller trigger 18 |
| INTTAUB0I15 | Channel 15 interrupt | Interrupt Controller INTTAUB0I15[ab]<br>DMA Controller trigger 19 |

a) These interrupts can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

b) These signal can be used to as a trigger source to start the A/D Converter. Refer to the section *"H/W Trigger Expansion"* in the chapter *"A/D Converter (ADAA)"*.

### TAUB H/W reset

The Time Array Units B and their registers are initialized by the following reset signal:

**Table 14-5 TAUBn reset signal**

| TAUBn | Reset signal |
|---|---|
| TAUBn | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

**I/O signals** The I/O signals of the Timer Array Unit B are listed in the following table.

**Table 14-6 TAUBn I/O signals**

| TAUB signal | Function | Connected to |
|---|---|---|
| **TAUB0:** | | |
| TAUB0TTIN0 | Channel 0 input | FCN0 TSOUT or port URTE10RX or TAUJ0 INTTAUJ0I3[b] |
| TAUB0TTIN1 | Channel 1 input | Port TAUB0I1[a] or FCN1 TSOUT or port URTE11RX or TAUB0TTIN0[b] |
| TAUB0TTIN2 | Channel 2 input | Port TAUB0I2[a] or port URTE2RX[b] |
| TAUB0TTIN3 | Channel 3 input | Port TAUB0I3[a] or TAUB0TTIN2[b] |
| TAUB0TTIN4 | Channel 4 input | Port TAUB0I4[a] |
| TAUB0TTIN5 | Channel 5 input | Port TAUB0I5[a] or TAUB0TTIN4[b] |
| TAUB0TTIN6 | Channel 6 input | Port TAUB0I6[a] |
| TAUB0TTIN7 | Channel 7 input | Port TAUB0I7[a] or TAUB0TTIN6[b] |
| TAUB0TTIN8 | Channel 8 input | Port TAUB0I8[a] |
| TAUB0TTIN9 | Channel 9 input | Port TAUB0I9[a] or TAUB0TTIN8[b] |
| TAUB0TTIN10 | Channel 10 input | Port TAUB0I10[a] or FCN0 TSOUT[b] |
| TAUB0TTIN11 | Channel 11 input | Port TAUB0I11[a] or FCN1 TSOUT or TAUB0TTIN10[b] |
| TAUB0TTIN12 | Channel 12 input | Port TAUB0I12[a] |
| TAUB0TTIN13 | Channel 13 input | Port TAUB0I13[a] or TAUB0TTIN12[b] |
| TAUB0TTIN14 | Channel 14 input | Port TAUB0I14[a] |
| TAUB0TTIN15 | Channel 15 input | Port TAUB0I15[a] or TAUJ0 INTTAUJ0I3 or TAUB0TTIN14[b] |
| TAUB0TTOUT0 | Channel 0 output | not connected |
| TAUB0TTOUT1 to TAUB0TTOUT15 | Channel 15 output | Port TAUB0IO1 to TAUB0O15 |

a) These input signals are passed through a noise filter, refer to the section *"Port Filters"* in the chapter *"Port Functions"*.
b) Refer to the section *"TAUB Input Selections"* below.

## 14.2 TAUB0 Input Selections

The TAUB0 has several options to connect its input signals:

- FCN0, FCN1, FCN2, FCN3, FCN4, FCN5 time stamp output signals TSOUT for timing measurements
- URTE2, URTE10, URTE11 data receive signals URTEnRX for baud rate measurement
- odd numbered TAUB0 input signals TAUB0TTINm can be connected to even numbered input signals
- TAUJ0 interrupt INTTAUJ0I3 for timing measurements

The following figure depicts the TAUB0 input selection scheme:

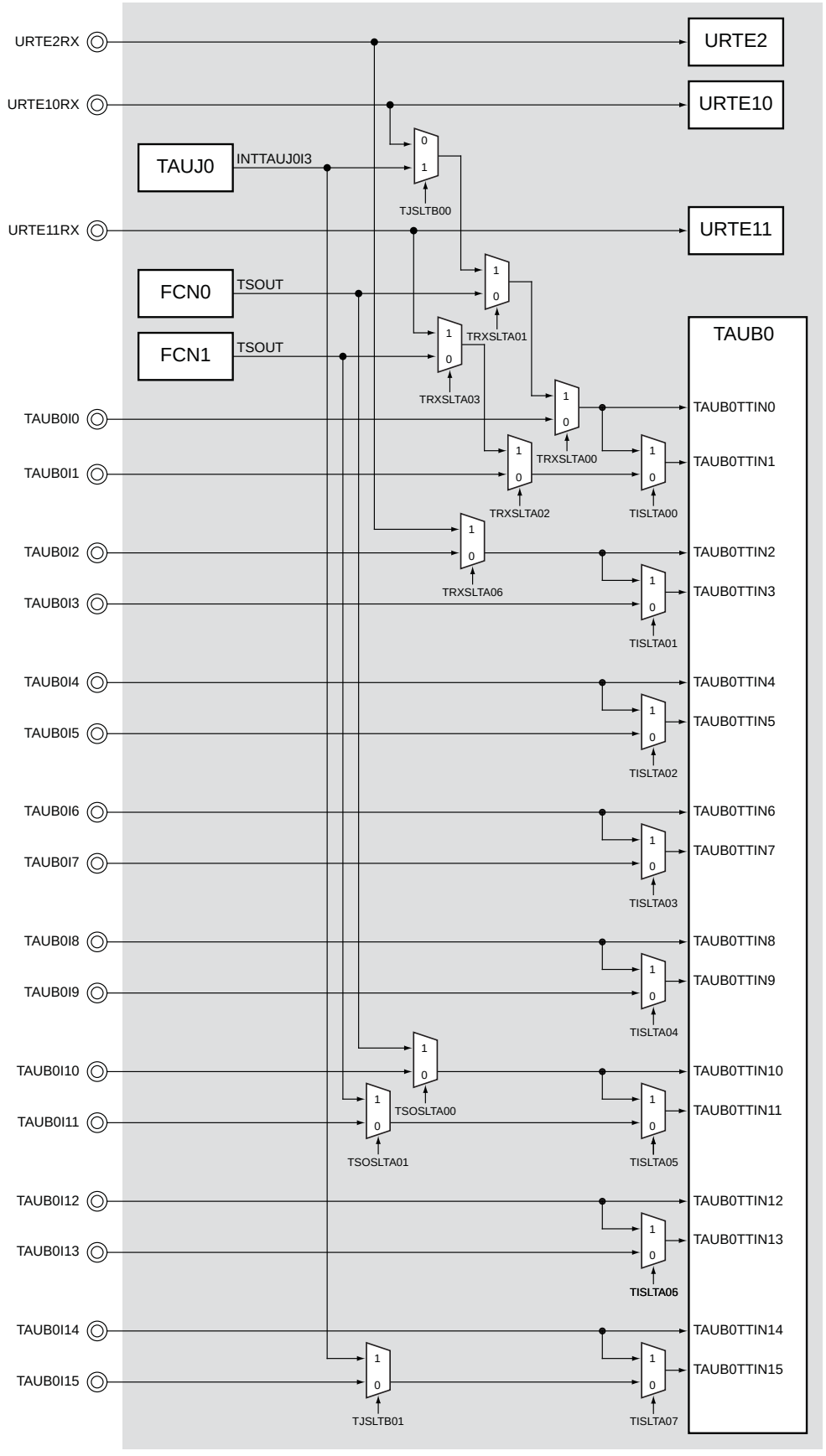



**Figure 14-2 TAUB0 input selector scheme**

The following tables show the optional inputs to several TAUB0 inputs:

**Table 14-7 TAUB0 input selections - TAUB0TTIN0 to TAUB0TTIN2**

| TAUB0 input | Input options | Selection control | | |
|---|---|---|---|---|
| | | TISLTA0 register bit | TRXSLTA0 register bits | TJSLTB0 register bits |
| TAUB0TTIN0 | FCN0 TSOUT (CAN I/F 0 time stamp output) | X | TRXSLTA0[1:0] = $01_B$ | X |
| | Port URTE10RX (URTE10 data receive signal) | | TRXSLTA0[1:0] = $11_B$ | TJSLTB00 = 0 |
| | TAUJ0 INTTAUJ0I3 (TAUJ0 channel 3 interrupt) | | | TJSLTB00 = 1 |
| TAUB0TTIN1 | Port TAUB0I1 | TISLTA00 = 0 | TRXSLTA0[3:2] = $x0_B$ | X |
| | FCN1 TSOUT (CAN I/F 1 time stamp output) | | TRXSLTA0[3:2] = $01_B$ | |
| | Port URTE11RX (URTE11 data receive signal) | | TRXSLTA0[3:2] = $11_B$ | |
| TAUB0TTIN2 | Port TAUB0I12 | X | TRXSLTA06 = 0 | X |
| | Port URTE2RX (URTE2 data receive signal) | | TRXSLTA06 = 1 | |

**Table 14-8 TAUB0 input selections - TAUB0TTIN3 to TAUB0TTIN9**

| TAUB0 input | Input options | Selection control TISLTA0 register bits |
|---|---|---|
| TAUB0TTIN3 | Port TAUB0I3 | TISLTA01 = 0 |
| | Input to TAUB0TTIN2 | TISLTA01 = 1 |
| TAUB0TTIN4 | Port TAUB0I4 | X |
| TAUB0TTIN5 | Port TAUB0I5 | TISLTA02 = 0 |
| | Input to TAUB0TTIN4 | TISLTA02 = 1 |
| TAUB0TTIN6 | Port TAUB0I6 | X |
| TAUB0TTIN7 | Port TAUB0I7 | TISLTA03 = 0 |
| | Input to TAUB0TTIN6 | TISLTA03 = 1 |
| TAUB0TTIN8 | Port TAUB0I8 | X |
| TAUB0TTIN9 | Port TAUB0I9 | TTISLTA04 = 0 |
| | Input to TAUB0TTIN8 | TTISLTA04 = 1 |

**Table 14-9 TAUB0 input selections - TAUB0TTIN10 to TAUB0TTIN15**

| TAUB0 input | Input options | Selection control | | |
|---|---|---|---|---|
| | | TISLTA0 register bit | TSOSLTA0 register bits | TJSLTB0 register bits |
| TAUB0TTIN10 | Port TAUB0I10 | X | TSOSLTA00 = 0 | X |
| | FCN0 TSOUT (CAN I/F 0 time stamp output) | | TSOSLTA00 = 1 | |
| TAUB0TTIN11 | Port TAUB0I11 | TISLTA05 = 0 | TSOSLTA01 = 0 | X |
| | FCN1 TSOUT (CAN I/F 1 time stamp output) | | TSOSLTA01 = 1 | |
| | Input to TAUB0TTIN10 | TISLTA05 = 1 | X | |
| TAUB0TTIN12 | Port TAUB0I12 | X | TSOSLTA02 = 0 | X |
| | FCN2 TSOUT (CAN I/F 2 time stamp output) | X | TSOSLTA02 = 1 | X |
| TAUB0TTIN13 | Port TAUB0I13 | TISLTA06 = 0 | X | X |
| | Input to TAUB0TTIN12 | TISLTA06 = 1 | | |
| | FCN3 TSOUT (CAN I/F 3 time stamp output) | X | TSOSLTA03 = 1 | |
| TAUB0TTIN14 | Port TAUB0I14 | X | TSOSLTA04 = 0 | X |
| | FCN4 TSOUT (CAN I/F 4 time stamp output) | X | TSOSLTA04 = 1 | X |
| TAUB0TTIN15 | Port TAUB0I15 | TISLTA07 = 0 | X | TJSLTB01 = 0 |
| | TAUJ0 INTTAUJ0I3 (TAUJ0 channel 3 interrupt) | | | TJSLTB01 = 1 |
| | Input to TAUB0TTIN14 | TISLTA07 = 1 | X | X |
| | FCN5 TSOUT (CAN I/F 5 time stamp output) | X | TSOSLTA05 = 1 | X |

**(1) TISLTA0 - TAUB0 odd inputs selection register**

This register selects the input signals to the odd TAUB0 inputs.

**Access** This register can be read/written in 8-bit units.

**Address** FF77 $1000_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TISLTA07 | TISLTA06 | TISLTA05 | TISLTA04 | TISLTA03 | TISLTA02 | TISLTA01 | TISLTA00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-10 TISLTA0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | TISLTA07 | Selection of TAUB0TTIN15:<br>0: Port TAUB0I15 or TAUJ0 interrupt INTTAUJ0I3[a]<br>1: TAUB0TTIN14 input |
| 6 | TISLTA06 | Selection of TAUB0TTIN13:<br>0: Port TAUB0I13<br>1: TAUB0TTIN12 input |
| 5 | TISLTA05 | Selection of TAUB0TTIN11:<br>0: Port TAUB0I11 or FCN1 time stamp TSOUT[b]<br>1: TAUB0TTIN10 input |
| 4 | TISLTA04 | Selection of TAUB0TTIN9:<br>0: Port TAUB0I9<br>1: TAUB0TTIN8 input |
| 3 | TISLTA03 | Selection of TAUB0TTIN7:<br>0: Port TAUB0I7<br>1: TAUB0TTIN6 input |
| 2 | TISLTA02 | Selection of TAUB0TTIN5:<br>0: Port TAUB0I5<br>1: TAUB0TTIN4 input |
| 1 | TISLTA01 | Selection of TAUB0TTIN3:<br>0: Port TAUB0I3<br>1: TAUB0TTIN2 input |
| 0 | TISLTA00 | Selection of TAUB0TTIN1:<br>0: Port TAUB0I1 or FCN1 time stamp TSOUT or port URTE11RX[c]<br>1: TAUB0TTIN0 input |

a) Refer also to the TJSLTB0 register description.
b) Refer also to the TSOSLTA0 register description.
c) Refer also th the TRXSLTA0 register description.

### (2) TRXSLTA0 - TAUB0 receive input selection register

This register selects the input signals to several TAUB0 inputs out of signals related to other functional modules (FCN, URTE).

**Access** This register can be read/written in 8-bit units.

**Address** FF77 $1004_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | TRXSLTA06 | 0 | 0 | TRXSLTA0[3:2] | | TRXSLTA0[1:0] | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-11 TRXSLTA0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 6 | TRXSLTA06 | Selection of TAUB0TTIN2:<br>TRXSLTA06: $0_B$ → Port TAUB0I2<br>TRXSLTA06: $1_B$ → Port URTE2RX |
| 3, 2 | TRXSLTA0[3:2] | Selection of TAUB0TTIN1 selector TISLTA0.TISLTA00 input:<br>TRXSLTA0[3:2]: $00_B$, $10_B$ → Port TAUB0I1<br>TRXSLTA0[3:2]: $01_B$ → FCN1 time stampTSOUT<br>TRXSLTA0[3:2]: $11_B$ → Port URTE11RX |
| 1, 0 | TRXSLTA0[1:0] | Selection of TAUB0TTIN0:<br>TRXSLTA0[1:0]: $01_B$ → FCN0 time stamp TSOUT<br>TRXSLTA0[1:0]: $11_B$ → Port URTE10RX or TAUJ0 interrupt TAUJ0I3[a] |

a) Refer also the the TJSLTB0 register description.

### (3) TSOSLTA0 - TAUB0 input selection register

This register selects the input signals to several TAUB0 inputs out of signals related to other functional blocks (FCN, URTE).

**Access** This register can be read/written in 8-bit units.

**Address** FF77 2014$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | TSOSLTA05 | TSOSLTA04 | TSOSLTA03 | TSOSLTA02 | TSOSLTA01 | TSOSLTA00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-12 TSOSLTA0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 5 | TSOSLTA05 | Selection of TAUB0TTIN15 selector TISLTA0.TISLTA07 input:<br>TSOSLTA05: 0 = Port TAUB0I15; 1 = FCN5 time stamp TSOUT |
| 4 | TSOSLTA04 | Selection of TAUB0TTIN14:<br>TSOSLTA04: 0 = Port TAUB0I14; 1 = FCN4 time stamp TSOUT |
| 3 | TSOSLTA03 | Selection of TAUB0TTIN13 selector TISLTA0.TISLTA06 input:<br>TSOSLTA03: 0 = Port TAUB0I13; 1 = FCN3 time stamp TSOUT |
| 2 | TSOSLTA02 | Selection of TAUB0TTIN12:<br>TSOSLTA02: 0 = Port TAUB0I12; 1 = FCN2 time stamp TSOUT |
| 1 | TSOSLTA01 | Selection of TAUB0TTIN11 selector TISLTA0.TISLTA05 input:<br>TSOSLTA01: 0 = Port TAUB0I11; 1 = FCN1 time stamp TSOUT |
| 0 | TSOSLTA00 | Selection of TAUB0TTIN12:<br>TSOSLTA00: 0 = Port TAUB0I12; 1 = FCN0 time stamp TSOUT |

### (4) TJSLTB0 - TAUB0 input selection register

This register selects the TAUJ0 interrupt signal INTTAUJ0I3 as an input signal to several TAUB0 inputs.

**Access** This register can be read/written in 8-bit units.

**Address** FF77 0418$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | TJSLTB01 | TJSLTB00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-13 TJOSLTB0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | TJSLTB01 | Selection of TAUB0TTIN15 selector TISLTA0.TISLTA07 input:<br>TJSLTB01: 0 – Port TAUB0I15; 1 – TAUJ0 interrupt INTTAUJ0I3 |
| 0 | TJSLTB00 | Selection of selector TRXSLTA0.TRXSLTA01 input:<br>TJSLTB00: 0 – Port URTE10RX; 1 – TAUJ0 interrupt INTTAUJ0I3 |

## 14.3 Functional Overview

**Features summary**

The TAUB has the following functions:

- 16 channels
- 16-bit counter and 16-bit data register per channel
- Independent channel operation
- Synchronous channel operation (master and slave operation)
- Generation of different types of output signal
- Counter can be triggered by external signal
- Interrupt generation

The following figure shows the main components of the TAUB:

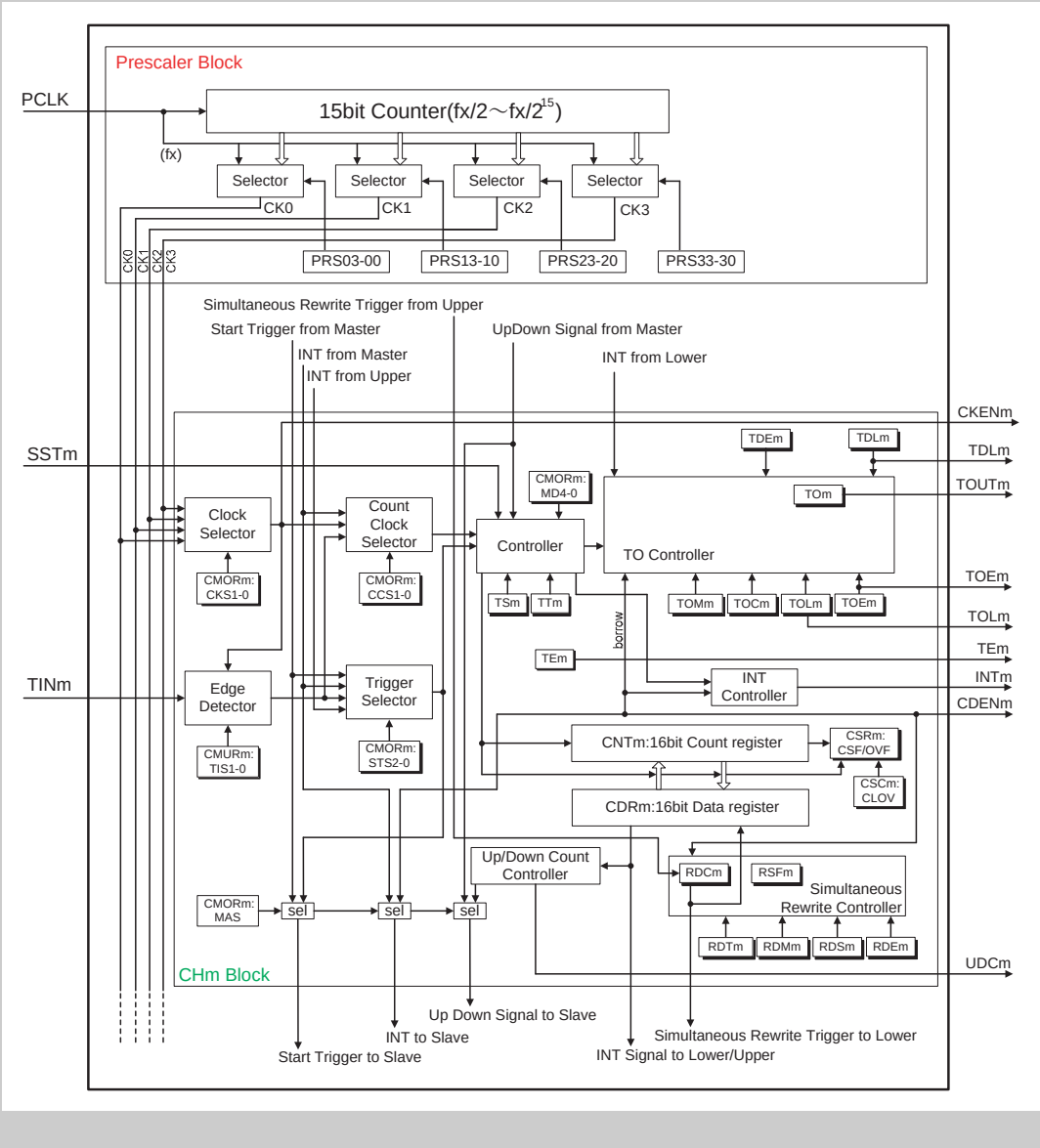



**Figure 14-3 Block diagram of the TAUB**

The prefix “TAUBn” has been omitted from the register names for the sake of clarity in the above figure.

### 14.3.1 Terms

In this chapter, the following terms are used:

- **Independent / synchronous channel operation**

  Independent or synchronous channel operation describes the dependency of channels on each other:

  – If a channel operates independent of all other channels, this is called independent channel operation.

  – If a channel operates depending on other channels, this is called synchronous channel operation.

- **Channel group**

  In synchronous channel operation, all channels that depend on each other are referred to as a “channel group”.

  A channel group has one master channel and one or more slave channels.

- **Operation mode**

  An operation mode can be selected for every channel m. The operation mode defines the *basic* operation and features of a channel.

  In synchronous channel operation, every channel in the channel group can operate in a different operation mode.

  Examples are “Capture Mode”, “Event Count Mode”, and “Interval Timer Mode”.

- **Channel output mode**

  The channel output mode defines the operation of TAUBnTTOUTm

  – of a single channel (independent output operation) or

  – of all channels in a channel group (synchronous output operation).

  Examples are “Independent Channel Output Mode” and “Synchronous Channel Output Mode with Dead Time”.

- **Channel operation function**

  The channel operation function defines the *complete* function and all features

  – of a single channel (independent channel operation) or

  – of all channels in a channel group (synchronous channel operation).

  It defines the operation mode, start and capture trigger, count clock, the real-time trigger generation, and the channel output mode.

  Examples are “Divider Function” and “Triangle PWM Output Function”

- **Upper / lower channel**

  Depending on the channel number m, a neighboring channel can be referred to as “upper” or “lower” channel:

  – Upper channel: Channel with a smaller channel number

  – Lower channel: Channel with a higher channel number

  Example:
  For channel 5, channel 3 is an upper channel and channel 9 is a lower channel.

## 14.4 Functional Description

The Timer Array Unit B is used to perform various count or timer operations and to output a signal which depends on the result of the operation. It contains one prescaler block for count clock generation and 16 channels, each equipped with a 16 bit counter TAUBnCNTm and a 16-bit data register TAUBnCDRm to hold the start or compare value of the counter.

It also contains several control and status registers.

**Independent and synchronous operation**

Every channel can operate in different operation modes, either independently or in combination with other channels (synchronously), i.e. multiple channels depend on each other with one master and one or more slave channels.

When a channel is operated independently, its operation mode and functions are not affected by those of other channels. When a channel is operated synchronously it is either a master or a slave. A master channel can have multiple slaves, and the state of one channel affects that of the other channels. For example, this means that one channel can control when another starts to count, is reset, etc.

The following describes the functional blocks:

**Prescaler block**

The prescaler block provides up to 4 clock signals (CK0 to CK3) that can be used as count clocks for all channels.

Count clocks CK0 to CK3 are derived from PCLK by a configurable prescaler division factor of $2^0$ to $2^{15}$.

**Clock and count clock selection**

For every channel, the count clock selector selects which of the following is used as the clock source:

- One of the clocks CK0 to CK3 (selected by the clock selector)
- INTTAUBnIm from master channel
- Edge detected TAUBnTTINm input signal

**Controller**

The controller controls the main operations of the counter:

- Operation mode (selected by bits TAUBnCMORm.MD[4:0])
- Counter start enable (TAUBnTS.TSm) and counter stop (TAUBnTT.TTm)

  When counter start is enabled, status flag TAUBnTE.TEm is set.
- Count direction (can be controlled by master channel)

**Trigger selector**

Depending on the selected operation mode, the counter starts automatically when it is enabled (TAUBnTE.TEm = 1), or it waits for an external start trigger signal. Any of the following signals can be used as the start trigger:

- Synchronous channel start trigger input TAUBnTSSTm
- TAUBnTTINm input valid edge
- INTTAUBnIm from the master or any upper channel
- Up/down output trigger signal TAUBnTUDSm of the master channel
- Dead-time output signal TAUBnTDL.TDLm of the TAUBnTTOUTm generation unit.

**Simultaneous rewrite controller**

Simultaneous rewrite control is a special function that can be used in synchronous operation modes. The data registers of all channels in a channel group can be rewritten at any time. The simultaneous rewrite controller

ensures that new data register values of all channels become effective at the same time.

**TAUBnTO Controller** The output control of every channel enables the generation of various output signal forms such as PWM signals or triangular waves.

**Signals** The TAUB has various input and output signals. A full list can be found in the first section of this chapter under the keyword “I/O signals”.

## 14.5 General Operating Procedure

The following lists the general operation procedure for the TAUBn:

After reset release, the operation of each channel is stopped. Clock supply is started and writing to each register is enabled. All circuits and registers of all channels are initialized. The control register of TAUBnTTOUTm is also initialized and outputs a low level.

1. Set the TAUBnTPS register to specify the clock frequency of CK0 to CK3.
2. Configure the desired TAUBn function:
   – Set the operation mode
   – Set the channel output mode
   – Set any other control bits
3. Enable the counter by setting the TAUBnTS.TSm bit to 1.
   The counter starts to count immediately, or when an appropriate trigger is detected, depending on the bit settings.
   The function is in operation.
4. If desired, and if possible for the configured function, stop the counter or perform a forced restart operation.
5. Stop the function by setting the TAUBnTT.TTm bit to 0.

**Note** A detailed description of the required control bits and the operation of the individual functions is given in *14.13 “Independent Channel Operation Functions” on page 501* and *14.18 “Synchronous Channel Operation Functions” on page 592* .

## 14.6 Operation Modes

The TAUB contains 12 operation modes. These determine the basic behavior of a channel, for example whether a timer counts up or down, whether the data register TAUBnCDRm acts as a compare register or stores the initial value of the counter, etc.

One operation mode can be set for each channel. It is specified using the TAUBnCMOR.MD[4:0] bits.

When choosing a function, these settings cannot be specified individually, but are grouped under the term operation mode. If a function uses multiple channels, the operation mode of each channel must be set correctly for the function to work correctly.

For more information about the operation modes required by each function, refer to the required function in *14.13 “Independent Channel Operation Functions” on page 501* and *14.18 “Synchronous Channel Operation Functions” on page 592* .

## 14.7 Concepts of Synchronous Channel Operation

In synchronous channel operation, multiple channels depend on each other, or are affected by changes in another channel. Therefore, several rules apply for the use of synchronous channel functions. These rules are detailed in *14.7.1 "Rules"*.

Two special features for synchronous channel operation are detailed in the following subchapters:

- *14.7.2 "Simultaneous start and stop of synchronous channel counters" on page 478*
- *14.8 "Simultaneous Rewrite" on page 479*

### 14.7.1 Rules

**Number of masters and slaves**

- Only even channels (CH0, CH2, CH4, …) can be set as master channels. Any channel apart from CH0 can be set as a slave channel.
- Only channels lower than the master channel can be set as slave channels, and several slave channels can be set for one master channel.
  Example: If CH2 is a master channel, CH3 and the lower channels (CH4, CH5, …) can be set as slave channels.
- If multiple master channels are used, slave channels cannot cross the master channels.
  Example: If CH0 and CH4 are master channels, CH1 to CH3 can be set as slave channels for CH0, but CH5 to CH7 cannot.

**Operation clock**

- The same operation clock must be set for the slave channel and the master channel. This is achieved using the TAUBnCMORm.CKS[1:0] bits of the slave and master channel.

The basic concepts of master/slave usage and operation clocks are illustrated in the following figure.

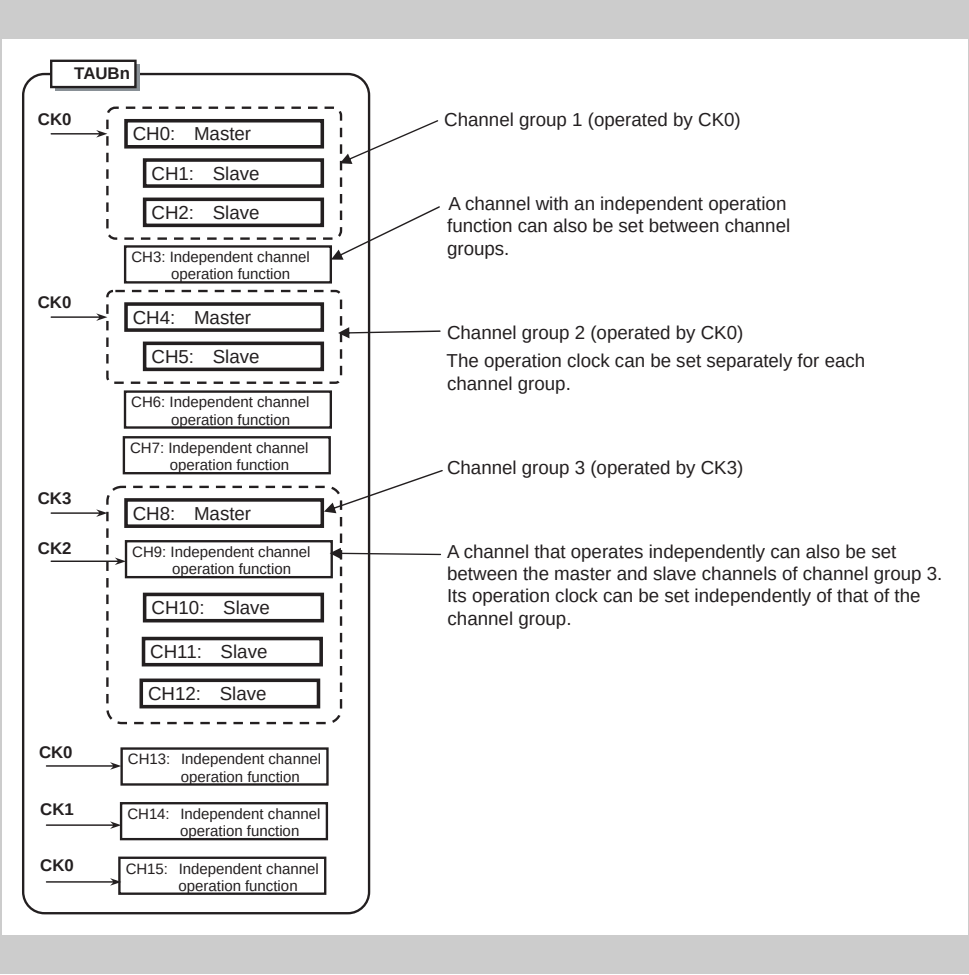



**Figure 14-4 Grouping of the channels and assignment of operation clocks**

**INTTAUBnIm, start trigger, and count clock**

- Master channels can transfer an interrupt request (INTTAUBnI), the start trigger, and the count clock to slave channels.
- Slave channels can use INTTAUBnI, the start trigger, and the count clock of the master channels but cannot transfer their INTTAUBnI, start trigger, or count clock to the lower channels.
- A master channel cannot use INTTAUBnI, the start trigger, or the count clock of the higher master channels.

### 14.7.2 Simultaneous start and stop of synchronous channel counters

Channels that are operated synchronously can be started and stopped simultaneously, both within a TAUB unit, and between TAUB units.

**(1) Simultaneous start and stop within a TAUB unit**

- To simultaneously start synchronized channels, the TAUBnTS.TSm bits of the channels must be set at the same time.
- To simultaneously stop synchronized channels, the TAUBnTT.TTm bits of the channels must be set at the same time.

Writing to the TAUBnTS.TSm bits sets the corresponding TAUBnTE.TEm bits to 1, enabling counting. TAUBnTS.TSm = 1 only enables the corresponding counter to start; the exact time that it starts depends on the operation mode.

**(2) Simultaneous start between TAUB units**

Counters in different TAUB units can also be started simultaneously if the corresponding counters are enabled before receiving the simultaneous trigger signal. The simultaneous start trigger register is then sent to the TAUBnTSSTm input.

## 14.8 Simultaneous Rewrite

### 14.8.1 Introduction

Simultaneous rewrite describes the ability to change the compare/start value and the output logic of multiple channels at the same time.

The corresponding data and control registers (TAUBnCDRm and TAUBnTOLm) can nevertheless be written at any time. The new value does not affect the counter operation or the output signal until simultaneous rewrite is triggered.

Simultaneous rewrite can be triggered by:

- The counter on the master channel or upper channel (depending on the selected operation mode) reaching a certain value
- INTTAUBnI being issued on the upper channel specified by TAUBnRDC.RDCm

There are four methods for simultaneous rewrite. These are listed in the following table, along with how to specify them and when they cause simultaneous rewrite to be triggered.

**Table 14-14 Simultaneous rewrite methods and when they are triggered**

| Method | Simultaneous rewrite triggered when | TAUBnRDE.RDEm | TAUBnRDS.RDSm | TAUBnRDM.RDMm |
|---|---|---|---|---|
| — | No simultaneous rewrite | 0 | 0 | 0 |
| A | The master channel (re)starts counting | 1 | 0 | 0 |
| B | The slave channel starts counting down at the upper peak of a triangular cycle | 1 | 0 | 1 |
| C1 | INTTAUBnIm is generated on an upper channel specified by TAUBnRDC.RDCm | 1 | 1 | 0 |
| TOLm | For TOLm, the following table shows whether TOLm can be rewritten during operation. The TOLm rewrite method is the same as that of CDRn. | | | |

The following table lists which of these four methods is available for each channel operation function. For more information about the individual channel operation functions, see the corresponding sections in *14.13 “Independent Channel Operation Functions” on page 501* and *14.18 “Synchronous Channel Operation Functions” on page 592* .

**Table 14-15 Channel operation functions and methods they use (1/2)**

| Function | A | B | C1 | TOLm |
|---|---|---|---|---|
| Simultaneous Rewrite Trigger Output Function Type 1 | | | X | |
| PWM Output Function | X | | X | X |
| One-Shot Pulse Output Function | X | | | |
| Delay Pulse Output Function | X | | | |
| Triangle PWM Output Function | | X | X | X |

**Table 14-15 Channel operation functions and methods they use (2/2)**

| Function | A | B | C1 | TOLm |
|---|---|---|---|---|
| Triangle PWM Output Function with Dead Time | | X | X | |
| AD Conversion Trigger Output Function Type 1 | X | | X | |
| AD Conversion Trigger Output Function Type 2 | | X | X | |

### 14.8.2 How to control simultaneous rewrite

The following figure shows the general procedure for simultaneous rewrite. The three main blocks (Initial settings, Start counter & count operation, and Simultaneous rewrite) are explained afterwards.

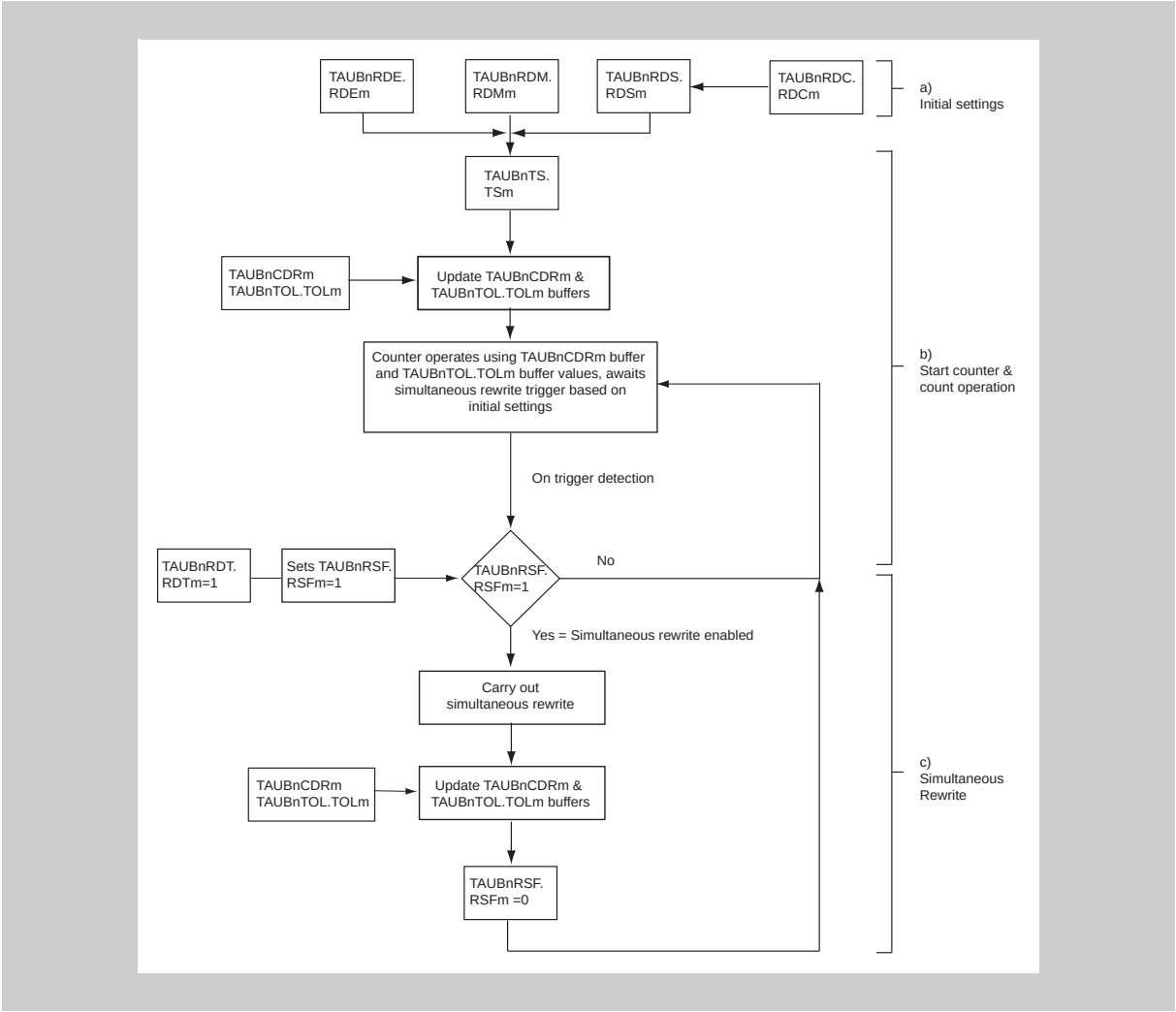



**Figure 14-5 General procedure for simultaneous rewrite**

#### (1) Initial settings

- To enable simultaneous rewrite in channel m, set TAUBnRDE.RDEm = 1
- To select the type of simultaneous rewrite, set TAUBnRDM.RDMm and TAUBnRDS.RDSm according to the values in *Table 14-14 “Simultaneous rewrite methods and when they are triggered” on page 479*
- To select which upper channel is monitored for the simultaneous rewrite trigger use TAUBnRDC.RDCm (prerequisite: TAUBnRDS.RDSm is set to upper channel)

**(2) Start counter and count operation**

- To start all the TAUBnCNTm counters in the channel group, set the corresponding TAUBnTS.TSm bits to 1. TAUBnTOL.TOLm and the values in the data registers (TAUBnCDRm) are written to the corresponding TAUBnTOL.TOLm buffer (TAUBnTOL.TOLm buf) and data buffer registers (TAUBnCDRm buf) and the counters start.
- Setting the reload data trigger bit (TAUBnRDT.RDTm) to 1 sets the reload flag (TAUBnRSF.RSFm) to 1, enabling simultaneous rewrite. TAUBnRDT.RDTm then immediately returns to 0, but TAUBnRSF.RSFm remains at 1 until simultaneous rewrite has taken place.
- When the specified trigger for simultaneous rewrite is detected, the TAUBnRSF.RSFm bit is checked to see if simultaneous rewrite is enabled (TAUBnRSF.RSFm = 1). If it is, simultaneous rewrite is carried out. Otherwise the value of the TAUBnRSF.RSFm bit is re-evaluated the next time the trigger is detected.

**(3) Simultaneous rewrite**

- When the simultaneous rewrite trigger is detected and simultaneous rewrite is enabled (TAUBnRSF.RSFm = 1), the current values of the data registers are copied to their buffers. These values are then written to the corresponding counters and the values are applied the next time the counter starts or restarts.
- The TAUBnRSF.RSFm bit is set to 0, and the system awaits the next simultaneous rewrite trigger.

### 14.8.3 Other general rules of simultaneous rewrite

The following rules also apply:

- TAUBnRDE.RDEm, TAUBnRDS.RDSm, TAUBnRDM.RDMm, and TAUBnRDC.RDCm cannot be changed while the counter is in operation (TAUBnTE.TEm = 1).
- TAUBnTOL.TOLm can only be rewritten during operation when in PWM output function or triangle PWM output function. For all other output functions, TAUBnTOL.TOLm must be written before the counter starts. If it is rewritten in another function, TAUBnTTOUTm outputs an invalid wave.
- When an upper channel is used as the channel issuing the simultaneous rewrite trigger (TAUBnRDS.RDSm = 1), the TAUBnRDC.RDCm bit controls all the lower channels. This means that if the TAUBnRDC.RDCm bits of CH2 and CH7 are set to 1 and the TAUBnRDC.RDCm bits of other channels are set to 0, CH2 and CH7 serve as simultaneous rewrite trigger generation channels. CH2 controls the lower channels CH3 to CH6, and CH7 controls the lower channels CH8 to CH15.
- If simultaneous rewrite is enabled and an upper channel is selected for the simultaneous rewrite trigger (TAUBnRDE.RDEm and TAUBnRDS.RDSm = 1) but no upper channel is set (TAUBnRDC.RDC[15:0] = 0), simultaneous rewrite cannot take place.

### 14.8.4 Types of simultaneous rewrite

In the following section the four simultaneous rewrite methods are explained using timing diagrams.

**(1) Simultaneous rewrite when the master channel (re)starts counting (method A)**

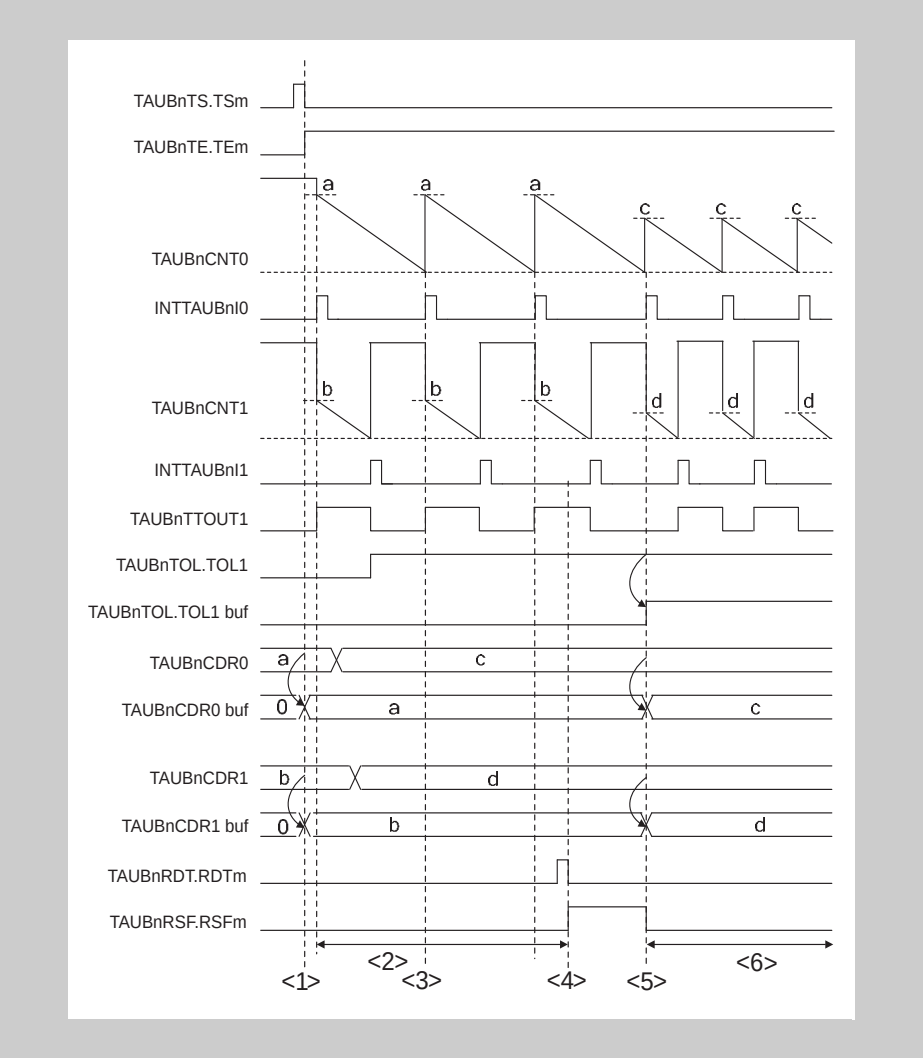



**Figure 14-6 Simultaneous rewrite when the master channel (re)starts counting**

Setup:

- CH0 is the master channel, counting down, CH1 represents an arbitrary slave channel, and simultaneous rewrite method A is applied.

Description:

1. When the counter starts, the value of TAUBnCDRm is copied to the TAUBnCDRm buffer and the value of TAUBnTOL.TOLm is copied to the TAUBnTOL.TOLm buffer. The TAUBnCDRm buffer value is written to the counter.
2. The TAUBnCDRm and TAUBnTOL.TOLm registers can be written at any time, but the values do not affect the counter as the counter reads the buffer values.
3. CH0 restarts counting, but simultaneous rewrite does not occur because it is disabled (TAUBnRSF.RSFm = 0).
4. The reload data trigger bit (TAUBnRDT.RDTm) is set to 1 which sets the status flag (TAUBnRSF.RSFm = 1), enabling simultaneous rewrite.
5. Simultaneous rewrite is triggered by counter TAUBnCNT0 starting to count down. The TAUBnCDRm value is written to the TAUBnCDRm buffer and the TAUBnTOL.TOLm value is written to the TAUBnTOL.TOLm buffer.
   – The counter starts to count down from the value in the TAUBnCDRm buffer and the TAUBnRSF.RSFm bit is reset to 0.
   – The output logic specified by TAUBnTOL.TOLm becomes effective.
6. The counters count down and await the next simultaneous rewrite trigger. The values of TAUBnCDRm and TAUBnTOL.TOLm can be changed again.

### (2) Simultaneous rewrite at the peak of a triangular cycle of the slave channel (method B)

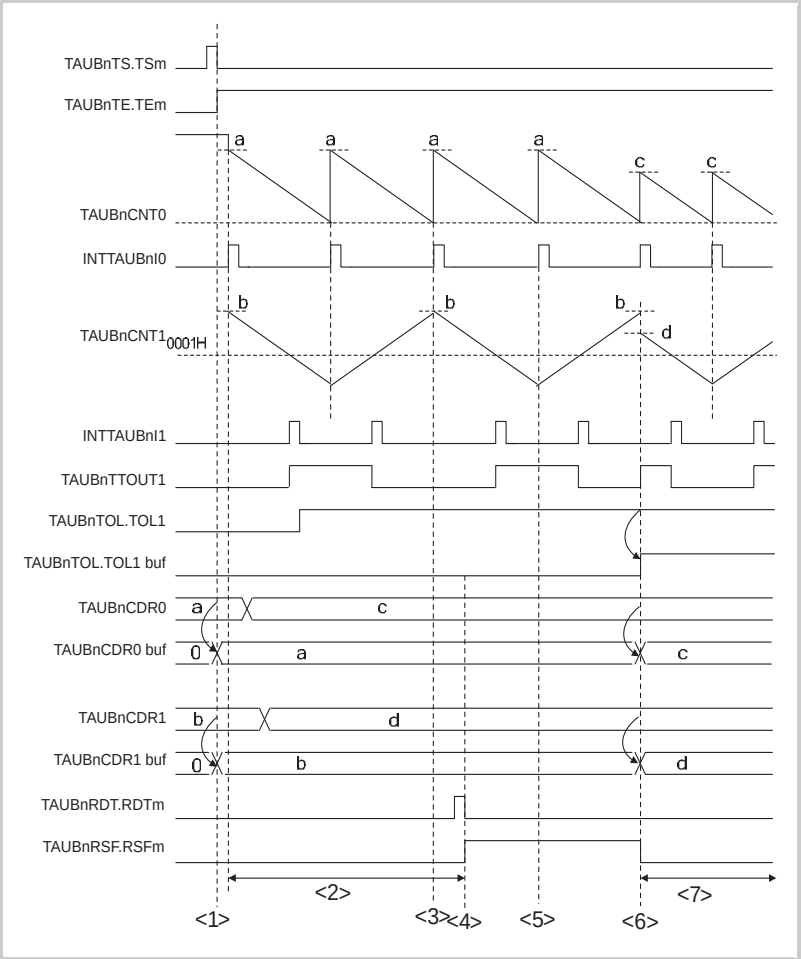



**Figure 14-7 Simultaneous rewrite at the peak of a triangular cycle of the slave channel**

Setup:

- CH0 is the master channel, counting up and down, CH1 represents an arbitrary slave channel, and simultaneous rewrite method B is applied.

Description:

1. When the counter starts, the value of TAUBnCDRm is copied to the TAUBnCDRm buffer. The buffer value is written to the counter.
2. The TAUBnCDRm and TAUBnTOL registers can be written at any time, but the values do not affect the counter as the counter reads the buffer values.
3. Simultaneous rewrite does not occur because it is disabled (TAUBnRSF.RSFm = 0).
4. The reload data trigger bit (TAUBnRDT.RDTm) is set to 1 which sets the status flag (TAUBnRSF.RSFm = 1), enabling simultaneous rewrite.
5. Even though simultaneous rewrite is enabled, it does not take place because it is only triggered by the slave channel starting to count down at an upper peak.
6. Simultaneous rewrite is triggered; the TAUBnCDRm value is written to the TAUBnCDRm buffer, the TAUBnTOL.TOLm value is written to the TAUBnTOL.TOLm buffer.
   – The counters start to count down from the value in the TAUBnCDRm buffer and the TAUBnRSF.RSFm bit is reset to 0.
   – The output logic specified by TAUBnTOL.TOLm becomes effective.
7. The counters count down and await the next simultaneous rewrite trigger. The values of TAUBnCDRm and TAUBnTOL.TOLm can be changed again.

**(3) Simultaneous rewrite when INTTAUBnIm is generated on an upper channel specified by TAUBnRDC.RDCm (method C1)**

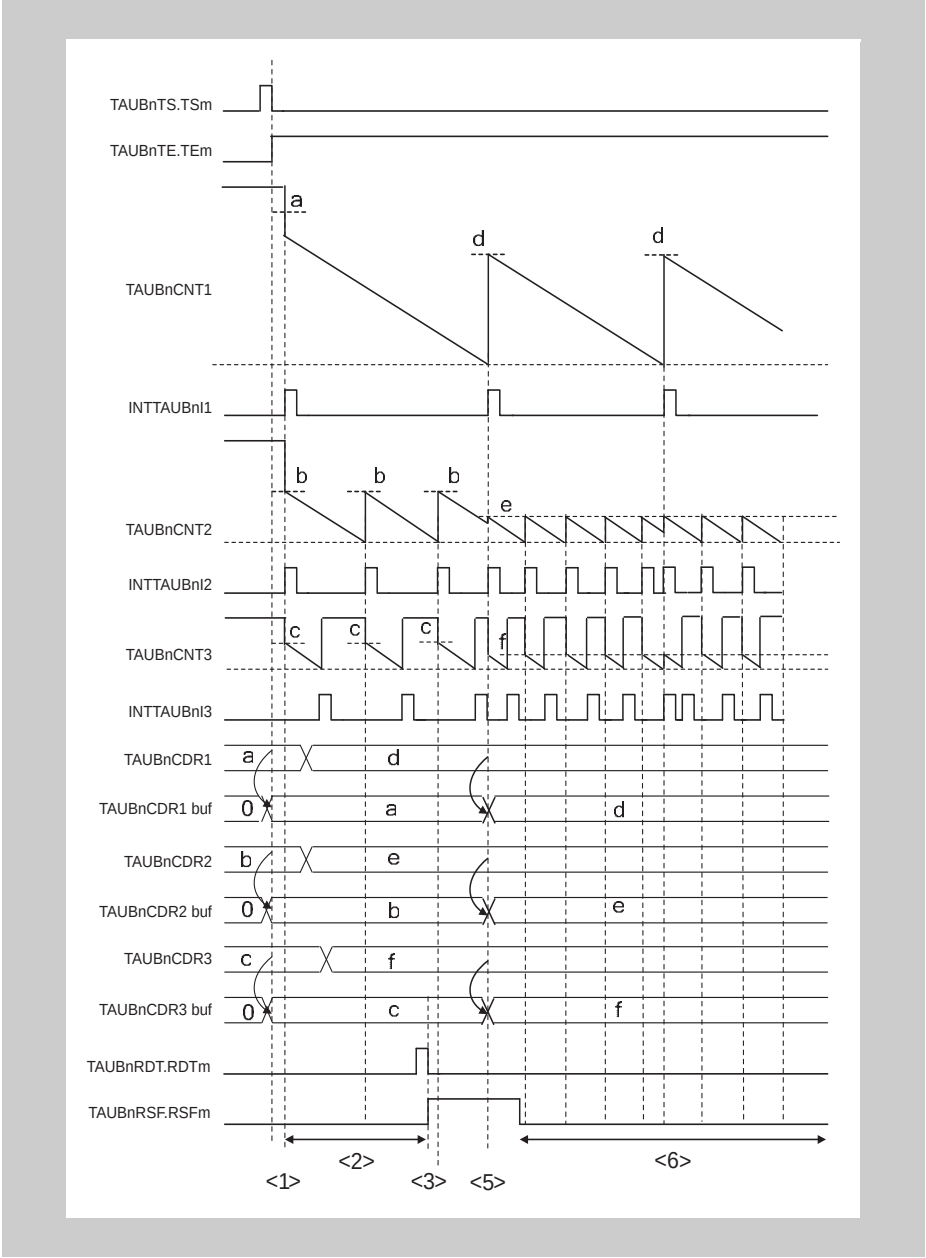



**Figure 14-8 Simultaneous rewrite when INTTAUBnIm is generated on an upper channel specified by TAUBnRDC.RDCm**

Setup:

- CH1 is an upper channel, counting down. CH2 is a master channel, CH3 is the slave channel, and simultaneous rewrite method C1 is applied. The TAUBnRDC register specifies which upper channel is monitored for an INTTAUBnI trigger.

Description:

1. When the counter starts, the value of TAUBnCDRm is copied to the TAUBnCDRm buffer. The buffer value is written to the counter.
2. The TAUBnCDRm register can be written at any time, but the value does not affect the counter as the counter reads the buffer value.
3. The reload data trigger bit (TAUBnRDT.RDTm) is set to 1 which sets the status flag (TAUBnRSF.RSFm = 1), enabling simultaneous rewrite.
4. Even though simultaneous rewrite is enabled, it does not take place because it is only triggered by an interrupt on channel 1.
5. Simultaneous rewrite is triggered by INT1 which is caused by counter 1 reaching $0000_H$. The TAUBnCDRm values are written to the corresponding TAUBnCDRm buffers, the counters start to count down from the values in the TAUBnCDRm buffers, and the TAUBnRSF.RSFm bit is reset to 0.
6. The counter counts down and awaits the next simultaneous rewrite trigger. The values of the TAUBnCDRm registers can be changed again.

## 14.9 Channel Output Modes

The output of the TAUBnTTOUTm pin can be controlled in two ways, the latter of which can be further split into individual modes:

- By software (Direct Channel Output Mode, TAUBnTOE.TOEm = 0)

  When controlled by software, the output register bit (TAUBnTO.TOm) can be written and the value of the bit is transferred to the output pin (TAUBnTTOUTm).

- By TAUB signals (TAUBnTOE.TOEm = 1)

  When operated by TAUB signals, the output level of TAUBnTTOUTm is set or reset or toggled by internal signals. The value of TAUBnTO.TOm is updated accordingly to reflect the value of TAUBnTTOUTm.

  – Independently (Independent Channel Output Mode, TAUBnTOM.TOMm = 0)

    When operated independently, the output of the TAUBnTTOUTm pin is only affected by settings of channel m. Therefore, independent channel operation must be selected (TAUBnTOM.TOMm = 0).

  – Synchronously (Synchronous Channel Output Mode, TAUBnTOM.TOMm = 1)

    When operated synchronously, the output of the TAUBnTTOUTm pin is affected by settings of channel m and those of other channels. Therefore, synchronous channel operation must be selected for all participating channels (TAUBnTOM.TOMm = 1).

The TAUBnTO.TOm bit can always be read to determine the current value of TAUBnTTOUTm, regardless of whether the pin is controlled by software, operated independently, or operated synchronously.

**Control bits**

The settings of the control bits required to select a specific channel output mode are listed in *Table 14-16 “Channel output modes” on page 490* .

The channel output modes are described in detail in

- *14.9.2 “Channel output modes controlled independently by TAUBn signals” on page 492*
- *14.9.3 “Channel output modes controlled synchronously by TAUBn signals” on page 493* .

**Output logic**

Positive logic or inverted logic of the output is specified by control bit TAUBnTOL.TOLm.

The value of the TAUBnTOL.TOLm bit must be set before the counter is started. It can only be changed during operation in PWM output function or triangle PWM output function. Otherwise, changes to TAUBnTOL.TOLm result in an invalid TAUBnTTOUTm signal.

Refer to *14.8 “Simultaneous Rewrite” on page 479* .

The various channel output modes and the channel output control bits are listed in the following table.

**Multiple outputs**

For a function on channel m that uses its output and the outputs of other channels (q):

- If channel m requires a certain operation mode for channel q, set the operation mode on channel q.
- If channel m does not require a certain operation mode for channel q, the counter on channel q must be disabled (TAUBnTE.TEq = 0), i.e. no operation mode can be used on channel q, even one that does not generate an output.

**Table 14-16 Channel output modes**

| Channel output mode | TAUBn TOE. TOEm | TAUBn TOM. TOMm | TAUBn TOC. TOCm | TAUBn TDE. TDEm |
|---|---|---|---|---|
| **By software** | | | | |
| Direct Channel Output Mode | 0 | x | | |
| **By timer signals, independently (Independent Channel Output Mode)** | | | | |
| Independent Channel Output Mode 1 | 1 | 0 | 0 | 0 |
| Independent Channel Output Mode 2 | | | 1 | |
| **By timer signals, synchronously (Synchronous Channel Output Mode)** | | | | |
| Synchronous Channel Output Mode 1 | 1 | 1 | 0 | 0 |
| Synchronous Channel Output Mode 2 | | | 1 | |
| with Dead Time Output | | | | 1 |

- All combinations not listed in this table are forbidden.
- Bits marked with an x can be set to any value.

**Note** The following bits cannot be changed during count operation (TAUBnTE.TE = 1):

- TAUBnTOM.TOMm,
- TAUBnTOC.TOCm,
- TAUBnTDE.TDEm

### 14.9.1 General procedure for specifying a channel output mode

The following steps describe the general procedure for specifying a TAUBnTTOUTm channel output mode. The prerequisite is that timer output operation is disabled (TAUBnTOE.TOEm = 0).

1. Set TAUBnTO.TOm to specify the initial level of the TAUBnTTOUTm output.
2. Set the channel output mode using *Table 14-16 “Channel output modes” on page 490* and the output logic using the TAUBnTOL.TOLm bit.
3. Start the counter (TAUBnTS.TSm = 1).

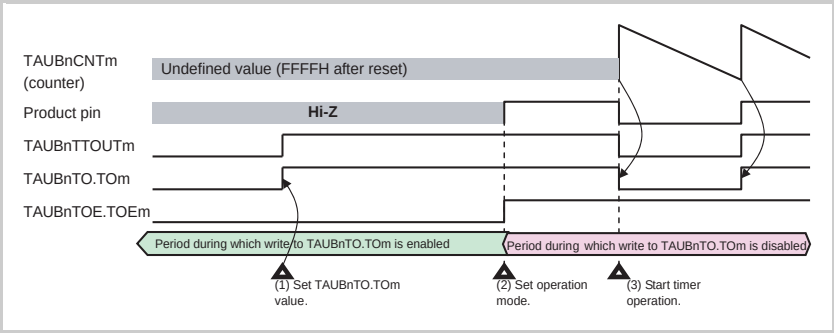



**Figure 14-9 General procedure for specifying a TAUBnTTOUTm channel output mode**

The following figure shows a general illustration of how the output changes when the counter is enabled:

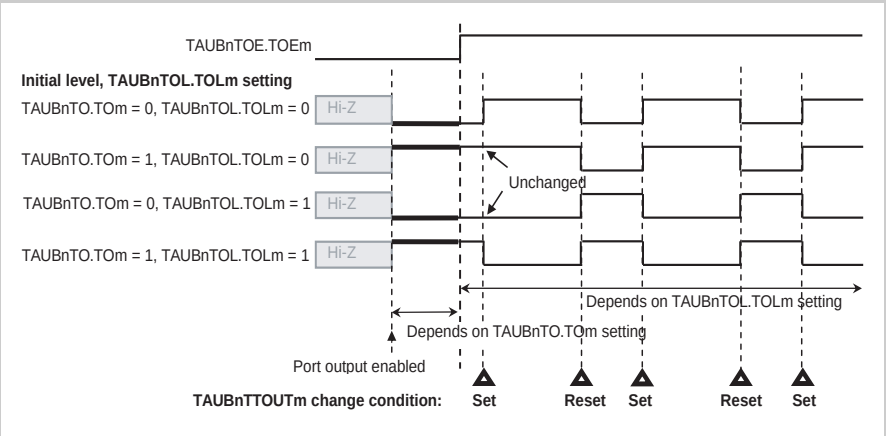



**Figure 14-10 General change of the TAUBnTTOUTm output**

- TAUBnTO.TOm sets the initial value of TAUBnTTOUTm and can be changed while TAUBnTOE.TOEm = 0.
- TAUBnTOL.TOLm specifies whether the set signal sets TAUBnTO.TOm to high (TAUBnTOL.TOLm = 0) or low (inverted logic, TAUBnTOL.TOLm = 1).

### 14.9.2 Channel output modes controlled independently by TAUBn signals

This chapter lists the channel output modes that are controlled independently by TAUBn signals. The control bits used to specify a mode are listed in *Table 14-16 “Channel output modes” on page 490* .

**(1) Independent Channel Output Mode 1**

**Set/reset conditions** In this output mode, TAUBnTTOUTm toggles when INTTAUBnIm is detected. The value of TAUBnTOL.TOLm is ignored.

**Prerequisites** None, other than those in *Table 14-16 “Channel output modes” on page 490* .

**(2) Independent Channel Output Mode 2**

**Set/reset conditions** In this output mode, TAUBnTTOUTm is set when INTTAUBnIm occurs upon count start and reset when INTTAUBnIm occurs due to a match between TAUBnCNTm and TAUBnCDRm.

**Prerequisites** None, other than those in *Table 14-16 “Channel output modes” on page 490* .

### 14.9.3 Channel output modes controlled synchronously by TAUBn signals

This chapter lists the channel output modes that are controlled synchronously by TAUBn signals. The control bits used to specify a mode are listed in *Table 14-16 “Channel output modes” on page 490* .

#### (1) Synchronous Channel Output Mode 1

**Set/reset conditions** In this output mode, INTTAUBnIm of the master channel serves as the set signal and INTTAUBnIm of the slave channel as the reset signal. If INTTAUBnIm of the master channel and INTTAUBnIm of the slave channel are generated at the same time, INTTAUBnIm of the slave channel (reset signal) has priority over INTTAUBnIm (set signal) of the master channel, i.e. the master channel is ignored.

**Prerequisites** None, other than those in *Table 14-16 “Channel output modes” on page 490* .

### (2) Synchronous Channel Output Mode 2

In this output mode, the operation mode must be set to Up Down Count mode. The result is a triangle PWM wave at TAUBnTTOUTm. For details refer to *14.21.1 “Triangle PWM Output Function” on page 636* .

**Set/reset conditions** TAUBnCNTm of the slave channel counts down and up alternatively. When it passes $0001_H$ it generates an interrupt, causing TAUBnTTOUTm to toggle.

**Prerequisites** A set of two channels is required to generate the triangle PWM output. TAUBnTTOUTm must be set to 0 before the function starts.

### (3) Synchronous Channel Output Mode 2 with Dead Time Output

In this output mode, a dead time delay is added to TAUBnTTOUTm. The set/reset conditions are shown in the following figure.

**Set/reset conditions**

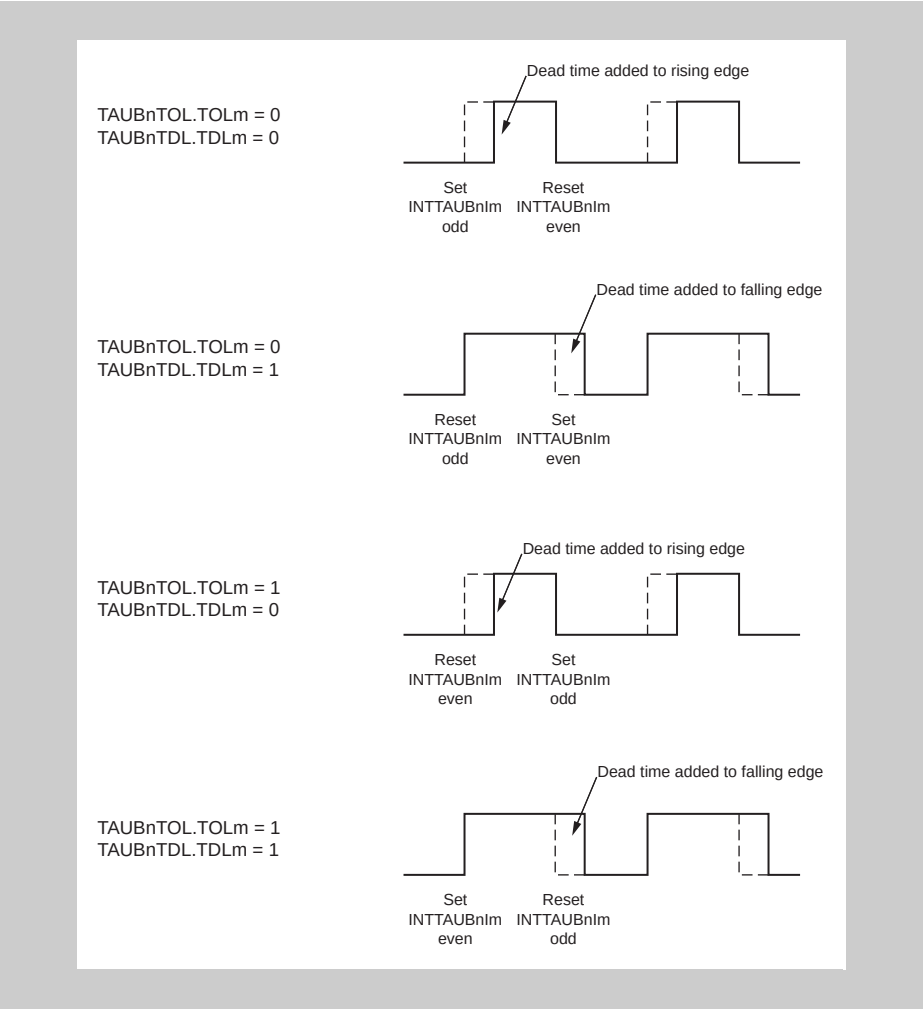



**Figure 14-11 Set/reset conditions for Synchronous Channel Output Mode 2 with Dead Time Output**

The edge to which the dead time is added is specified using the TAUBnTDL.TDLm bit; for rising edge set TAUBnTDL.TDLm = 0 and for falling edge set TAUBnTDL.TDLm = 1.

**Prerequisites** Dead time control requires a set of three channels, each operating in the following modes:

- One master channel

  The master channel must be set to Interval Timer Mode

- One even slave channel

  The even slave channel must be set to Up Down Count Mode

- One odd slave channel (even channel + 1)

  The odd slave channel must be set to One Count Mode

The values of the following bits must be the same for the odd channel and the even channel:

- TAUBnTOE.TOEm,
- TAUBnTOM.TOMm,
- TAUBnTOC.TOCm,
- TAUBnTDE.TDEm

## 14.10 Start Timing of Operating Modes

This chapter describes when the counters of the different operating modes start after the TAUBnTS.TSm bit is set to 1.

In all modes, the value of the data register and whether or not an interrupt is issued depends on the individual mode and corresponding register settings.

### 14.10.1 Interval Timer Mode, Judge Mode, Capture Mode, Up Down Count Mode

The counter starts at the start of the next count clock cycle after TAUBnTS.TSm is set to 1. The value of data register is also loaded at the point the counter starts.

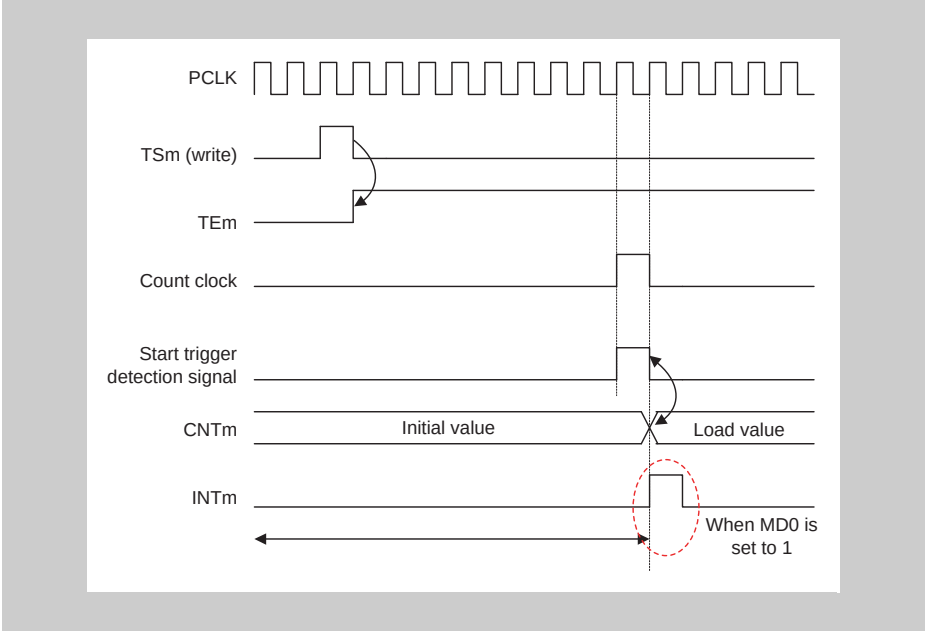



**Figure 14-12 Start timing of Interval Timer Mode, Judge Mode, Capture Mode, Up Down Count Mode**

**Note** In Up Down Count Mode, MD0 must be set to 0.

### 14.10.2 Event Mode

The value of the data register is loaded as soon as TAUBnTS.TSm is set to 1. The counter also starts immediately. The value of the data register changes with the subsequent count clock cycles.

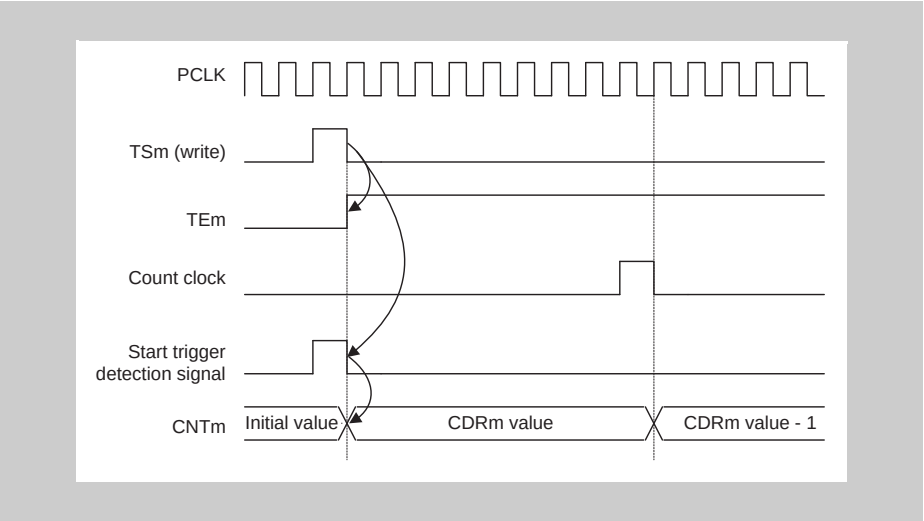



**Figure 14-13 Start timing of Event Mode**

### 14.10.3 All other operating modes

In all other operating modes, the count clock cycles are ignored with regard to starting the counter. The counter is only triggered by detection of a valid TAUBnTTINm edge. The value of data register is also loaded at the point the counter starts. Nevertheless, the count clock cycles determine the frequency with which all operations take place.

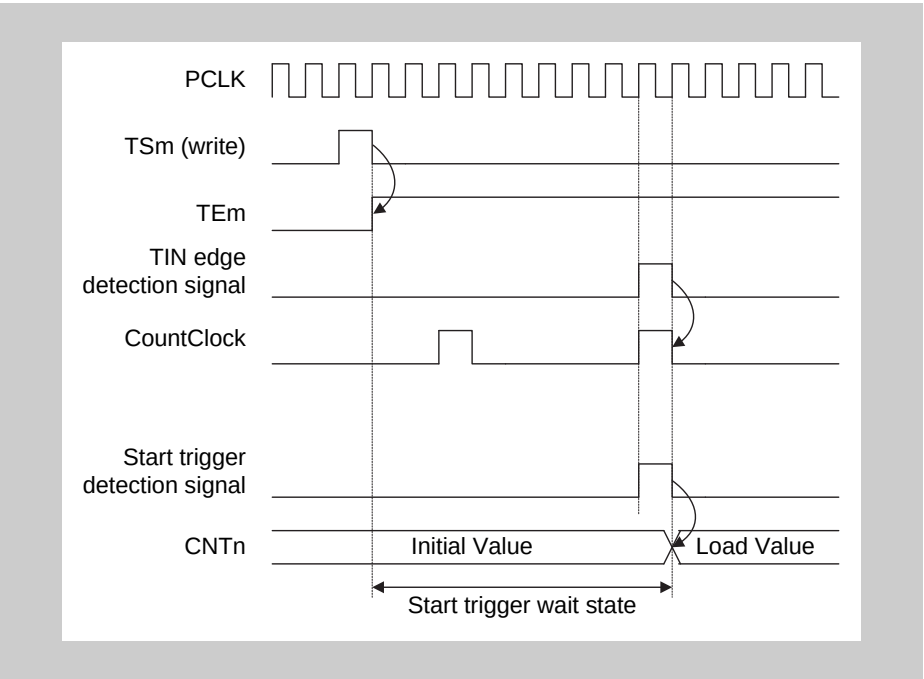



**Figure 14-14 Start timing of all other operating modes**

## 14.11 TAUBnTTOUTm toggle and INTTAUBnIm Generation when Counter start is triggered (MD0-bit)

It is possible to specify whether an INTTAUBnIm is generated when the counter starts, gets restarted or is triggered by an external signal, using the TAUBnCMOR.MD0 bit. The effect of the bit depends on the selected mode, as shown in the following table. The effects of INTTAUBnIm on TAUBnTTOUTm depend on the selected channel operation function.

**Table 14-17 Effect of CMOR.MD0 bit on generation of INTTAUBnIm when counter is triggered**

| Mode | TAUBnCMOR.MD0 bit | INTTAUBnIm generated when counter is (re)started or triggered by TINm input signal |
|---|---|---|
| Interval Timer Mode<br>Capture Mode<br>Count Capture Mode | 0 | No |
| | 1 | Yes |
| Capture & One Count Mode<br>Capture & Gate Count Mode<br>Event Count Mode<br>Up Down Count Mode | 0 | No |
| One Count Mode<br>Gate Count Mode | 0/1 | No, regardless of setting of TAUBnCMOR.MD0 bit. |
| Pulse One Count Mode | | Yes, regardless of setting of TAUBnCMOR.MD0 bit. |

**Note** As an example see figure *14-23 "Forced restart operation, TAUBnCMORm.MD0 = 1"* and figure *14-24 "Forced restart operation, TAUBnCMORm.MD0 = 0"*.
Refer to the description of *"Role of the MD0 bit"* also.

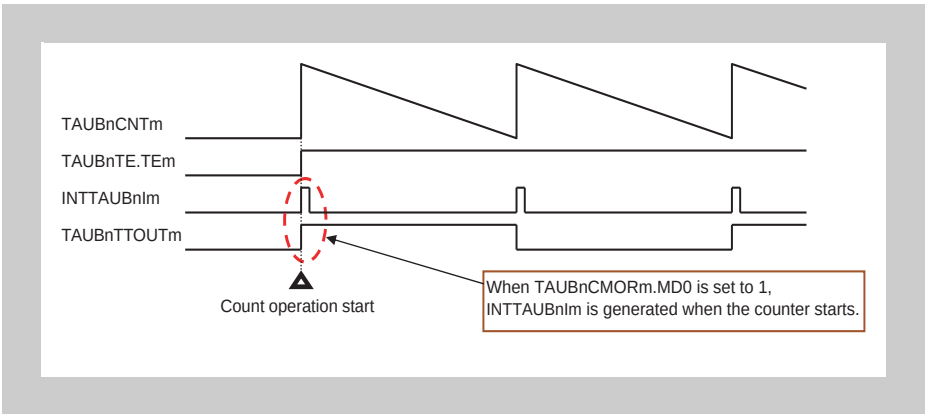



**Figure 14-15 INTTAUBnIm generated when counter starts**

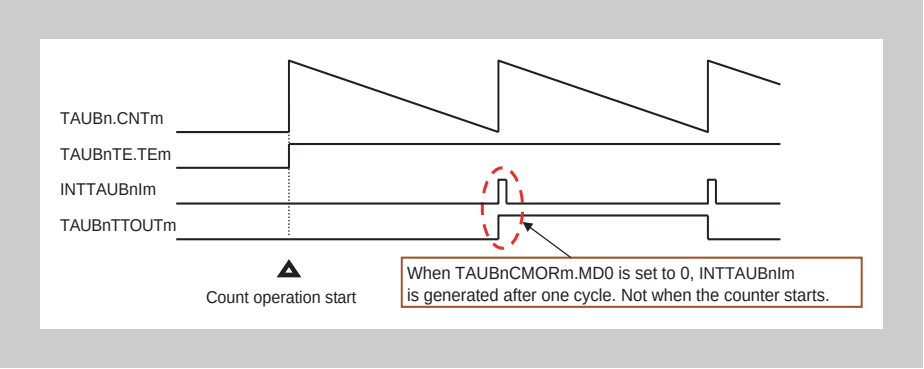



**Figure 14-16 INTTAUBnIm not generated when counter starts**

## 14.12 TAUBnTTINm Edge Detection

Edge detection is based on the operation clock. This means that an edge can only be detected at the next rising edge of the operation clock. This can lead to a maximum delay of one operation clock cycle.

The following figure shows when edge detection takes place.

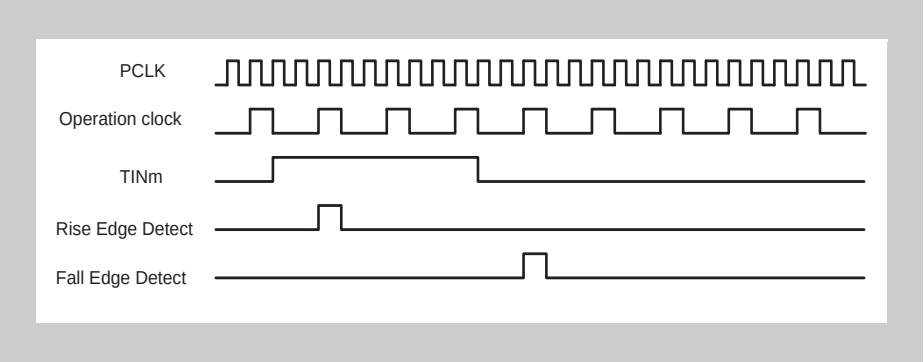



**Figure 14-17 Basic edge detection timing**

## 14.13 Independent Channel Operation Functions

The following sections list the independent channel operation functions provided by the Timer Array Unit B. For a general overview of independent channel operation, see *14.4 “Functional Description” on page 472* .

## 14.14 Independent Channel Interrupt Functions

This chapter describes functions that generate interrupts at regular intervals or with a specified delay.

- *14.14.1 “Interval Timer Function”*
- *14.14.2 “TAUBnTTINm Input Interval Timer Function”*
- *14.14.3 “One-Pulse Output Function”*

### 14.14.1 Interval Timer Function

#### (1) Overview

**Summary** This function is used as a reference timer for generating timer interrupts (INTTAUBnIm) at regular intervals. When an interrupt is generated, the TAUBnTTOUTm signal toggles, resulting in a square wave.

**Prerequisites**

- The operation mode must be set to Interval Timer Mode, refer to *Table 14-18 “TAUBnCMORm settings for Interval Timer Function” on page 504*
- The channel output mode must be set to Independent Channel Output Mode 1, refer to *14.9 “Channel Output Modes” on page 489*

**Description** The counter is started by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. The current value of TAUBnCDRm is written to TAUBnCNTm and the counter starts to count down from this value.

When the counter reaches $0000_H$, INTTAUBnIm is generated and the TAUBnTTOUTm signal toggles. TAUBnCNTm then reloads the TAUBnCDRm value and subsequently continues operation.

The value of TAUBnCDRm can be rewritten at any time, and the changed value of TAUBnCDRm is applied the next time the counter starts to count down.

The counter can be stopped by setting TAUBnTT.TTm to 1, which in turn sets TAUBnTE.TEm to 0. TAUBnCNTm and TAUBnTTOUTm stop but retain their values. The counter can be reset by setting TAUBnTS.TSm to 1. The counter can also be forcibly restarted (without stopping it first) by setting TAUBnTS.TSm to 1 during operation.

**Conditions** If the TAUBnCMORm.MD0 bit is set to 0, the first interrupt after a start or restart is not generated, and therefore TAUBnTTOUTm does not toggle. This results in an inverted TAUBnTTOUTm signal compared to when TAUBnCMORm.MD0 is set to 1. For details refer to *14.11 “TAUBnTTOUTm toggle and INTTAUBnIm Generation when Counter start is triggered (MD0-bit)” on page 498* .

#### (2) Equations

INTTAUBnIm cycle = count clock cycle x (TAUBnCDRm + 1)

TAUBnTTOUTm square wave cycle = count clock cycle x (TAUBnCDRm + 1) x 2

### (3) Block diagram and general timing diagram

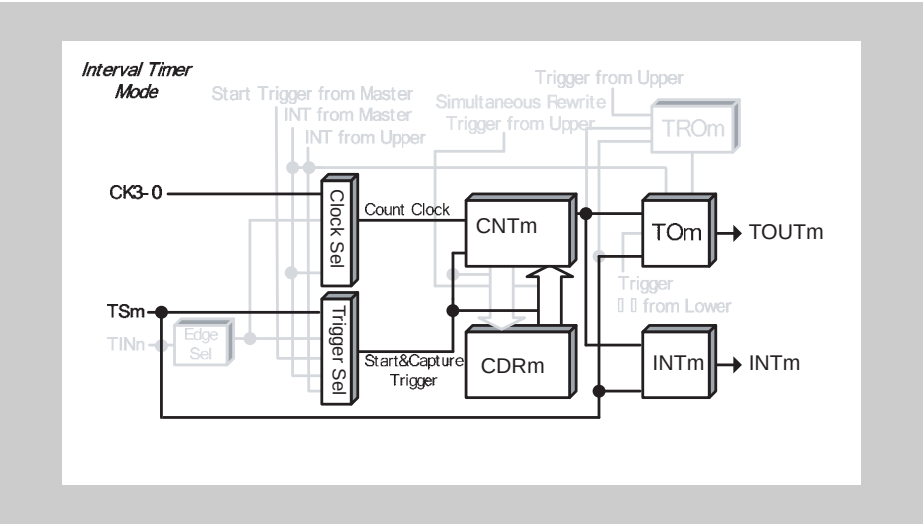



**Figure 14-18 Block diagram for Interval Timer Function**

The following settings apply to the general timing diagram:

- INTTAUBnIm generated at operation start (TAUBnCMORm.MD0 = 1)

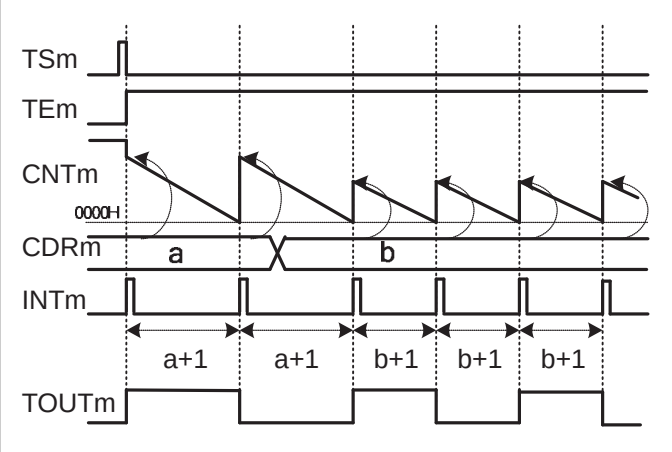



**Figure 14-19 General timing diagram for Interval Timer Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | - | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 14-18 TAUBnCMORm settings for Interval Timer Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUBnIm not generated and TAUBnTTOUTm does not toggle at operation start or restart<br>1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start or restart |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-19 TAUBnCMURm settings for Interval Timer Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used, so set to 00 |

### (c) Channel output mode

**Table 14-20 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOE.TOEm | 1: Disables Direct Channel Output Mode |
| TOM.TOMm | 0: Independent channel output |
| TOC.TOCm | 0: Operation mode 1 (= Toggle mode if TAUBnTOM.TOMm = 0) |
| TOL.TOLm | 0: Positive logic |
| TDE.TDEm | 0: Disables dead time operation |
| TDL.TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

**Note** The channel output mode can also be set to Direct Channel Output Mode by setting TAUBnTOE.TOEm = 0. TAUBnTTOUTm can then be controlled independently of the interrupts. For details refer to *14-16 “Channel output modes” on page 490* .

### (d) Simultaneous rewrite

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the Interval Timer Function. Therefore, these registers must be set to 0.

**Table 14-21 Simultaneous rewrite settings for Interval Timer Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDS.RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDM.RDMm | |
| RDC.RDCm | |

### (5) Operating procedure for Interval Timer Function

**Table 14-22 Operating procedure for Interval Timer Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-18 "TAUBnCMORm settings for Interval Timer Function" on page 504* and *Table 14-19 "TAUBnCMURm settings for Interval Timer Function" on page 504*<br><br>Set the value of the TAUBnCDRm register<br><br>Set the channel output mode by setting the control bits as described in *Table 14-20 "Control bit settings for Independent Channel Output Mode 1" on page 505* | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>TAUBnCNTm loads the TAUBnCDRm value.<br>When TAUBnCMORm.MD0 = 1, INTTAUBnIm is generated and TAUBnTTOUTm toggles. |
| During operation | The TAUBnCDRm register value can be changed at any time.<br>The TAUBnCNTm register can be read at all times. | TAUBnCNTm counts down. When the counter reaches $0000_H$:<br>• TAUBnCNTm reloads the TAUBnCDRm value and continues count operation<br>• INTTAUBnIm is generated and TAUBnTTOUTm toggles. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values. |

Restart

### (6) Specific timing diagrams

#### (a) TAUBnCDRm = $0000_H$, count clock = PCLK/2

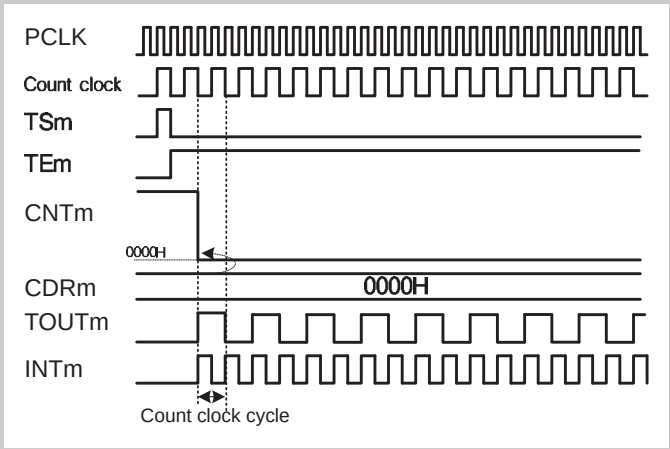



**Figure 14-20 TAUBnCDRm = $0000_H$, count clock = PCLK/2**

- If TAUBnCDRm = $0000_H$ and the count clock = PCLK/$2^1$, the TAUBnCDRm value is written to TAUBnCNTm every count clock, meaning that TAUBnCNTm is always $0000_H$.
- INTTAUBnIm is generated every count clock, resulting in TAUBnTTOUTm toggling every count clock.

#### (b) TAUBnCDRm = $0000_H$, count clock = PCLK

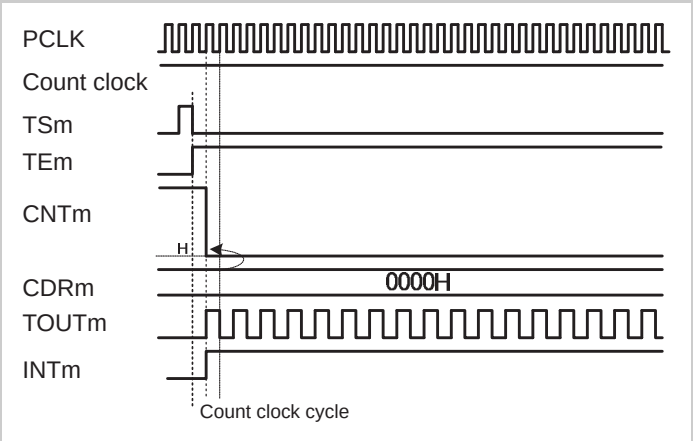



**Figure 14-21 TAUBnCDRm = $0000_H$, count clock = PCLK**

- If TAUBnCDRm = $0000_H$ and the count clock = PCLK, the TAUBnCDRm value is written to TAUBnCNTm every PCLK clock, meaning that TAUBnCNTm is always $0000_H$.
- INTTAUBnIm is generated continuously, resulting in TAUBnTTOUTm toggling every PCLK clock.

### (c) Operation stop and restart

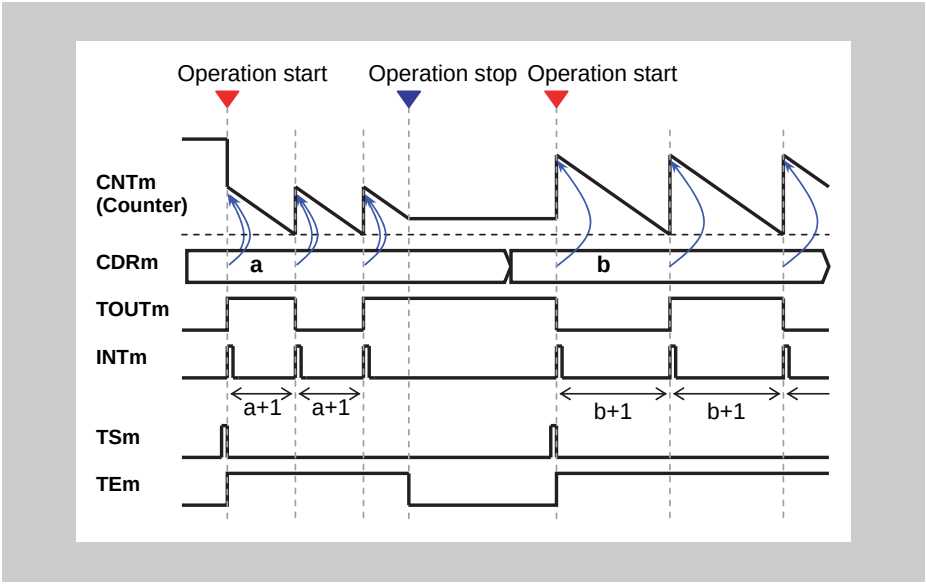



**Figure 14-22 Operation stop and restart, TAUBnCMORm.MD0 = 1**

- The counter can be stopped by setting TAUBnTT.TTm to 1, which in turn sets TAUBnTE.TEm to 0.
- TAUBnCNTm and TAUBnTTOUTm stop but retain their values.
- The counter can be restarted by setting TAUBnTS.TSm to 1.

### (d) Forced restart

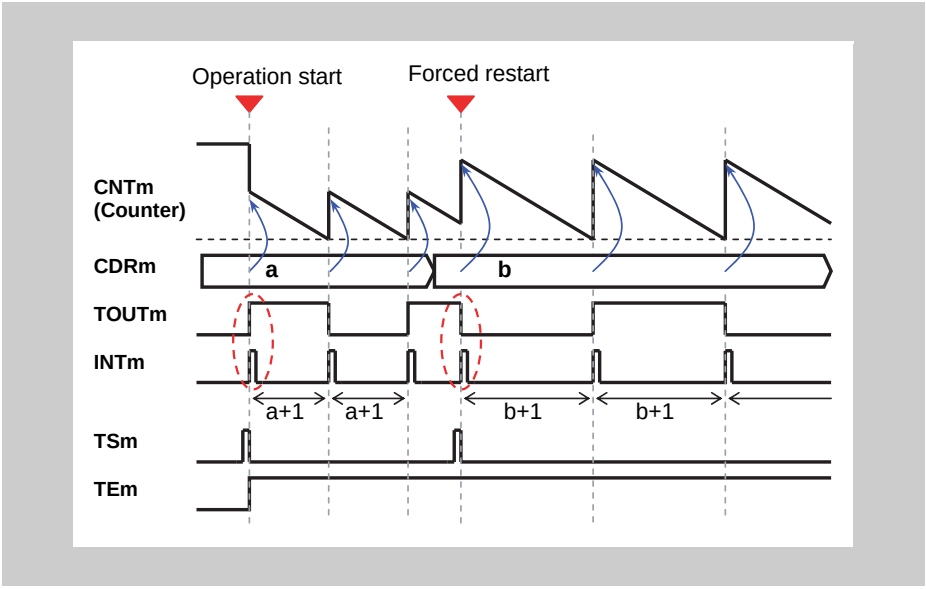



Figure 14-23 Forced restart operation, TAUBnCMORm.MD0 = 1

- The counter can be forcibly restarted (without stopping it first) by setting TAUBnTS.TSm to 1 during operation.
- If the TAUBnCMORm.MD0 bit is set to 1, an interrupt at start or restart is generated and the output TOUTm toggles.

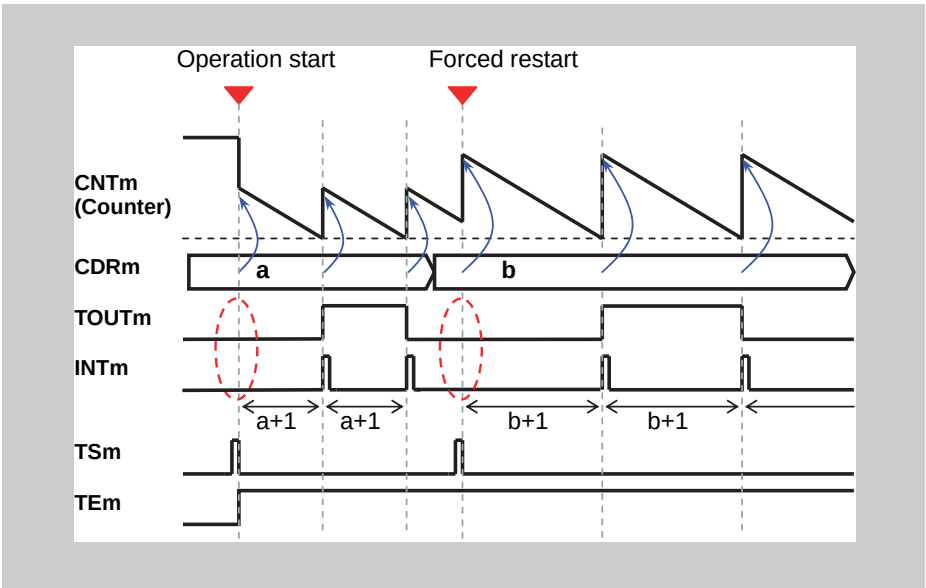



Figure 14-24 Forced restart operation, TAUBnCMORm.MD0 = 0

- The counter can be forcibly restarted (without stopping it first) by setting TAUBnTS.TSm to 1 during operation.
- If the TAUBnCMORm.MD0 bit is set to 0, the interrupt at start or restart is not generated and TOUTm does not toggle..

## 14.14.2 TAUBnTTINm Input Interval Timer Function

### (1) Overview

**Summary** This function is used as a reference timer for generating timer interrupts (INTTAUBnIm) at regular intervals or when a valid TAUBnTTINm input edge is detected. When an interrupt is generated, the TAUBnTTOUTm signal toggles, resulting in a square wave.

**Prerequisites**

- The operation mode must be set to Interval Timer Mode, refer to *Table 14-23 “TAUBnCMORm settings for TAUBnTTINm Input Interval Timer Function” on page 512*
- The channel output mode must be set to Independent Channel Output Mode 1, refer to *14.9 “Channel Output Modes” on page 489*

**Description** This function operates in an identical manner to the Interval Timer Function (see *14.14.1 “Interval Timer Function” on page 502* ), except that this function is restarted by a valid TAUBnTTINm input edge. The type of edge used as the trigger is specified using the TAUBnCMURm.TIS[1:0] bits. Either rising edge, falling edge, or rising and falling edge can be selected.

### (2) Equations

INTTAUBnIm cycle = count clock cycle × (TAUBnCDRm + 1)

TAUBnTTOUTm square wave cycle = count clock cycle × (TAUBnCDRm + 1) × 2

### (3) Block diagram and general timing diagram

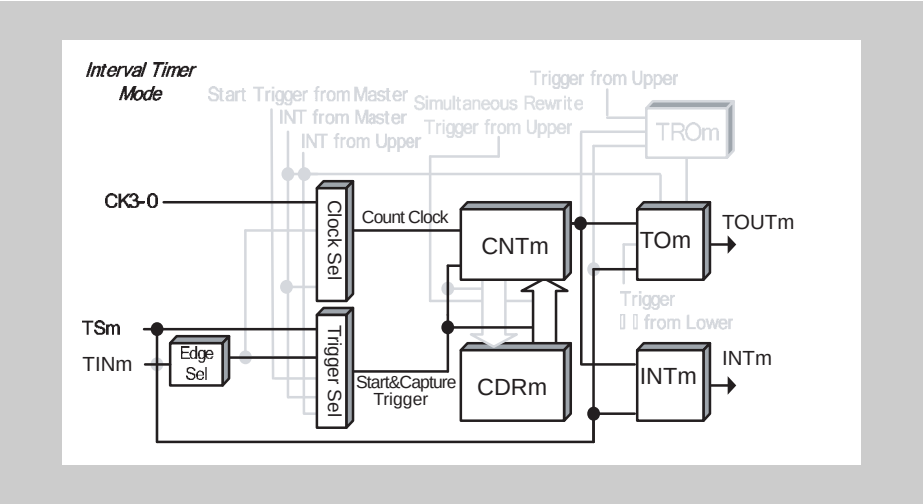



**Figure 14-25 Block diagram for TAUBnTTINm Input Interval Timer Function**

The following settings apply to the general timing diagram:

- INTTAUBnIm generated at operation start (TAUBnCMORm.MD0 = 1) and valid edge detection of input signal.
- Rising edge detection (TAUBnCMURm.TIS[1:0] = $01_B$)

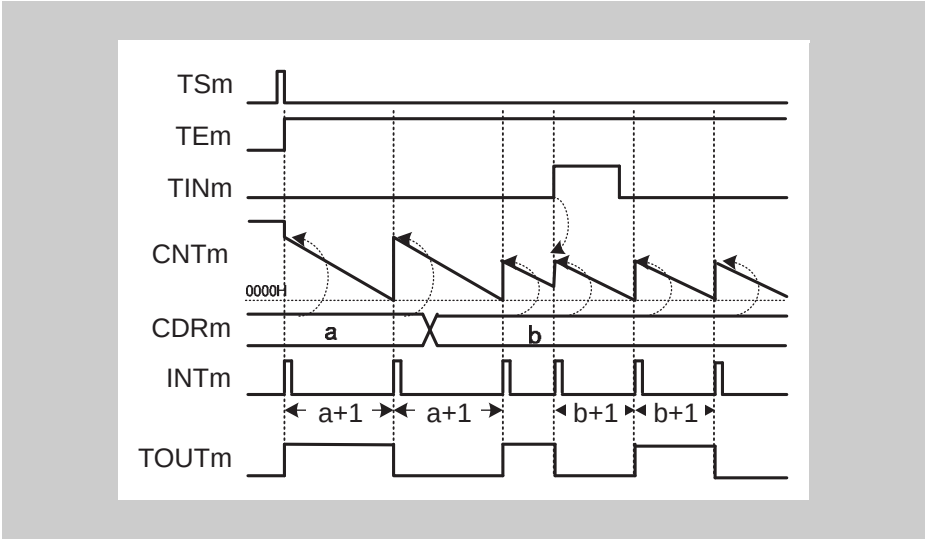



**Figure 14-26 General timing diagram for TAUBnTTINm Input Interval Timer Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

Table 14-23 TAUBnCMORm settings for TAUBnTTINm Input Interval Timer Function

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid TAUBnTTINm input edge signal is used as the external start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUBnIm not generated and TAUBnTTOUTm does not toggle at operation start<br>1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

Table 14-24 TAUBnCMURm settings for TAUBnTTINm Input Interval Timer Function

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection |

### (c) Channel output mode

**Table 14-25 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOE.TOEm | 1: Disables Direct Channel Output Mode |
| TOM.TOMm | 0: Independent channel output |
| TOC.TOCm | 0: Operation mode 1 (= Toggle mode if TAUBnTOM.TOMm = 0) |
| TOL.TOLm | 0: Positive logic |
| TDE.TDEm | 0: Disables dead time operation |
| TDL.TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

**Note** The channel output mode can also be set to Direct Channel Output Mode by setting TAUBnTOE.TOEm = 0. TAUBnTTOUTm can then be controlled independently of the interrupts. For details refer to *14-16 “Channel output modes” on page 490* .

### (d) Simultaneous rewrite

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the TAUBnTTINm Input Interval Timer Function. Therefore, these registers must be set to 0.

**Table 14-26 Simultaneous rewrite settings for TAUBnTTINm Input Interval Timer Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDS.RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDM.RDMm | |
| RDC.RDCm | |

### (5) Operating procedure for TAUBnTTINm Input Interval Timer Function

**Table 14-27 Operating procedure for TAUBnTTINm Input Interval Timer Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-23 "TAUBnCMORm settings for TAUBnTTINm Input Interval Timer Function" on page 512* and *Table 14-24 "TAUBnCMURm settings for TAUBnTTINm Input Interval Timer Function" on page 512*<br><br>Set the value of the TAUBnCDRm register<br><br>Set the channel output mode by setting the control bits as described in *Table 14-25 "Control bit settings for Independent Channel Output Mode 1" on page 513* | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>TAUBnCNTm loads the TAUBnCDRm value.<br>When TAUBnCMORm.MD0 = 1, INTTAUBnIm is generated and TAUBnTTOUTm toggles. |
| During operation | The values of the TAUBnCMURm.TIS[1:0] and TAUBnTO.TOm bits and the TAUBnCDRm register can be changed at any time.<br>The TAUBnCNTm register can be read at all times.<br><br>Detection of TAUBnTTINm edge | TAUBnCNTm counts down. When the counter reaches $0000_H$:<br>• TAUBnCNTm reloads the TAUBnCDRm value and continues count operation<br>• INTTAUBnIm is generated and TAUBnTTOUTm toggles<br>When a TAUBnTTINm input valid edge is detected during count operation, TAUBnCNTm reloads the TAUBnCDRm value and continues count operation.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values. |

Restart

**(6) Specific timing diagrams**

The timing diagrams in *14.14.1 "Interval Timer Function" on page 502* also apply, except for this function the counter can also be restarted by a valid TAUBnTTINm input edge.

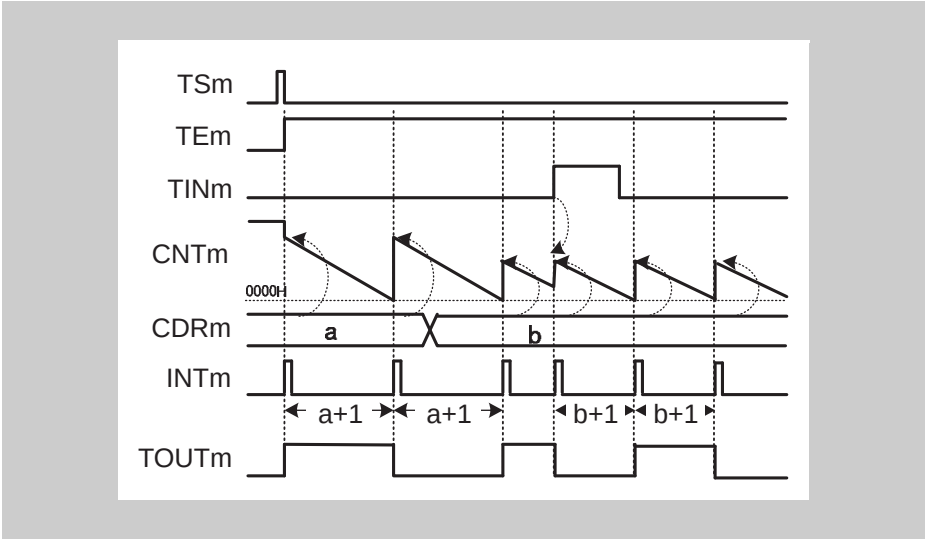



**Figure 14-27 Counter triggered by rising TAUBnTTINm input edge (TAUBnCMURm.TIS[1:0] = $01_B$), TAUBnCMORm.MD0 = 1**

- If a valid TAUBnTTINm input edge is detected, an interrupt is generated which causes TAUBnTTOUTm to toggle. In this example, the valid edge is a rising edge (TAUBnCMURm.TIS[1:0] = $01_B$).

### 14.14.3 One-Pulse Output Function

#### (1) Overview

**Summary** This function generates an interrupt (INTTAUBnIm) when a valid TAUBnTTINm input edge is detected and also a specific interval later. TAUBnTTINm input signal pulses that occur within the defined interval are ignored. When an interrupt is generated, the TAUBnTTOUTm signal toggles, resulting in a square wave.

**Prerequisites**
- The operation mode must be set to Pulse One Count Mode, refer to *Table 14-28 “TAUBnCMORm settings for One-Pulse Output Function” on page 518*
- The channel output mode must be set to Independent Channel Output Mode 1, refer to *14.9 “Channel Output Modes” on page 489*
- Trigger detection must be disabled during counting (TAUBnCMORn.MD0 = 0).

**Description** The counter is enabled by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation.

The counter starts when a valid TAUBnTTINm input edge is detected. The value of TAUBnCDRm is written to TAUBnCNTm and the counter starts to count down from the TAUBnCDRm value. An interrupt is generated and TAUBnTTOUTm toggles.

When the counter reaches $0001_H$ an interrupt is generated and TAUBnTTOUTm toggles. The counter stops at $0000_H$ and awaits the next valid TAUBnTTINm input edge.

When the counter is counting down, further TAUBnTTINm input signals are ignored, i.e. the counter does not reset.

The value of TAUBnCDRm can be rewritten at any time, and the changed value of TAUBnCDRm is applied the next time the counter starts to count down.

**Conditions** The type of edge used as the trigger is specified by the TAUBnCMURm.TIS[1:0] bits:
- If TAUBnCMURm.TIS[1:0] = $00_B$, falling edges trigger the counter.
- If TAUBnCMURm.TIS[1:0] = $01_B$, rising edges trigger the counter.
- If TAUBnCMURm.TIS[1:0] = $10_B$, rising and falling edges trigger the counter.

#### (2) Equations

Interval between TAUBnTTINm and INTTAUBnIm = TAUBnTTOUTm (timer output) width = count clock cycle × TAUBnCDRm

### (3) Block diagram and general timing diagram

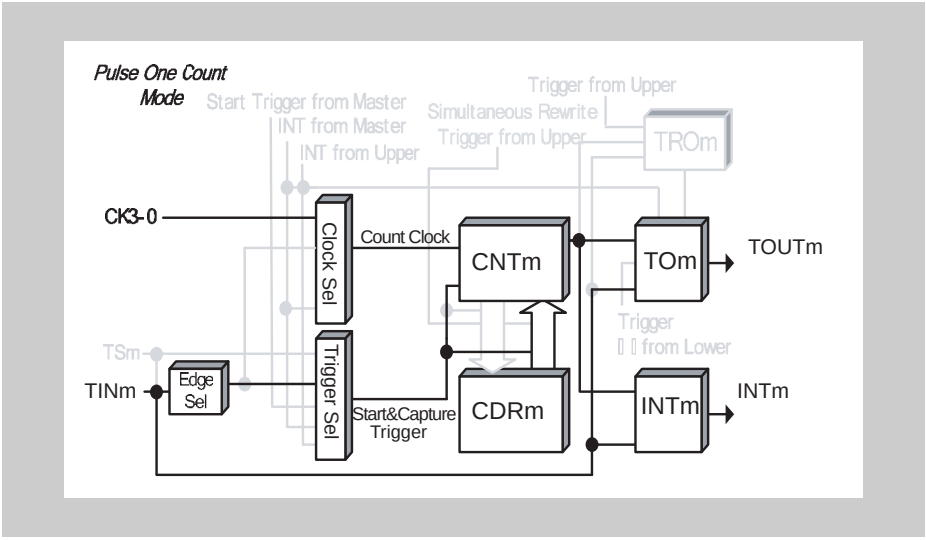



**Figure 14-28 Block diagram for One-Pulse Output Function**

The following settings apply to the general timing diagram:

- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

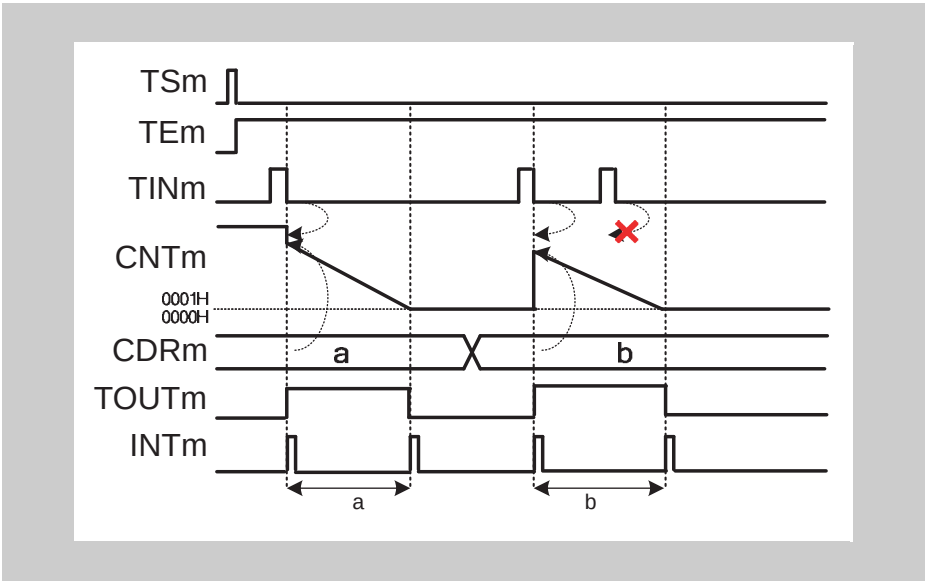



**Figure 14-29 General timing diagram for One-Pulse Output Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-28 TAUBnCMORm settings for One-Pulse Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid TAUBnTTINm input edge signal is used as the external start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 1010: Pulse One Count Mode |
| MD0 | 0: INTTAUBnIm not generated and TAUBnTTOUTm does not toggle at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-29 TAUBnCMURm settings for One-Pulse Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection (low width measurement) |

### (c) Channel output mode

**Table 14-30 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 0: Independent channel output |
| TOCm | 1: Independent Channel Output Mode 2 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

**Note** The channel output mode can also be set to Direct Channel Output Mode by setting TAUBnTOE.TOEm = 0. TAUBnTTOUTm can then be controlled independently of the interrupts. For details refer to *Table 14-16 “Channel output modes” on page 490* .

### (d) Simultaneous rewrite

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the One-Pulse Output Function. Therefore, these registers must be set to 0.

**Table 14-31 Simultaneous rewrite settings for One-Pulse Output Function**

| Bit name | Setting |
|---|---|
| RDEm | 0: Disables simultaneous rewrite |
| RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDMm | |
| RDCm | |

### (5) Operating procedure for One-Pulse Output Function

**Table 14-32 Operating procedure for One-Pulse Output Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-28 "TAUBnCMORm settings for One-Pulse Output Function" on page 518* and *Table 14-29 "TAUBnCMURm settings for One-Pulse Output Function" on page 518*<br><br>Set the value of the TAUBnCDRm register<br><br>Set the channel output mode by setting the control bits as described in *Table 14-30 "Control bit settings for Independent Channel Output Mode 1" on page 519* | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUBnTTINm start edge | TAUBnTE.TEm is set to 1 and TAUBnCNTm waits for detection of the TAUBnTTINm start edge.<br><br>When a start edge is detected, TAUBnCNTm loads the TAUBnCDRm value. |
| During operation | The value of TAUBnCDRm can be changed at any time.<br>The TAUBnCNTm register can be read at all times. | INTTAUBnIm is generated when TAUBnCNTm starts and TAUBnTTOUTm is set to its active level.<br>TAUBnCNTm counts down. When the counter reaches $0001_H$:<br>• INTTAUBnIm is generated<br>• TAUBnTTOUTm is set to its inactive level.<br>TAUBnCNTm stops counting and waits for a trigger.<br><br>If a trigger occurs while TAUBnCNTm is counting, the trigger is ignored.<br><br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values. |

Restart

## 14.15 Independent Channel Signal Measurement Functions

This chapter describes functions that measure the widths of an individual TAUBnTTINm pulse or the total width of successive TAUBnTTINm pulses. It also describes functions that measure the interval of the signal or that compare the width of a pulse with a reference value.

- *14.15.1 “TAUBnTTINm Input Pulse Interval Measurement Function”*
- *14.15.2 “TAUBnTTINm Input Signal Width Measurement Function”*
- *14.15.3 “Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)”*
- *14.15.4 “TAUBnTTINm Input Period Count Detection Function”*
- *14.15.5 “Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)”*
- *14.15.6 “TAUBnTTINm Input Pulse Interval Judgment Function”*
- *14.15.7 “TAUBnTTINm Input Signal Width Judgment Function”*

### 14.15.1 TAUBnTTINm Input Pulse Interval Measurement Function

#### (1) Overview

**Summary** This function captures the count value and uses this value and the overflow bit TAUBnCSRm.OVF to measure the interval of the TAUBnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Capture Mode, refer to *Table 14-34 "TAUBnCMORm settings for TAUBnTTINm Input Pulse Interval Measurement Function" on page 524*
- TAUBnTTOUTm is not used for this function

**Description** The counter is started by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. The counter TAUBnCNTm starts counting up from $0000_H$. When a valid TAUBnTTINm edge is detected, the value of TAUBnCNTm is captured, transferred to TAUBnCDRm, and an interrupt INTTAUBnIm is generated. The counter resets to $0000_H$ and subsequently continues operation.

If the counter reaches $FFFF_H$ before a valid TAUBnTTINm edge is detected, it overflows to $0000_H$. The counter is reset to $0000_H$ and subsequently continues operation. The values transferred to TAUBnCDRm and TAUBnCSRm.OVF respectively depend on the values of bits TAUBnCMORm.COS[1:0]:

**Table 14-33 Effects of an overflow**

| TAUBnCMORm.COS[1:0] | When overflow occurs | | When a valid TAUBnTTINm input is then detected | |
|---|---|---|---|---|
| | TAUBnCDRm | TAUBnCSRm.OVF | TAUBnCDRm and TAUBnCNTm | TAUBnCSRm.OVF |
| 00 | Unchanged | 0 | TAUBnCNTm written to TAUBnCDRm | 1 |
| 01 | | 1 | | |
| 10 | Set to $FFFF_H$ | 0 | TAUBnCNTm set to 0, TAUBnCDRm unchanged | 0 |
| 11 | | 1 | | |

If an overflow is set (TAUBnCSRm.OVF = 1), it can only be cleared by a CPU command that sets TAUBnCSCm.CLOV = 1.

The combination of the value of TAUBnCDRm and TAUBnCSRm.OVF can be used to deduce the interval of the TAUBnTTINm signal. However, if an overflow occurs multiple times before a valid TAUBnTTINm input is detected, the overflow bit TAUBnCSRm.OVF cannot indicate this.

The function can be stopped by setting TAUBnTT.TTm = 1, which in turn sets TAUBnTE.TEm = 0. TAUBnCNTm stops but retains its value. While the function is stopped, TAUBnTTINm input valid edge detection and TAUBnCNTm capture are not performed.

The function can be restarted by setting TAUBnTS.TSm = 1. The counter is reset to $0000_H$ and subsequently continues operation. The counter can also be forcibly restarted (without stopping it first) by setting TAUBnTS.TSm = 1 during operation.

**Conditions** If the TAUBnCMORm.MD0 bit is set to 0, the interrupt at start or restart is not generated. For details refer to *14.11 "TAUBnTTOUTm toggle and INTTAUBnIm Generation when Counter start is triggered (MD0-bit)" on page 498* .

**Note** When TAUBnCMORm.COS[1:0] = $11_B$, the value of TAUBnCNTm is *not* written to TAUBnCDRm when the first valid TAUBnTTINm input edge occurs after an overflow. However, an interrupt is generated.

**(2) Equations**

TAUBnTTINm input pulse interval = count clock cycle x [(TAUBnCSRm.OVF x ($FFFF_H$ + 1)) + TAUBnCDRm capture value + 1]

**(3) Block diagram and general timing diagram**

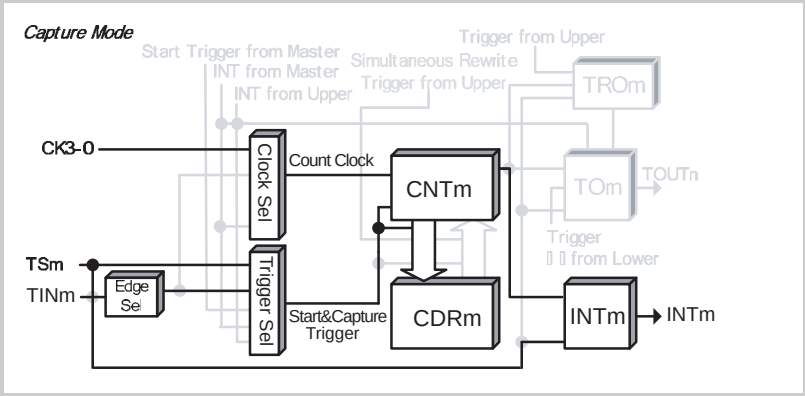



**Figure 14-30 Block diagram for TAUBnTTINm Input Pulse Interval Measurement Function**

The following settings apply to the general timing diagram:

- INTTAUBnIm not generated at operation start (TAUBnCMORm.MD0 = 0)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)
- When a valid TAUBnTTINm input is detected after an overflow TAUBnCDRm is changed and TAUBnCSRm.OVF is set to 1 (TAUBnCMORm.COS[1:0] = $00_B$)

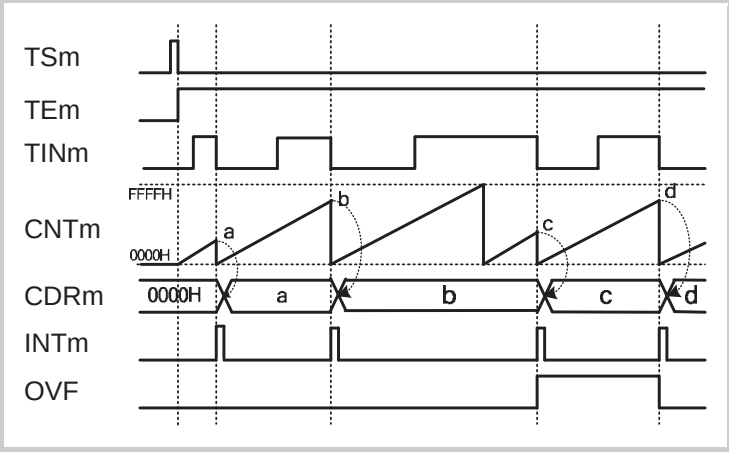



**Figure 14-31 General timing diagram for TAUBnTTINm Input Pulse Interval Measurement Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | - | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 14-34 TAUBnCMORm settings for TAUBnTTINm Input Pulse Interval Measurement Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid edge of the TAUBnTTINm input signal is the external capture trigger |
| COS[1:0] | See *Table 14-33 “Effects of an overflow” on page 522* |
| MD[4:1] | 0010: Capture Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start<br>1: Generates INTTAUBnIm at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-35 TAUBnCMURm settings for TAUBnTTINm Input Pulse Interval Measurement Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection (low width measurement)<br>11: Rising and falling edge detection (high width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simuBtaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the TAUBnTTINm Input Pulse Interval Measurement Function. Therefore, these registers must be set to 0.

**Table 14-36 Simultaneous rewrite settings for TAUBnTTINm Input Pulse Interval Measurement Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDS.RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDM.RDMm | |
| RDC.RDCm | |

### (5) Operating procedure for TAUBnTTINm Input Pulse Interval Measurement Function

**Table 14-37 Operating procedure for TAUBnTTINm Input Pulse Interval Measurement Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-34 "TAUBnCMORm settings for TAUBnTTINm Input Pulse Interval Measurement Function" on page 524* and *Table 14-35 "TAUBnCMURm settings for TAUBnTTINm Input Pulse Interval Measurement Function" on page 524*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>TAUBnCNTm is cleared to $0000_H$. INTTAUBnIm is generated when TAUBnCMORm.MD0 is set to 1. |
| During operation | Detection of TAUBnTTINm edges.<br><br>The TAUBnCMURm.TIS[1:0] bits can be changed at any time.<br>The TAUBnCDRm and TAUBnCSRm registers can be read at any time. | TAUBnCNTm starts to count up from $0000_H$. When a TAUBnTTINm valid edge is detected:<br>• TAUBnCNTm transfers (captures) its value to TAUBnCDRm, and returns to $0000_H$<br>• INTTAUBnIm is then generated.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and both it and TAUBnCSRm.OVF retain their current values. |

Restart

**(6) Specific timing diagrams: overflow behavior**

**(a) TAUBnCMORm.COS[1:0] = $00_B$**

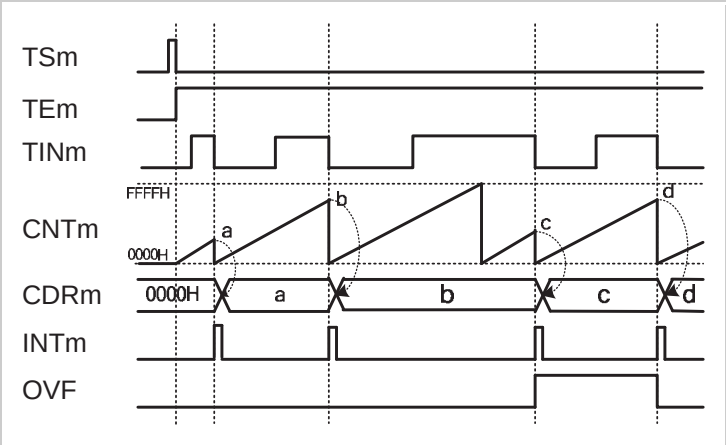



**Figure 14-32 TAUBnCMORm.COS[1:0] = $00_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, the value of TAUBnCDRm remains unchanged and TAUBnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUBnTTINm input edge, the value of TAUBnCNTm is written to TAUBnCDRm and TAUBnCSRm.OVF is set to 1.

**(b) TAUBnCMORm.COS[1:0] = $01_B$**

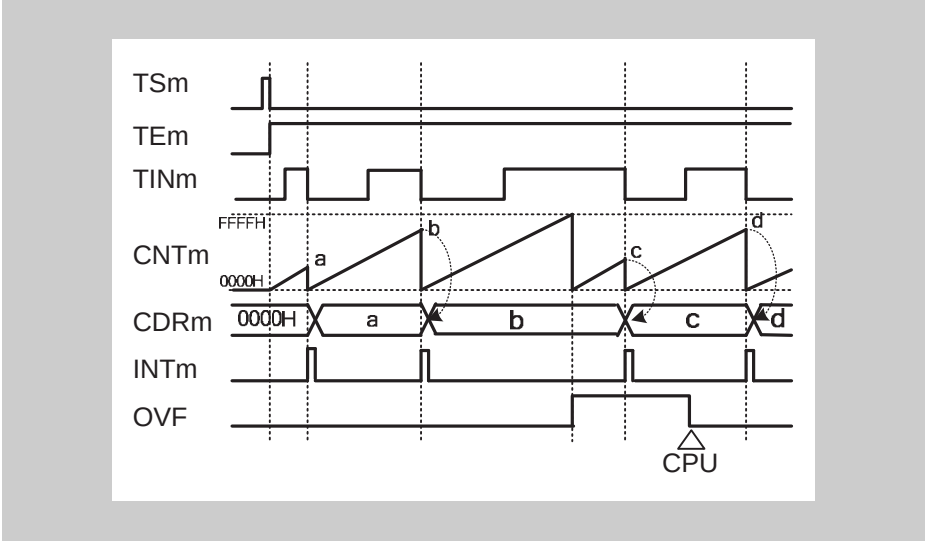



**Figure 14-33 TAUBnCMORm.COS[1:0] = $01_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, the value of TAUBnCDRm remains unchanged and TAUBnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUBnTTINm input edge, the value of TAUBnCNTm is written to TAUBnCDRm.
- TAUBnCSRm.OVF is only cleared by a CPU command.

**(c) TAUBnCMORm.COS[1:0] = $10_B$**

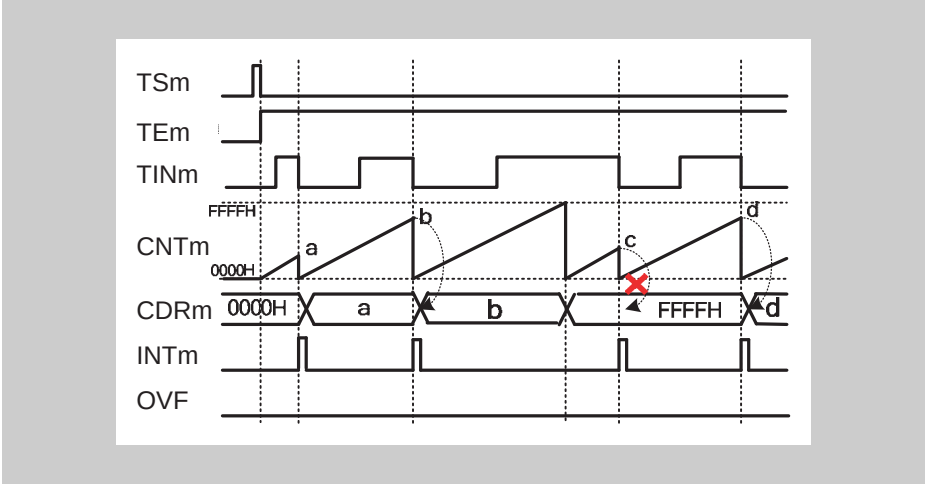



**Figure 14-34 TAUBnCMORm.COS[1:0] = $10_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, TAUBnCDRm is set to $FFFF_H$ and TAUBnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUBnTTINm input edge, TAUBnCNTm is reset to 0, but TAUBnCDRm and TAUBnCSRm.OVF remain unchanged.
- Thus, the next TAUBnTTINm input valid edge after the overflow is ignored.

**(d) TAUBnCMORm.COS[1:0] = $11_B$**

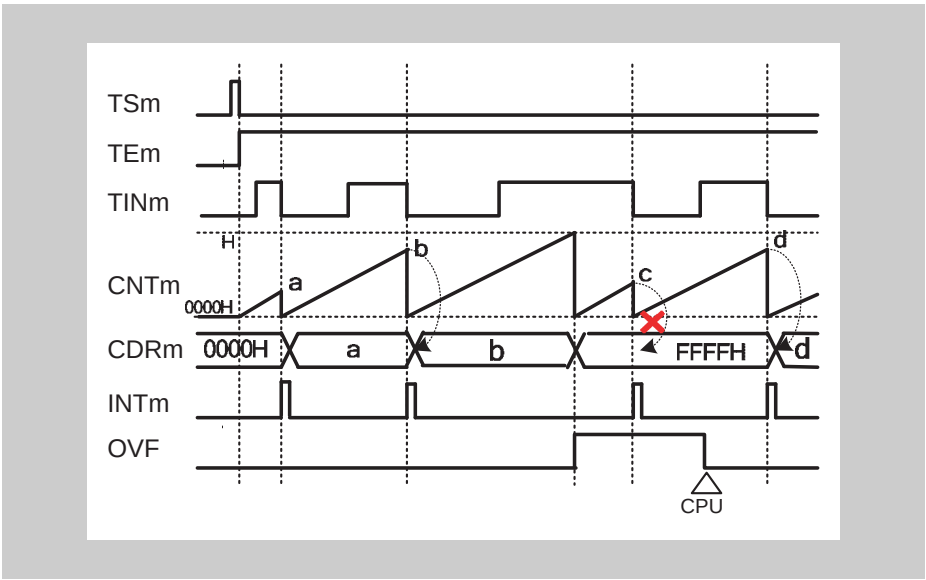



**Figure 14-35 TAUBnCMORm.COS[1:0] = $11_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, TAUBnCDRm is set to $FFFF_H$, and TAUBnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUBnTTINm input edge, TAUBnCNTm is reset to 0, but TAUBnCDRm and TAUBnCSRm.OVF remain unchanged.
- Thus, the next TAUBnTTINm input valid edge after the overflow is ignored.
- TAUBnCSRm.OVF is cleared by a CPU command.

## 14.15.2 TAUBnTTINm Input Signal Width Measurement Function

### (1) Overview

**Summary** This function measures the width of a TAUBnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Capture & One Count Mode, refer to *Table 14-39 "TAUBnCMORm settings for TAUBnTTINm Input Signal Width Measurement Function" on page 532*
- TAUBnTTOUTm is not used for this function
- TAUBnCMORm.MD0 must be set to 0

**Description** The counter is started by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. When a valid TAUBnTTINm start edge is detected, the counter TAUBnCNTm starts counting up from $0000_H$. When a valid TAUBnTTINm stop edge is detected, the value of TAUBnCNTm is captured, transferred to TAUBnCDRm, and an interrupt INTTAUBnIm is generated. The counter retains its value and awaits the next valid TAUBnTTINm input start edge.

If the counter reaches $FFFF_H$ before a valid TAUBnTTINm stop edge is detected, it overflows. The counter is reset to $0000_H$ and subsequently continues operation. The values transferred to TAUBnCDRm and TAUBnCSRm.OVF respectively depend on the values of bits TAUBnCMORm.COS[1:0]:

**Table 14-38 Effects of an overflow**

| TAUBnCMORm.COS[1:0] | When overflow occurs | | When a valid TAUBnTTINm input stop edge is detected | |
|---|---|---|---|---|
| | TAUBnCDRm | TAUBnCSRm.OVF | TAUBnCDRm and TAUBnCNTm | TAUBnCSRm.OVF |
| 00 | Unchanged | 0 | TAUBnCNTm written to TAUBnCDRm | 1 |
| 01 | | 1 | | |
| 10 | Set to $FFFF_H$ | 0 | TAUBnCNTm stops counting TAUBnCDRm unchanged | 0 |
| 11 | | 1 | | |

If an overflow is set (TAUBnCSRm.OVF = 1), it can only be cleared by a CPU command that sets TAUBnCSCm.CLOV = 1.

The combination of the value of TAUBnCDRm and TAUBnCSRm.OVF can be used to deduce the width of the TAUBnTTINm signal. However, if an overflow occurs multiple times before a valid TAUBnTTINm input is detected, the overflow bit TAUBnCSRm.OVF cannot indicate this.

This function cannot be forcibly restarted.

**Note** When TAUBnCMORm.COS[1:0] = $11_B$, the value of TAUBnCNTm is *not* written to TAUBnCDRm when the first valid TAUBnTTINm input edge occurs after an overflow. However, an interrupt is generated.

### (2) Equations

TAUBnTTINm input signal width = count clock cycle × [(TAUBnCSRm.OVF × ($FFFF_H$ + 1)) + TAUBnCDRm capture value + 1]

### (3) Block diagram and general timing diagram

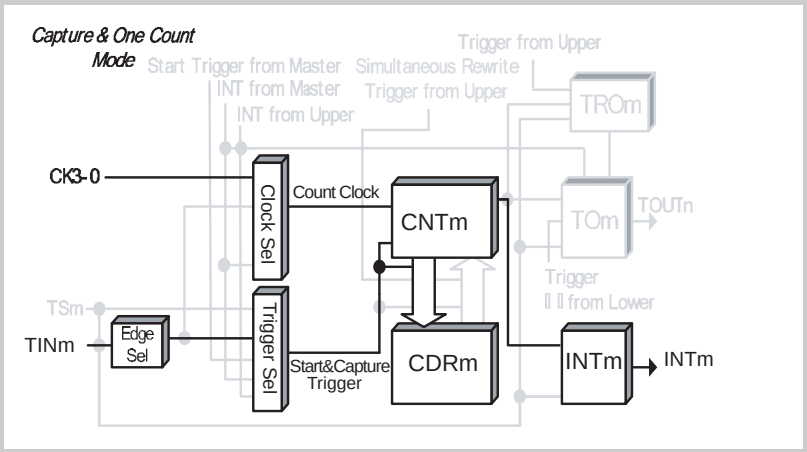



**Figure 14-36 Block diagram for TAUBnTTINm Input Signal Width Measurement Function**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUBnCMURm.TIS[1:0] = $11_B$)
- When a valid TAUBnTTINm input is detected after an overflow TAUBnCDRm is changed and TAUBnCSRm.OVF is set to 1 (TAUBnCMORm.COS[1:0] = $00_B$)

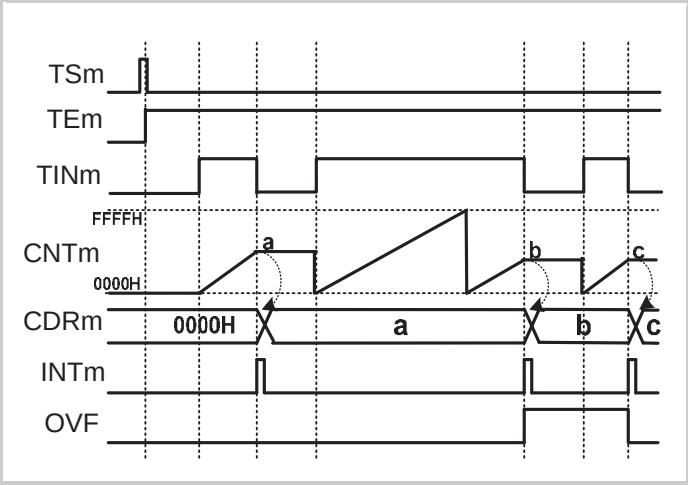



**Figure 14-37 General timing diagram for TAUBnTTINm Input Signal Width Measurement Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-39 TAUBnCMORm settings for TAUBnTTINm Input Signal Width Measurement Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUBnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | See *Table 14-38 “Effects of an overflow” on page 530* |
| MD[4:1] | 0110: Capture & One Count Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-40 TAUBnCMURm settings for TAUBnTTINm Input Signal Width Measurement Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (low width measurement)<br>11: Rising and falling edge detection (high width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the TAUBnTTINm Input Signal Width Measurement Function. Therefore, these registers must be set to 0.

**Table 14-41 Simultaneous rewrite settings for TAUBnTTINm Input Signal Width Measurement Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDS.RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDM.RDMm | |
| RDC.RDCm | |

### (5) Operating procedure for TAUBnTTINm Input Signal Width Measurement Function

**Table 14-42 Operating procedure for TAUBnTTINm Input Signal Width Measurement Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-39 "TAUBnCMORm settings for TAUBnTTINm Input Signal Width Measurement Function" on page 532* and *Table 14-40 "TAUBnCMURm settings for TAUBnTTINm Input Signal Width Measurement Function" on page 532*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and TAUBnCNTm waits for detection of the TAUBnTTINm start edge.<br>When a TAUBnTTINm start is detected, TAUBnCNTm starts to count up. |
| During operation | Detection of TAUBnTTINm edges.<br><br>The TAUBnCMURm.TIS[1:0] bits can be changed at any time.<br>The TAUBnCDRm and TAUBnCSRm registers can be read at any time. | TAUBnCNTm starts to count up from $0000_H$. When a TAUBnTTINm valid edge is detected:<br>• TAUBnCNTm transfers (captures) its value to TAUBnCDRm, and retains its value<br>• INTTAUBnIm is then generated.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and both it and TAUBnCSRm.OVF retain their current values. |

Restart

**(6) Specific timing diagrams: overflow behavior**

**(a) TAUBnCMORm.COS[1:0] = $00_B$**

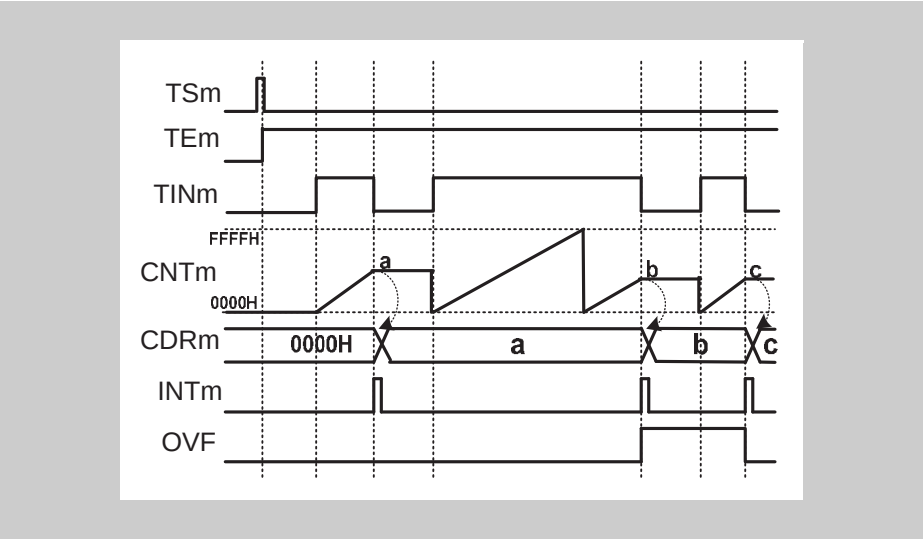



**Figure 14-38 TAUBnCMORm.COS[1:0] = $00_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, the value of TAUBnCDRm remains unchanged and TAUBnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUBnTTINm input edge, the value of TAUBnCNTm is written to TAUBnCDRm and TAUBnCSRm.OVF is set to 1.

**(b) TAUBnCMORm.COS[1:0] = $01_B$**

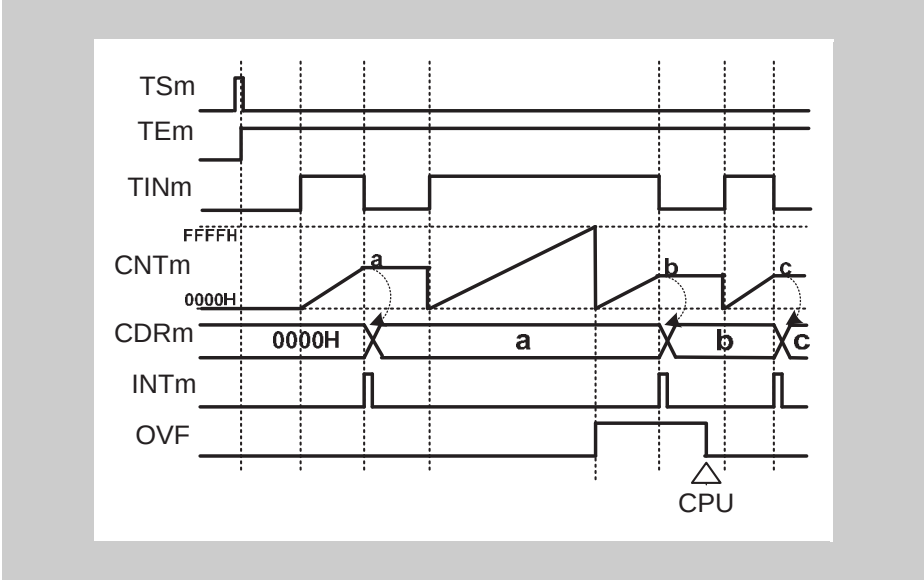



**Figure 14-39 TAUBnCMORm.COS[1:0] = $01_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, the value of TAUBnCDRm remains unchanged and TAUBnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUBnTTINm input edge, the value of TAUBnCNTm is written to TAUBnCDRm.
- TAUBnCSRm.OVF is only cleared by a CPU command.

**(c) TAUBnCMORm.COS[1:0] = $10_B$**

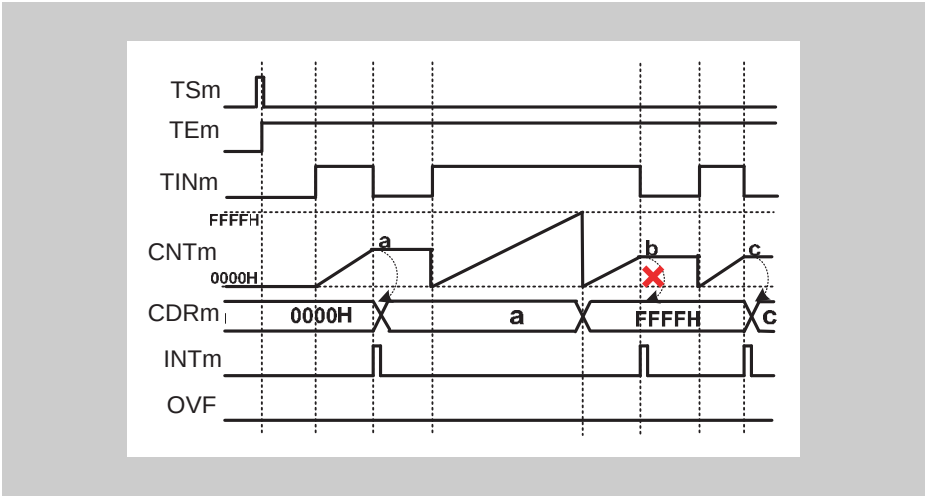



**Figure 14-40 TAUBnCMORm.COS[1:0] = $10_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, TAUBnCDRm is set to $FFFF_H$ and TAUBnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUBnTTINm input edge, TAUBnCNTm is reset to 0, but TAUBnCDRm and TAUBnCSRm.OVF remain unchanged.
- Thus, the next TAUBnTTINm input valid edge after the overflow is ignored.

**(d) TAUBnCMORm.COS[1:0] = $11_B$**

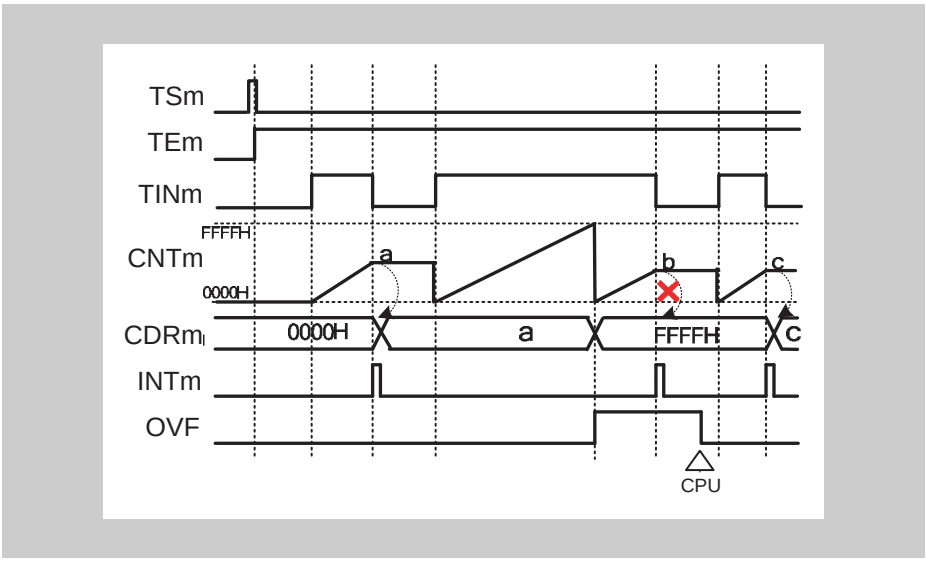



**Figure 14-41 TAUBnCMORm.COS[1:0] = $11_B$, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, TAUBnCDRm is set to $FFFF_H$, and TAUBnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUBnTTINm input edge, TAUBnCNTm is reset to 0, but TAUBnCDRm and TAUBnCSRm.OVF remain unchanged.
- Thus, the next TAUBnTTINm input valid edge after the overflow is ignored.
- TAUBnCSRm.OVF is cleared by a CPU command.

### 14.15.3 Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)

#### (1) Overview

**Summary** This function measures the width of an individual TAUBnTTINm input signal. An interrupt is generated if the TAUBnTTINm input width is longer than $FFFF_H$.

**Prerequisites**
- The operation mode must be set to One Count Mode, refer to *Table 14-43 "TAUBnCMORm settings for Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)" on page 540*
- TAUBnTTOUTm is not used for this function
- The value of TAUBnCDRm must be set to $FFFF_H$.

**Description** The counter is enabled by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation.

The counter starts when a valid TAUBnTTINm input start edge is detected. $FFFF_H$ is written to TAUBnCNTm and the counter starts to count down.

When a valid stop edge is detected, the counter stops and retains the current value.

When the next TAUBnTTINm input start edge is detected, TAUBnCNTm reloads $FFFF_H$ and starts to count down.

If the counter reaches $0000_H$ before a stop edge is detected, an interrupt is generated.

**Conditions** The valid start and stop edges are specified by the TAUBnCMURm.TIS[1:0] bits:
- If TAUBnCMURm.TIS[1:0] = $10_B$, the TAUBnTTINm input low width is measured. The start trigger is a falling edge and the stop trigger is a rising edge.
- If TAUBnCMURm.TIS[1:0] = $11_B$, the TAUBnTTINm input high width is measured. The start trigger is a rising edge and the stop trigger is a falling edge.

**Note** The counter cannot be restarted during operation.

### (2) Block diagram and general timing diagram

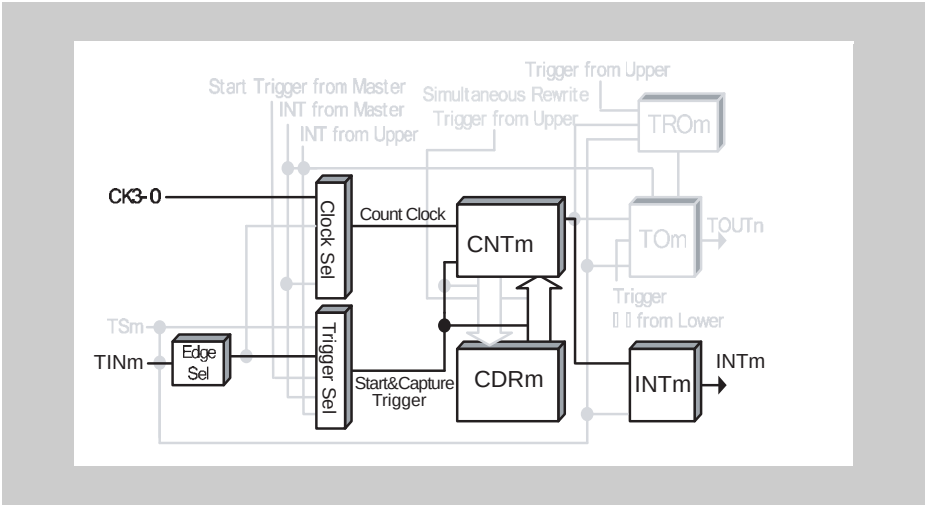



**Figure 14-42 Block diagram for Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUBnCMURm.TIS[1:0] = $11_B$)

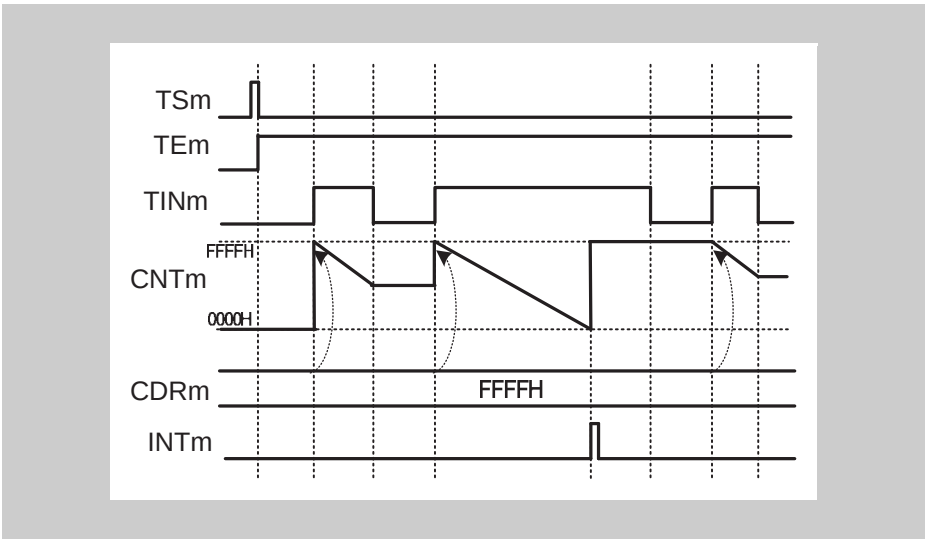



**Figure 14-43 General timing diagram for Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)**

### (3) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-43 TAUBnCMORm settings for Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUBnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-44 TAUBnCMURm settings Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (Low width measurement)<br>11: Rising and falling edge detection (High width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement). Therefore, these registers must be set to 0.

### (4) Operating procedure for Overflow Interrupt Output Function

**Table 14-45 Simultaneous rewrite settings for Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)**

| Bit name | Setting |
|---|---|
| RDEm | 0: Disables simultaneous rewrite |
| RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDMm | |
| RDCm | |

**(During TAUBnTTINm Width Measurement)**

**Table 14-46 Operating procedure for Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-43 "TAUBnCMORm settings for Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)" on page 540* and *Table 14-44 "TAUBnCMURm settings Overflow Interrupt Output Function (During TAUBnTTINm Width Measurement)" on page 540*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUBnTTINm start edge | TAUBnTE.TEm is set to 1 and TAUBnCNTm waits for detection of the start edge.<br><br>When a start edge is detected, TAUBnCNTm loads the TAUBnCDRm value ($FFFF_H$). |
| During operation | The TAUBnCNTm register can be read at any time.<br><br>Detection of TAUBnTTINm edges. | TAUBnCNTm counts down. When the counter reaches $0000_H$:<br>• INTTAUBnIm is generated<br>• TAUBnCNTm stops counting at $FFFF_H$ and waits for a trigger.<br>When a reverse edge of TAUBnTTINm is detected during count operation:<br>• TAUBnCNTm stops counting and waits for a trigger.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and retains its current value. |

Restart

### 14.15.4 TAUBnTTINm Input Period Count Detection Function

#### (1) Overview

**Summary** This function measures the cumulative width of a TAUBnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Capture & Gate Count Mode, refer to *Table 14-47 "TAUBnCMORm settings for TAUBnTTINm Input Period Count Detection Function" on page 545*
- TAUBnTTOUTm is not used for this function

**Description** The counter is enabled by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. The counter awaits a valid TAUBnTTINm input edge.

When a valid TAUBnTTINm input start edge is detected, the counter starts to count from $0000_H$.

When a valid TAUBnTTINm input stop edge is detected, the current TAUBnCNTm value is written to TAUBnCDRm and an interrupt (INTTAUBnIm) is generated. The counter stops and retains its value until the next valid TAUBnTTINm input start edge is detected.

If the counter reaches $FFFF_H$ the bit TAUBnCSRm.OVF is set to 1 and the counter restarts from $0000_H$. The value of TAUBnCSRm.OVF is reset by the CPU by setting TAUBnCSCm.CLOV = 1.

**Note** The input TAUBnTTINm is sampled at the frequency of the operation clock, specified by the TAUBnCMORm.CKS[1:0] bits. As a result, the output cycle of TAUBnTTOUTm has an error of ± 1 operation clock cycle.

**Conditions** The valid start and stop edges are specified by the TAUBnCMURm.TIS[1:0] bits:
- If TAUBnCMURm.TIS[1:0] = $10_B$, the TAUBnTTINm input low width is measured. The start trigger is a falling edge and the stop trigger is a rising edge.
- If TAUBnCMURm.TIS[1:0] = $11_B$, the TAUBnTTINm input high width is measured. The start trigger is a rising edge and the stop trigger is a falling edge.

#### (2) Equations

Cumulative TAUBnTTINm input width =
count clock cycle × (($FFFF_H$ × TAUBnCSRm.OVF) + (TAUBnCDRm capture value + 1))

### (3) Block diagram and general timing diagram

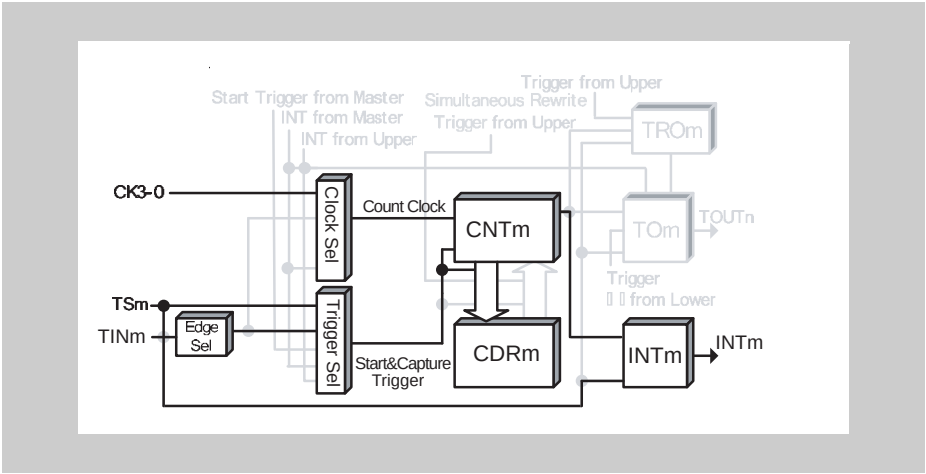



**Figure 14-44 Block diagram for TAUBnTTINm Input Period Count Detection Function**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUBnCMURm.TIS[1:0] = $11_B$)

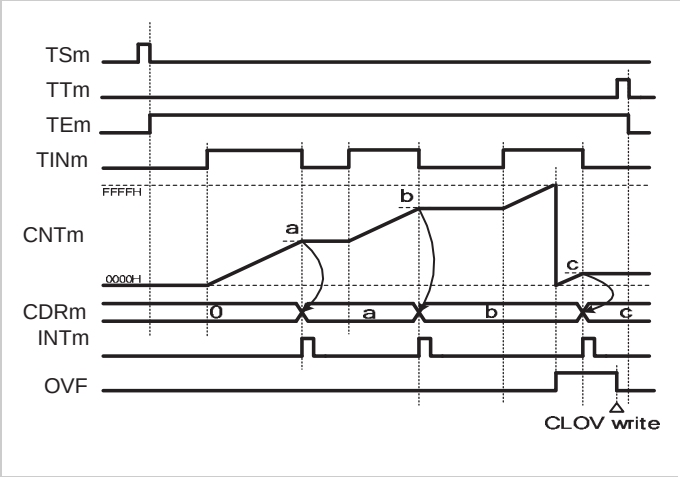



**Figure 14-45 General timing diagram for TAUBnTTINm Input Period Count Detection Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-47 TAUBnCMORm settings for TAUBnTTINm Input Period Count Detection Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUBnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | 01: Overflow (TAUBnCSRm.OVF) set upon counter overflow and cleared by a CPU instruction |
| MD[4:1] | 1101: Capture & Gate Count Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start and start trigger during count disabled |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-48 TAUBnCMURm settings for TAUBnTTINm Input Period Count Detection Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (Low width measurement)<br>11: Rising and falling edge detection (High width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the TAUBnTTINm Input Period Count Detection Function. Therefore, these registers must be set to 0.

**Table 14-49 Simultaneous rewrite settings for TAUBnTTINm Input Period Count Detection Function**

| Bit name | Setting |
|---|---|
| RDEm | 0: Disables simultaneous rewrite |
| RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDMm | |
| RDCm | |

### (5) Operating procedure for TAUBnTTINm Input Period Count Detection Function

**Table 14-50 Operating procedure for TAUBnTTINm Input Period Count Detection Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-47 "TAUBnCMORm settings for TAUBnTTINm Input Period Count Detection Function" on page 545* and *Table 14-48 "TAUBnCMURm settings for TAUBnTTINm Input Period Count Detection Function" on page 545*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUBnTTINm start edge | TAUBnTE.TEm is set to 1 and TAUBnCNTm waits for detection of the TAUBnTTINm start edge.<br><br>When a start edge is detected, TAUBnCNTm is cleared to $0000_H$ and TAUBnCNTm starts to count up. |
| During operation | Detection of TAUBnTTINm edges.<br><br>The TAUBnCDRm, TAUBnCNTm, and TAUBnCSRm registers can be read at any time.<br>The TAUBnCSC.CLOV bit can be set to 1. | When a TAUBnTTINm start edge (rising edge for high width measurement, falling edge for low width measurement) is detected, TAUBnCNTm starts to count up from the stop value.<br>When TAUBnCNTm detects a capture edge (falling edge for high width measurement, rising edge for low width measurement), it transfers the value to TAUBnCDRm and INTTAUBnIm is generated.<br>Counting stops at the "value transferred to TAUBnCDRm + 1" value and TAUBnCNTm waits for detection of the TAUBnTTINm start edge.<br>If the TAUBnCNTm reaches $FFFF_H$, the counter overflows and TAUBnCSR.OVF is set to 1.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and it and TAUBnCSRm.OVF retain their current values. |

Restart

### (6) Specific timing diagrams

#### (a) Operation stop and restart

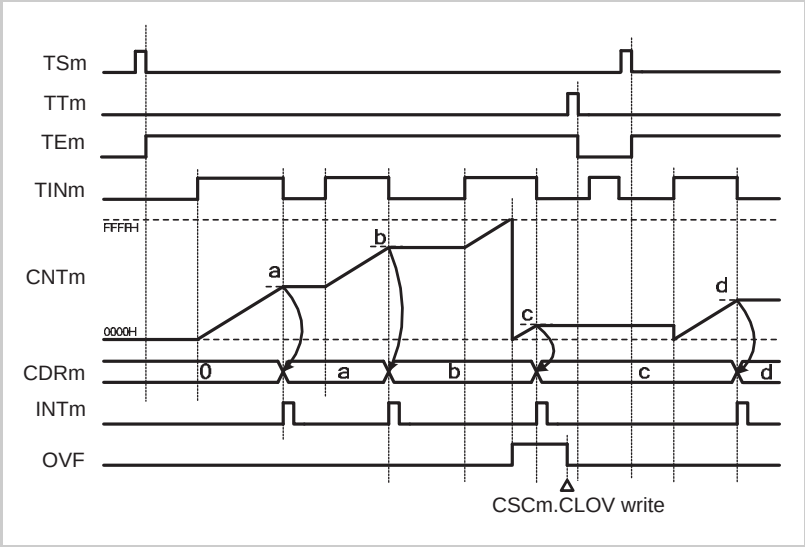



**Figure 14-46 Operation stop and restart, TAUBnCMURm.TIS[1:0] = $11_B$**

- The counter can be stopped by setting TAUBnTT.TTm to 1, which in turn sets TAUBnTE.TEm to 0.
- TAUBnCNTm stops and the current value is retained.
- If the counter is stopped, valid TAUBnTTINm input edges are ignored.
- The counter can be restarted by setting TAUBnTS.TSm to 1. TAUBnCNTm restarts to count from $0000_H$.

### 14.15.5 Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)

#### (1) Overview

**Summary** This function measures the cumulative width of a TAUBnTTINm input signal. An interrupt is generated if the cumulative TAUBnTTINm input width is longer than $FFFF_H$.

**Prerequisites**
- The operation mode must be set to Gate Count Mode, refer to *Table 14-51 "TAUBnCMORm settings for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)" on page 551*
- TAUBnTTOUTm is not used for this function
- The value of TAUBnCDRm must be set to $FFFF_H$.

**Description** The counter is enabled by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation.

The counter starts when a valid TAUBnTTINm input start edge is detected. $FFFF_H$ is written to TAUBnCNTm and the counter starts to count down.

When a valid stop edge is detected, the counter stops and retains the current value. The counter awaits the next TAUBnTTINm input start edge and then continues to count down from the current value.

When the counter reaches $0000_H$ an interrupt is generated. $FFFF_H$ is written to TAUBnCNTm and the counter continues to count down until a TAUBnTTINm input stop edge is detected.

**Conditions** The valid start and stop edges are specified by the TAUBnCMURm.TIS[1:0] bits:
- If TAUBnCMURm.TIS[1:0] = $10_B$, the TAUBnTTINm input low width is measured. The start trigger is a falling edge and the stop trigger is a rising edge.
- If TAUBnCMURm.TIS[1:0] = $11_B$, the TAUBnTTINm input high width is measured. The start trigger is a rising edge and the stop trigger is a falling edge.

**Note** The counter cannot be restarted during operation.

### (2) Block diagram and general timing diagram

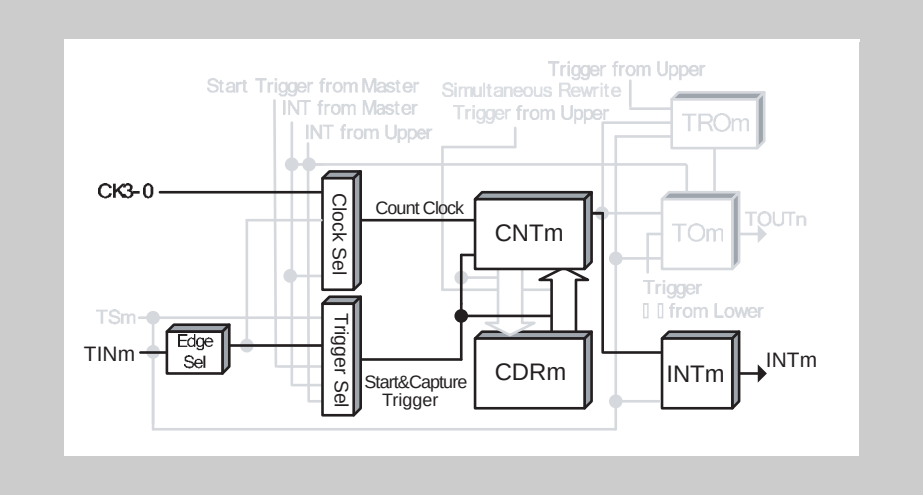



**Figure 14-47 Block diagram for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUBnCMURm.TIS[1:0] = $11_B$)

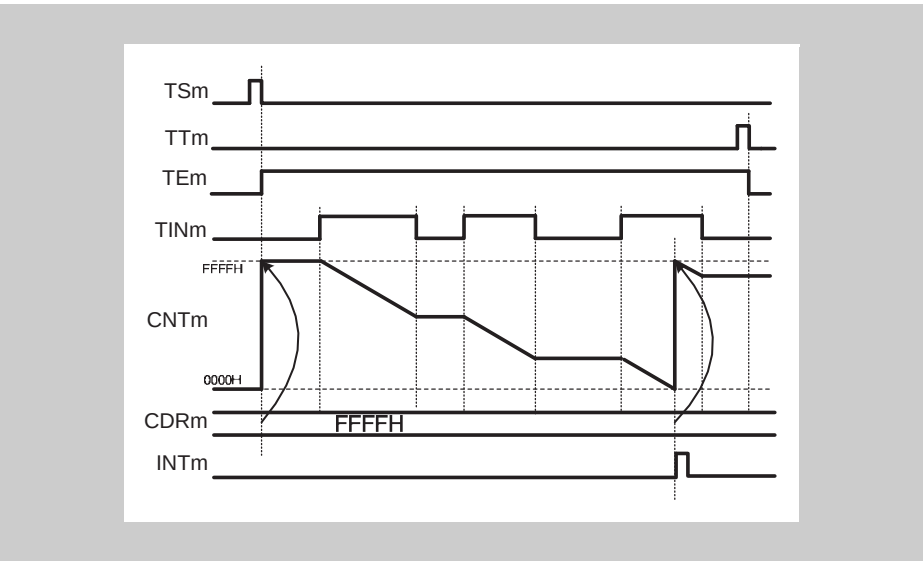



**Figure 14-48 General timing diagram for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)**

### (3) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-51 TAUBnCMORm settings for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUBnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 1100: Gate Count Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-52 TAUBnCMURm settings for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (Low width measurement)<br>11: Rising and falling edge detection (High width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection). Therefore, these registers must be set to 0.

**Table 14-53 Simultaneous rewrite settings for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)**

| Bit name | Setting |
|---|---|
| RDEm | 0: Disables simultaneous rewrite |
| RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDMm | |
| RDCm | |

### (4) Operating procedure for Overflow Interrupt Output Function (during TAUBnTTINm input period count detection)

**Table 14-54 Operating procedure for Overflow Interrupt Output Function (during TAUBnTTINm input period count detection)**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-51 "TAUBnCMORm settings for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)" on page 551* and *Table 14-52 "TAUBnCMURm settings for Overflow Interrupt Output Function (During TAUBnTTINm Input Period Count Detection)" on page 551*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUBnTTINm start edge | TAUBnTE.TEm is set to 1 and TAUBnCNTm waits for detection of the start edge.<br><br>When a start edge is detected, TAUBnCNTm loads the TAUBnCDRm value ($FFFF_H$). |
| During operation | The TAUBnCNTm register can be read at all times. | TAUBnCNTm counts down. When the counter reaches $0000_H$:<br>• INTTAUBnIm is generated<br>• TAUBnCNTm loads the TAUBnCDRm value ($FFFF_H$) and continues to count down.<br>When a reverse edge of TAUBnTTINm is detected during count operation:<br>• TAUBnCNTm counts down from the stop value.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and retains its current value. |

Restart

### 14.15.6 TAUBnTTINm Input Pulse Interval Judgment Function

#### (1) Overview

**Summary** This function outputs the result of a comparison between the count value (TAUBnCNTm) and the value in the channel data register (TAUBnCDRm) when a TAUBnTTINm input pulse occurs. An interrupt signal INTTAUBnIm is generated if the result of the comparison is true.

**Prerequisites**
- The operation mode must be set to Judge Mode, refer to *Table 14-55 "TAUBnCMORm settings for TAUBnTTINm Input Pulse Interval Judgment Function" on page 556*
- TAUBnTTOUTm is not used for this function

**Description** The counter is started by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. The current value of TAUBnCDRm is written to TAUBnCNTm and the counter starts to count down from this value.

When a TAUBnTTINm valid edge is detected or TAUBnTS.TSm is set to 1, the function compares the current values of TAUBnCNTm and TAUBnCDRm. An interrupt signal INTTAUBnIm is generated if the result of the comparison is true. TAUBnCNTm reloads the value of TAUBnCDRm and subsequently continues operation, regardless of the result of the comparison.

If the counter reaches $0000_H$ before a TAUBnTTINm valid edge is detected, TAUBnCNTm overflows and is set to $FFFF_H$. It then continues to count down.

The value of TAUBnCDRm can be rewritten at any time, and the changed value of TAUBnCDRm is applied the next time the function starts to count down.

**Conditions** The TAUBnCMORm.MD0 bit specifies the type of comparison:
- If TAUBnCMORm.MD0 = 0, INTTAUBnIm is generated when TAUBnCNTm ≤ TAUBnCDRm.
- If TAUBnCMORm.MD0 = 1, INTTAUBnIm is generated when TAUBnCNTm > TAUBnCDRm.

### (2) Block diagram and general timing diagram

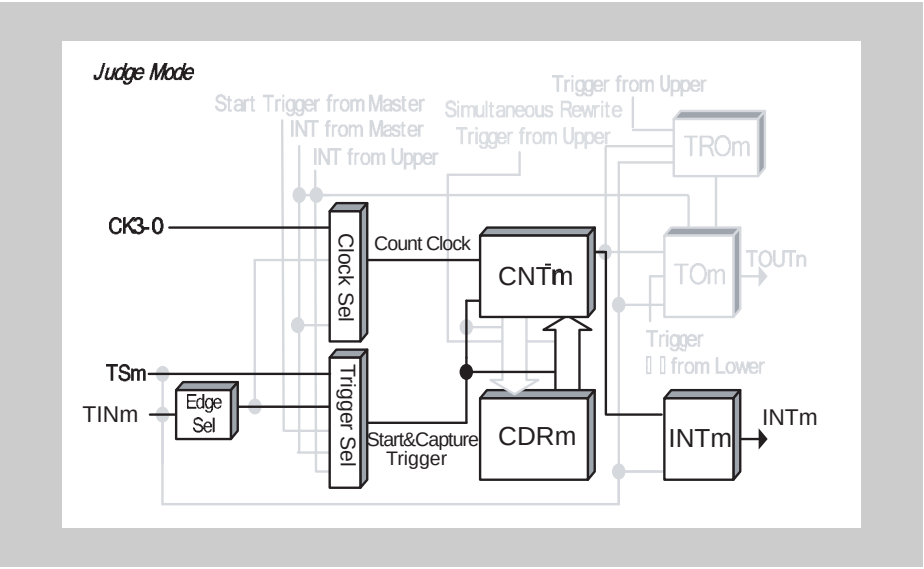



**Figure 14-49 Block diagram for TAUBnTTINm Input Pulse Interval Judgment Function**

The following settings apply to the general timing diagram:

- INTTAUBnIm not generated at operation start (TAUBnCMORm.MD0 = 0)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

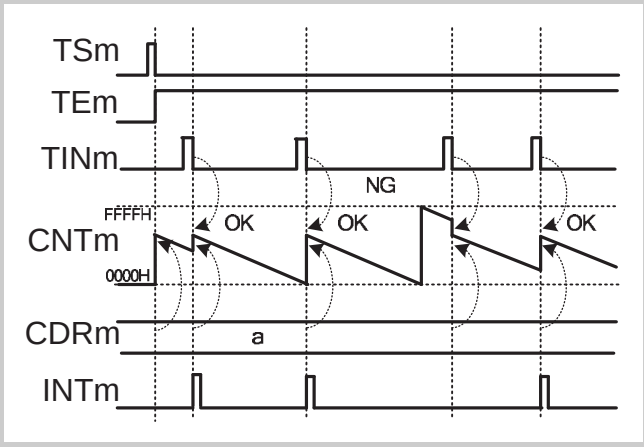



**Figure 14-50 General timing diagram for TAUBnTTINm Input Pulse Interval Judgment Function**

### (3) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-55 TAUBnCMORm settings for TAUBnTTINm Input Pulse Interval Judgment Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid edge of the TAUBnTTINm input signal is the external start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0001: Judge Mode |
| MD0 | 0: INTTAUBnIm is generated when TAUBnCNTm ≤ TAUBnCDRm<br>1: INTTAUBnIm is generated when TAUBnCNTm > TAUBnCDRm |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-56 TAUBnCMURm settings for TAUBnTTINm Input Pulse Interval Judgment Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection (low width measurement)<br>11: Rising and falling edge detection (high width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the TAUBnTTINm Input Pulse Interval Judgment Function. Therefore, these registers must be set to 0.

**Table 14-57 Simultaneous rewrite settings for TAUBnTTINm Input Pulse Interval Judgment Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDS.RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDM.RDMm | |
| RDC.RDCm | |

### (4) Operating procedure for for TAUBnTTINm Input Pulse Interval Judgment Function

**Table 14-58 Operating procedure for TAUBnTTINm Input Pulse Interval Judgment Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-55 "TAUBnCMORm settings for TAUBnTTINm Input Pulse Interval Judgment Function" on page 556* and *Table 14-56 "TAUBnCMURm settings for TAUBnTTINm Input Pulse Interval Judgment Function" on page 556*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>TAUBnCNTm loads the TAUBnCDRm value. |
| During operation | Detection of TAUBnTTINm edges.<br><br>The value of TAUBnCDRm can be changed at any time.<br><br>The TAUBnCNTm register can be read at any time. | TAUBnCNTm counts down. When a TAUBnTTINm input edge is detected<br>• TAUBnCNTm reloads TAUBnCDRm and continues count operation.<br>• TAUBnCNTm compares the values and judges the condition according to the TAUBnCMORm.MD0 setting.<br>• If the condition is satisfied, INTTAUBnIm is generated.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and retains its current value. |

Restart

## 14.15.7 TAUBnTTINm Input Signal Width Judgment Function

### (1) Overview

**Summary**

This function outputs the result of a comparison between the count value (TAUBnCNTm) and the value in the channel data register (TAUBnCDRm) at a valid stop edge of a TAUBnTTINm input signal. An interrupt signal INTTAUBnIm is generated if the result of the comparison is true.

**Prerequisites**

- The operation mode must be set to Judge & One Count Mode, refer to *Table 14-59 "TAUBnCMORm settings for TAUBnTTINm Input Signal Width Judgment Function" on page 561*
- TAUBnTTOUTm is not used for this function

**Description**

The counter is started by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. When a valid TAUBnTTINm input start edge is detected, the current value of TAUBnCDRm is written to TAUBnCNTm and the counter starts to count down from this value.

When a TAUBnTTINm valid stop edge is detected, the function compares the current values of TAUBnCNTm and TAUBnCDRm. An interrupt signal INTTAUBnIm is generated if the result of the comparison is true. The counter TAUBnCNTm retains its value until the next TAUBnTTINm valid start edge is detected, regardless of the result of the comparison.

If the counter reaches $0000_H$ before a valid TAUBnTTINm stop edge is detected, TAUBnCNTm overflows and is set to $FFFF_H$. It then continues to count down.

The value of TAUBnCDRm can be rewritten at any time, and the changed value of TAUBnCDRm is applied the next time the function starts to count down.

**Conditions**

- The TAUBnCMORm.MD0 bit specifies the type of comparison:
  - If TAUBnCMORm.MD0 = 0, INTTAUBnIm is generated when TAUBnCNTm ≤ TAUBnCDRm.
  - If TAUBnCMORm.MD0 = 1, INTTAUBnIm is generated when TAUBnCNTm > TAUBnCDRm.
- The TAUBnCMURm.TIS[1:0] bits specify the type of width measurement:
  - For high width measurement, the start edge is a rising TAUBnTTINm edge and the stop edge is a falling TAUBnTTINm edge.
  - For low width measurement, the start edge is a falling TAUBnTTINm edge and the stop edge is a rising TAUBnTTINm edge.
- Forced restart is not possible for this function.

### (2) Block diagram and general timing diagram

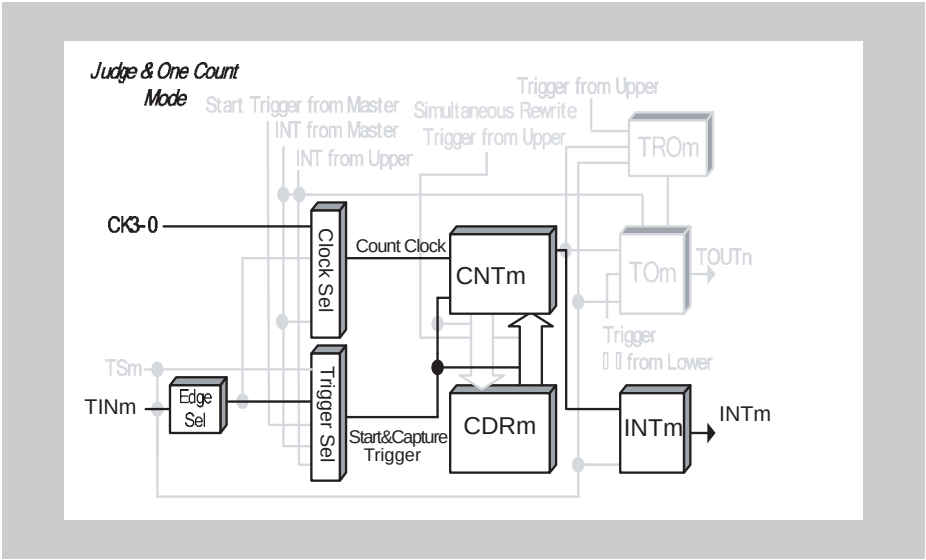



**Figure 14-51 Block diagram for TAUBnTTINm Input Signal Width Judgment Function**

The following settings apply to the general timing diagram:

- INTTAUBnIm is generated when TAUBnCNTm ≤ TAUBnCDRm (TAUBnCMORm.MD0 = 0)
- TAUBnTTINm valid start edge = rising edge, TAUBnTTINm valid stop edge = falling edge (TAUBnCMURm.TIS[1:0] = $11_B$)

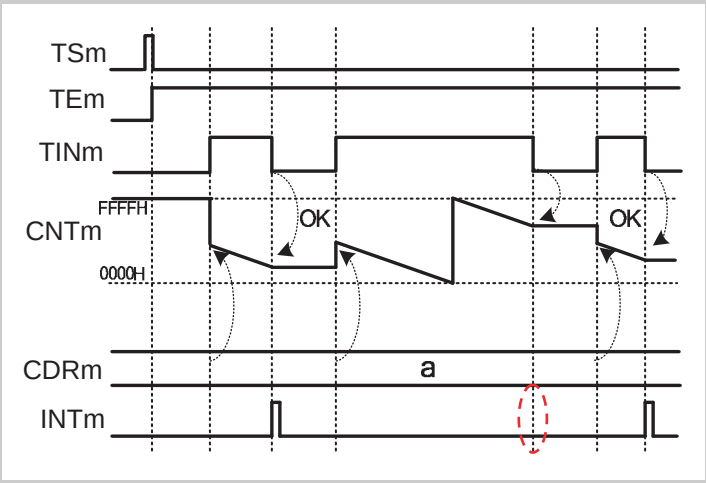



**Figure 14-52 General timing diagram for TAUBnTTINm Input Signal Width Judgment Function**

### (3) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-59 TAUBnCMORm settings for TAUBnTTINm Input Signal Width Judgment Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUBnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0111: Judge & One Count Mode |
| MD0 | 0: INTTAUBnIm is generated when TAUBnCNTm ≤ TAUBnCDRm<br>1: INTTAUBnIm is generated when TAUBnCNTm > TAUBnCDRm |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-60 TAUBnCMURm settings for TAUBnTTINm Input Signal Width Judgment Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (low width measurement)<br>11: Rising and falling edge detection (high width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the TAUBnTTINm Input Signal Width Judgment Function. Therefore, these registers must be set to 0.

**Table 14-61 Simultaneous rewrite settings for TAUBnTTINm Input Signal Width Judgment Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDS.RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDM.RDMm | |
| RDC.RDCm | |

### (4) Operating procedure for TAUBnTTINm Input Signal Width Judgment Function

**Table 14-62 Operating procedure for TAUBnTTINm Input Signal Width Judgment Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-59 "TAUBnCMORm settings for TAUBnTTINm Input Signal Width Judgment Function" on page 561* and *Table 14-60 "TAUBnCMURm settings for TAUBnTTINm Input Signal Width Judgment Function" on page 561*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUBnTTINm start edge | TAUBnTE.TEm is set to 1 and TAUBnCNTm waits for detection of the TAUBnTTINm start edge.<br><br>When a TAUBnTTINm start edge is detected, TAUBnCNTm loads the TAUBnCDRm value. |
| During operation | Detection of TAUBnTTINm edges<br><br>The value of TAUBnCDRm can be changed at any time.<br><br>The TAUBnCNTm register can be read at any time. | TAUBnCNTm counts down. When a TAUBnTTINm stop edge is detected:<br>• TAUBnCNTm stops and waits for detection of the TAUBnTTINm start edge.<br>• TAUBnCNTm compares the values and judges the condition according to the TAUBnCMORm.MD0.<br>• If the condition is satisfied, INTTAUBnIm is generated<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and retains its current value. |

Restart

## 14.16 Independent Channel Simultaneous Rewrite Functions

This chapter describes functions that carry out simultaneous rewrite:

- *14.16.1 “Simultaneous Rewrite Trigger Generation Function Type 1”*

### 14.16.1 Simultaneous Rewrite Trigger Generation Function Type 1

#### (1) Overview

**Summary** This function generates an interrupt on a specific channel that can be used by lower channels as a simultaneous rewrite trigger. The interrupt is generated at regular intervals.

**Prerequisites**

- Two (or more) channels, each with simultaneous rewrite enabled (TAUBnRDE.RDEm = 1)
- The operation mode of the upper channel must be set to Interval Timer Mode, refer to *Table 14-63 “TAUBnCMORm settings for Simultaneous Rewrite Trigger Generation Function Type 1” on page 568*
- The operation mode of the lower channel(s) can be set as desired

**Description** The counters are started by setting the channel trigger bits (TAUBnTS.TSm and TAUBnTS.TSm_lower) to 1. This in turn sets TAUBnTE.TEm and TAUBnTE.TEm_lower = 1, enabling count operation. The current value of the data register buffer of the upper channel (TAUBnCDRm buf) is written to the counter (TAUBnCNTm) and the counter starts to count down from this value. The counter(s) of the lower channel(s) start to count as specified by their selected operating modes.

When a counter reaches $0000_H$, an interrupt is generated from the channel. The corresponding TAUBnCNTm then reloads the current TAUBnCDRm buffer value and subsequently continues operation.

If the channel where the interrupt occurs is specified as the trigger channel for simultaneous rewrite (TAUBnRDC.RDCm = 1) and is an upper channel, simultaneous rewrite takes place on all lower channels in which simultaneous rewrite is currently possible (TAUBnRSF.RSFm = 1).

The values of the data registers are copied to the corresponding data register buffers. Each time a counter starts to count down, it reads the value in the data register buffer and counts down from this value.

The value of a data register can be changed at any time, but it is only transferred to the corresponding data register buffer when simultaneous rewrite occurs.

**Conditions**

- The channel which is monitored for INTTAUBnIm is specified by setting TAUBnRDC.RDCm = 1 for the corresponding channel. The TAUBnRDC.RDCm bit must be 0 for all other channels in which simultaneous rewrite should take place.
- If the TAUBnCMORm.MD0 bit is set to 0, the first interrupt after a start or restart is not generated. For details refer to *14.11 “TAUBnTTOUTm toggle and INTTAUBnIm Generation when Counter start is triggered (MD0-bit)” on page 498* .

#### (2) Equations

Simultaneous rewrite trigger generation cycle = count clock cycle × (TAUBnCDRm + 1)

To control simultaneous rewrite, the following condition must be satisfied:

TAUBnCDRm = [(value of TAUBnCDRm of master channel subject to simultaneous rewrite + 1) × number of interrupts] - 1

That is, the ratio of TAUBnCDRm + 1 and TAUBnCDRm_master + 1 must be an integer. This integer corresponds to the number of interrupts.

### (3) Block diagram and general timing diagram

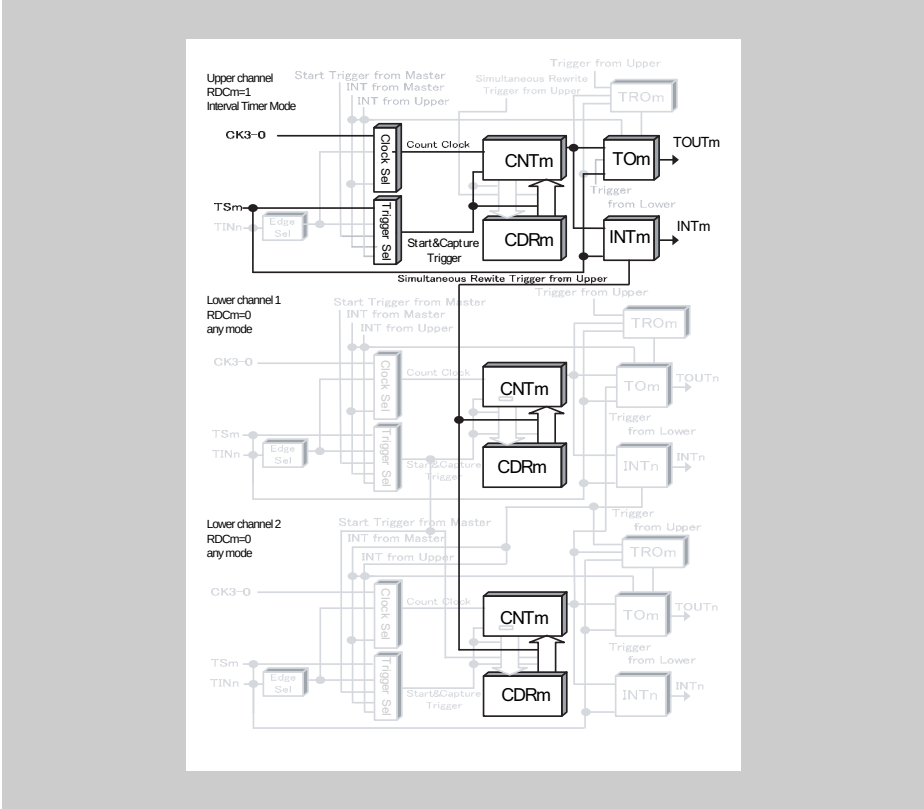



**Figure 14-53 Block diagram for Simultaneous Rewrite Trigger Generation Function Type 1**

The following settings apply to the general timing diagram:

- INTTAUBnIm generated at operation start (TAUBnCMORm.MD0 = 1)

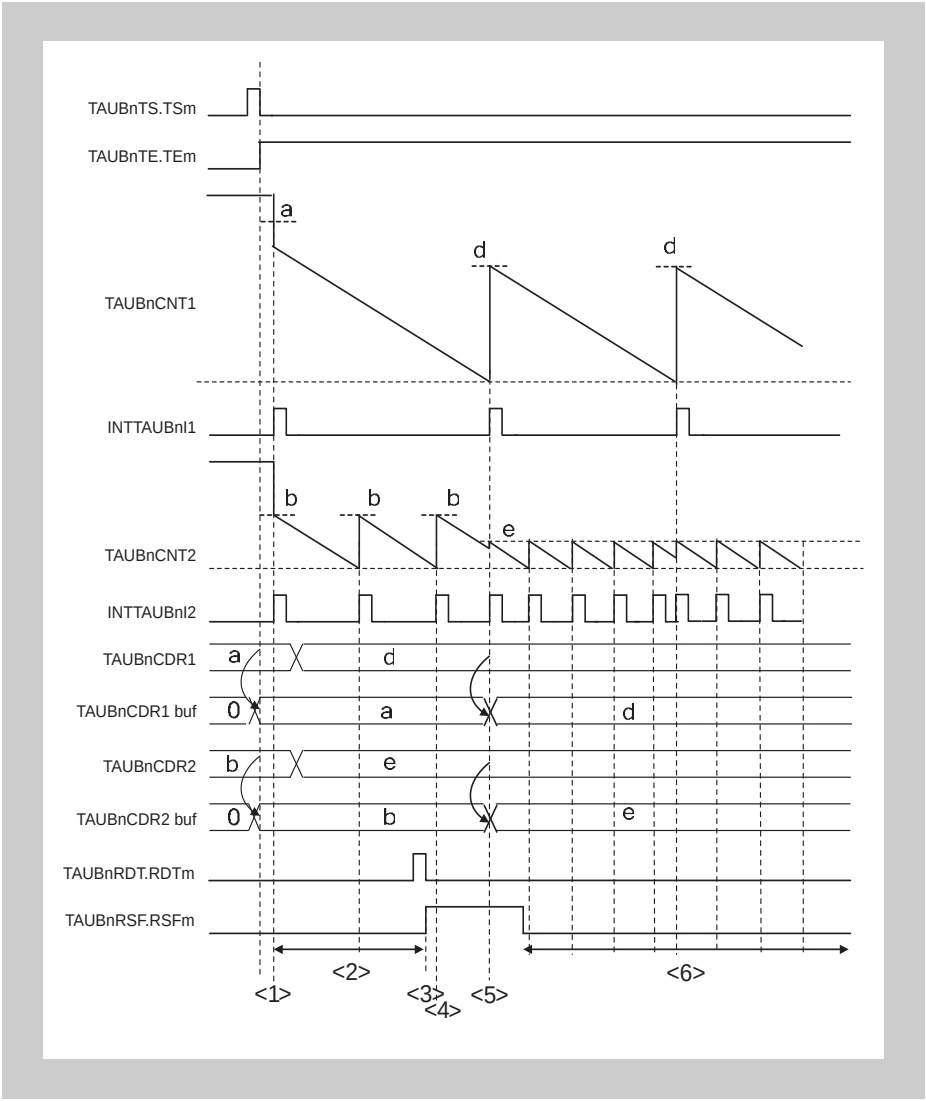



**Figure 14-54 General timing diagram for Simultaneous Rewrite Trigger Generation Function Type 1**

### (4) Register settings for the upper channel

#### (a) TAUBnCMORm for the upper channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-63 TAUBnCMORm settings for Simultaneous Rewrite Trigger Generation Function Type 1**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start<br>1: Generates INTTAUBnIm at operation start |

#### (b) TAUBnCMURm for the upper channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-64 TAUBnCMURm settings for Simultaneous Rewrite Trigger Generation Function Type 1**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used, so set to 00 |

**(c) Channel output mode for the upper channel**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite for the upper channel**

**Table 14-65 Simultaneous rewrite settings for Simultaneous Rewrite Trigger Generation Function Type 1**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 1: Selects an upper channel as the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The signal that controls simultaneous rewrite is loaded when the master channel starts counting |
| RDC.RDCm | 1: Channel is monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger |

**(5) Register settings for the lower channel(s)**

**(a) TAUBnCMORm for the lower channel(s)**

The TAUBnCMORm register of the lower channel(s) can be set arbitrarily.

**(b) TAUBnCMURm for the lower channel(s)**

The TAUBnCMURm register of the lower channel(s) can be set arbitrarily.

**(c) Channel output mode for the lower channel(s)**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite for the lower channel(s)**

**Table 14-66 Simultaneous rewrite settings for the lower channel in Simultaneous Rewrite Trigger Generation Function Type 1**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 1: Selects an upper channel as the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The signal that controls simultaneous rewrite is loaded when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous-rewrite trigger |

### (6) Operating procedure for Simultaneous Rewrite Trigger Generation Function Type 1

**Table 14-67 Operating procedure for Simultaneous Rewrite Trigger Generation Function Type 1**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers for the upper channel as described in *Table 14-63 "TAUBnCMORm settings for Simultaneous Rewrite Trigger Generation Function Type 1" on page 568* and *Table 14-64 "TAUBnCMURm settings for Simultaneous Rewrite Trigger Generation Function Type 1" on page 568*<br><br>Set the TAUBnCMORm register and TAUBnCMURm registers for the lower channel as described in *5 "Register settings for the lower channel(s)"*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>TAUBnCNTm loads the TAUBnCDRm value.<br>When TAUBnCMORm.MD0 = 1, INTTAUBnIm is generated and TAUBnTTOUTm toggles. |
| During operation | TAUBnRDT.RDTm can be changed.<br>TAUBnRSF.RSFm can be read at all times. | TAUBnCNTm counts down. When the counter reaches $0000_H$:<br>• TAUBnCNTm reloads the TAUBnCDRm value and continues count operation<br>• INTTAUBnIm is generated<br>• TAUBnTTOUTm toggles.<br>Simultaneous rewrite is controlled when INTTAUBnIm is generated from the channel where TAUBnRDC.RDCm is set to 1.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and both it and TAUBnTTOUTm retain their current values.<br><br>When TAUBnTOE.TOEm is 0, TAUBnTTOUTm output is initialized to the value set by TAUBnTO.TOm. |

Restart

## 14.17 Other Independent Channel Functions

This chapter describes a function that generates an interrupt when a certain number of TAUBnTTINm pulses has occurred, a function that divides the frequency of TAUBnTTINm, and a function that measures the duration between the function start and a TAUBnTTINm input signal:

- *14.17.1 “External Event Count Function”*
- *14.17.2 “Clock Divide Function”*
- *14.17.3 “TAUBnTTINm Input Position Detection Function”*

### 14.17.1 External Event Count Function

#### (1) Overview

**Summary** This function is used as an event timer. It generates an interrupt (INTTAUBnIm) when a specific number of TAUBnTTINm input pulses has occurred.

**Prerequisites**
- The operation mode must be set to Event Count Mode, refer to *Table 14-68 "TAUBnCMORm settings for External Event Count Function" on page 574*
- TAUBnTTOUTm is not used for this function

**Description** The counter is enabled by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. When the counter starts, the current value of TAUBnCDRm is written to TAUBnCNTm.

When a valid TAUBnTTINm input edge is detected, the value of TAUBnCNTm reduces by 1. TAUBnCNTm retains this value until a valid TAUBnTTINm input edge is detected or the counter is restarted.

When the counter value reaches $0000_H$, INTTAUBnIm is generated. TAUBnCNTm then reloads the TAUBnCDRm value and subsequently continues operation.

The counter can be stopped by setting TAUBnTT.TTm to 1, which in turn sets TAUBnTE.TEm to 0. TAUBnCNTm stops and retains its value. The counter can be restarted by setting TAUBnTS.TSm to 1. The counter can also be restarted without stopping it first (forced restart) by setting TAUBnTS.TSm to 1 during operation.

The value of TAUBnCDRm can be rewritten at any time, and the changed value of TAUBnCDRm is applied the next time the counter starts to count down.

**Conditions** The type of edge used as the trigger is specified by the TAUBnCMURm.TIS[1:0] bits:
- If TAUBnCMURm.TIS[1:0] = $00_B$, the falling edges are counted.
- If TAUBnCMURm.TIS[1:0] = $01_B$, the rising edges are counted.
- If TAUBnCMURm.TIS[1:0] = $10_B$, the rising and falling edges are counted.

#### (2) Equations

Number of valid edges,
detected before INTTAUBnIm is generated = TAUBnCDRm + 1

### (3) Block diagram and general timing diagram

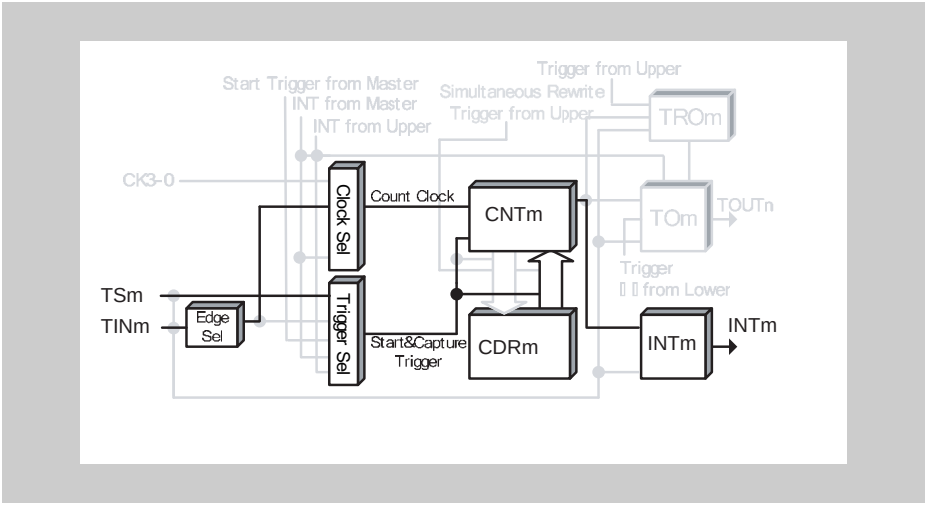



**Figure 14-55 Block diagram for External Event Count Function**

The following settings apply to the general timing diagram:

- Rising edge detection (TAUBnCMURm.TIS[1:0] = $01_B$)

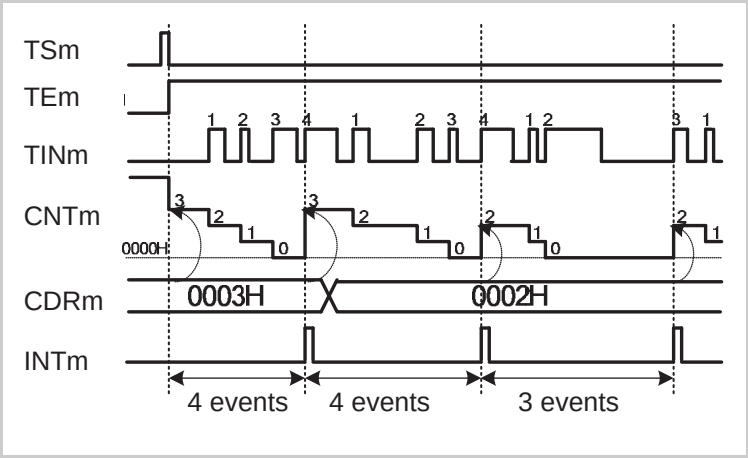



**Figure 14-56 General timing diagram for External Event Count Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

Table 14-68 TAUBnCMORm settings for External Event Count Function

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 1: Valid TAUBnTTINm input edge is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0011: Event Count Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

Table 14-69 TAUBnCMURm settings for External Event Count Function

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge<br>01: Rising edge<br>10: Rising and falling edge |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the External Event Count Function. Therefore, these registers must be set to 0.

**Table 14-70 Simultaneous rewrite settings for External Event Count Function**

| Bit name | Setting |
|---|---|
| RDEm | 0: Disables simultaneous rewrite |
| RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDMm | |
| RDCm | |

### (5) Operating procedure for External Event Count Function

Table 14-71 Operating procedure for External Event Count Function

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-68 "TAUBnCMORm settings for External Event Count Function" on page 574* and *Table 14-69 "TAUBnCMURm settings for External Event Count Function" on page 574*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>TAUBnCNTm loads the TAUBnCDRm value and waits for detection of the TAUBnTTINm input edge. |
| During operation | Detection of TAUBnTTINm edges.<br><br>The value of TAUBnCDRm can be changed at any time.<br><br>The TAUBnCNTm register can be read at any time. | TAUBnCNTm performs count-down operation each time a TAUBnTTINm input edge is detected. When the counter reaches $0000_H$:<br>• TAUBnCNTm reloads the TAUBnCDRm value and continues count operation<br>• INTTAUBnIm is generated.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and retains its current value. |

Restart

### (6) Specific timing diagrams

#### (a) TAUBnCDRm = $0000_H$

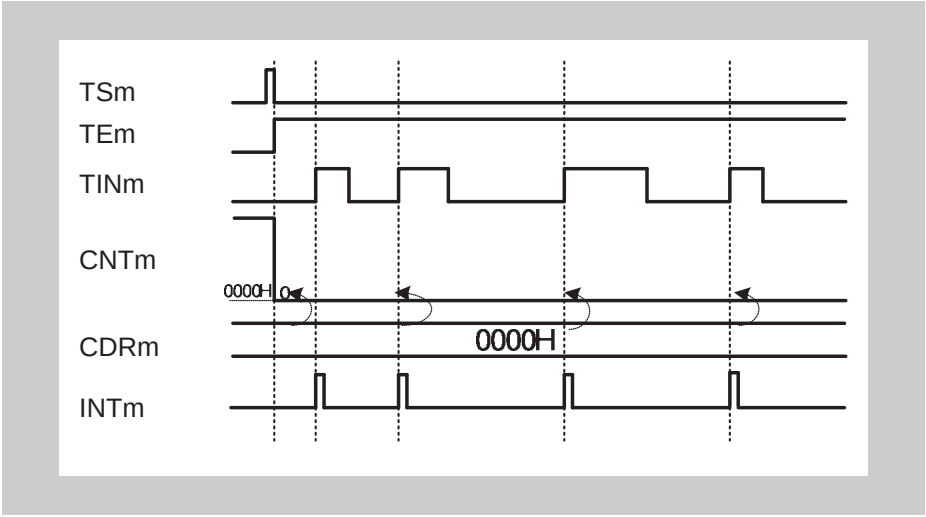



**Figure 14-57 TAUBnCDRm = $0000_H$, TAUBnCMURm.TIS[1:0] = 01**

- If $0000_H$ = TAUBnCDRm, $0000_H$ is written to TAUBnCNTm every time a valid TAUBnTTINm input edge is detected.

  This means, INTTAUBnIm is generated every time a valid TAUBnTTINm input edge is detected.

#### (b) Operation stop and restart

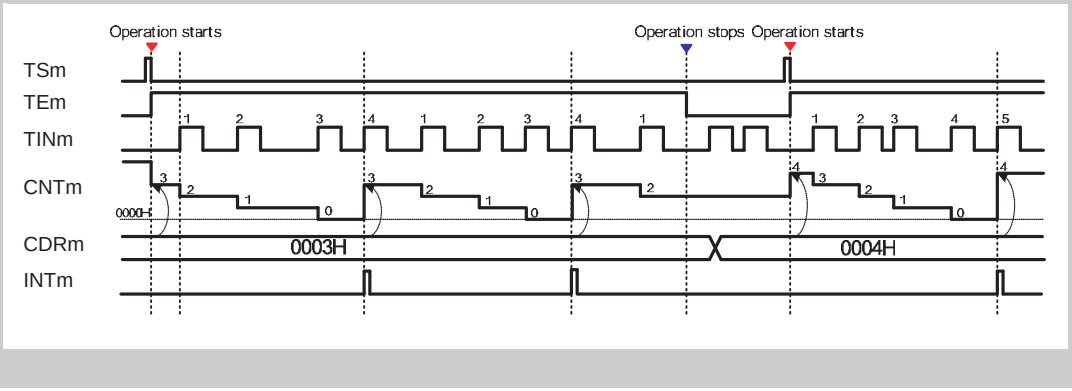



**Figure 14-58 Operation stop and restart, TAUBnCMURm.TIS[1:0] = 01**

- The counter can be stopped by setting TAUBnTT.TTm to 1, which in turn sets TAUBnTE.TEm to 0.
- TAUBnCNTm stops and the current value is retained. TAUBnTTINm continues and TAUBnCNTm ignores the valid edge.
- The counter can be restarted by setting TAUBnTS.TSm to 1. TAUBnCNTm loads the TAUBnCDRm value and restarts count operation.

### (c) Forced restart

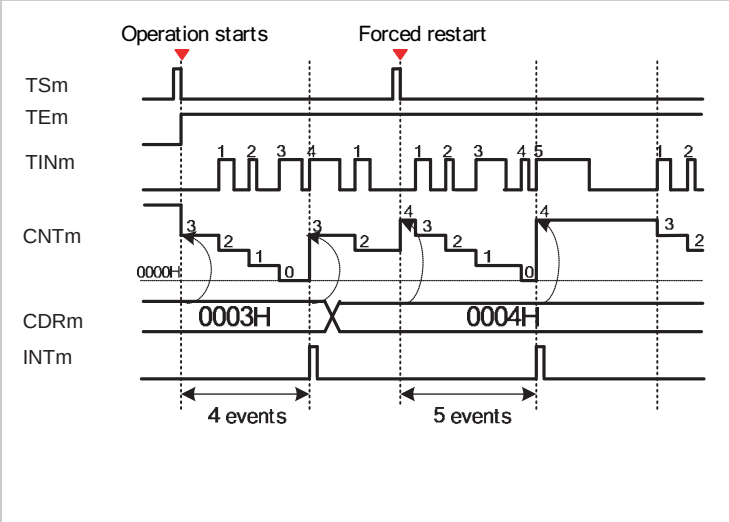



**Figure 14-59 Forced restart, TAUBnCMURm.TIS[1:0] = 01**

A forced restart applies a change to TAUBnCDRm immediately.

- The counter can be restarted (without stopping it first), by setting TAUBnTS.TSm to 1 during operation.
- The value of TAUBnCDRm is written to TAUBnCNTm and the counter awaits the next valid TAUBnTTINm input edge.

## 14.17.2 Clock Divide Function

### (1) Overview

**Summary** This function is used as a frequency divider. The frequency of the input signal TAUBnTTINm is divided by a factor related to TAUBnCDRm, and the resulting signal is output to TAUBnTTOUTm.

**Prerequisites**

- TAUBnTTINm must have a fixed frequency
- The operation mode must be set to Interval Timer Mode, refer to *Table 14-72 “TAUBnCMORm settings for Clock Divide Function” on page 581*
- The channel output mode must be set to Independent Channel Output Mode 1, refer to *14.9 “Channel Output Modes” on page 489*

**Description** The counter is started by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. The current value of TAUBnCDRm is written to TAUBnCNTm and the counter starts to count down from this value, using TAUBnTTINm as the count clock.

When the counter value reaches $0000_H$, INTTAUBnIm is generated and the TAUBnTTOUTm signal toggles. TAUBnCNTm then reloads the TAUBnCDRm value and subsequently continues operation.

The value of TAUBnCDRm can be rewritten at any time, and the changed value of TAUBnCDRm is applied the next time the function starts to count down.

The counter can be stopped by setting TAUBnTT.TTm = 1, which in turn sets TAUBnTE.TEm = 0. TAUBnCNTm and TAUBnTTOUTm stop but retain their values. The function can be restarted by setting TAUBnTS.TSm = 1. The counter can also be forcibly restarted (without stopping it first) by setting TAUBnTS.TSm = 1 during operation.

**Conditions** If the TAUBnCMORm.MD0 bit is set to 0, the first interrupt after a start or restart is not generated, and therefore TAUBnTTOUTm does not toggle. This results in an inverted TAUBnTTOUTm signal compared to when TAUBnCMORm.MD0 is set to 1. For details refer to *14.11 “TAUBnTTOUTm toggle and INTTAUBnIm Generation when Counter start is triggered (MD0-bit)” on page 498* .

**Note** The input TAUBnTTINm is sampled at the frequency of the operation clock, specified by TAUBnCMORm.CKS[1:0] bits. As a result, the output cycle of TAUBnTTOUTm has an error of ± 1 operation clock cycle.

### (2) Equations

- When rising edge detection is selected:
  TAUBnTTOUTm frequency = TAUBnTTINm frequency / [(TAUBnCDRm + 1) × 2]
- When falling edge detection is selected:
  TAUBnTTOUTm frequency = TAUBnTTINm frequency / [(TAUBnCDRm + 1) × 2]
- When rising and falling edge detection is selected:
  TAUBnTTOUTm frequency = TAUBnTTINm frequency / (TAUBnCDRm + 1)

### (3) Block diagram and general timing diagram

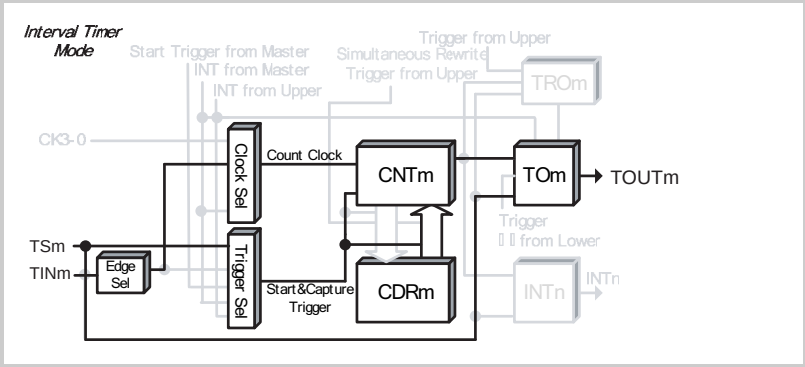



**Figure 14-60 Block diagram for Clock Divide Function**

The following settings apply to the general timing diagram:

- INTTAUBnIm generated at operation start (TAUBnCMORm.MD0 = 1)
- Rising edge detection (TAUBnCMURm.TIS[1:0] = $01_B$)

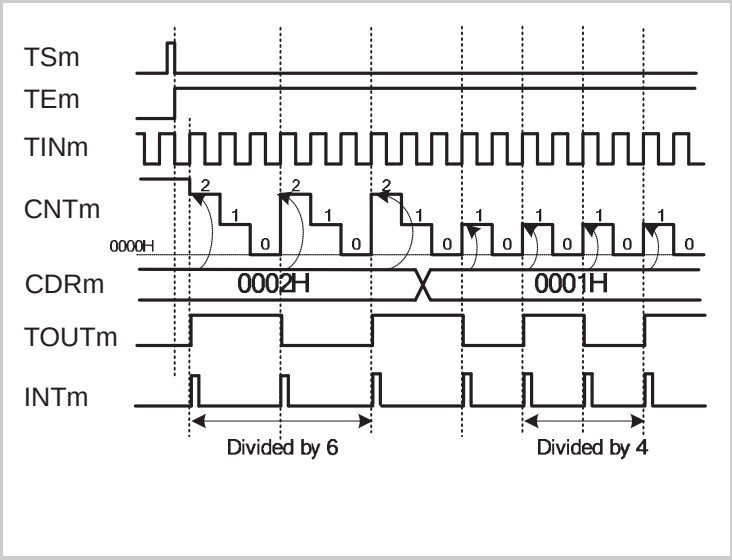



**Figure 14-61 General timing diagram for Clock Divide Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

Table 14-72 TAUBnCMORm settings for Clock Divide Function

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 1: Valid TAUBnTTINm input edge is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUBnIm not generated and TAUBnTTOUTm does not toggle at operation start<br>1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

Table 14-73 TAUBnCMURm settings for Clock Divide Function

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection |

### (c) Channel output mode

**Table 14-74 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOE.TOEm | 1: Disables Direct Channel Output Mode |
| TOM.TOMm | 0: Independent channel output |
| TOC.TOCm | 0: Operation mode 1 (= Toggle mode if TAUBnTOM.TOMm = 0) |
| TOL.TOLm | 0: Positive logic |
| TDE.TDEm | 0: Disables dead time operation |
| TDL.TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

**Note** The channel output mode can also be set to Direct Channel Output Mode by setting TAUBnTOE.TOEm = 0. TAUBnTTOUTm can then be controlled independently of the interrupts. For details refer to *Table 14-16 “Channel output modes” on page 490* .

### (d) Simultaneous rewrite

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the Clock Divide Function. Therefore, these registers must be set to 0.

**Table 14-75 Simultaneous rewrite settings for Clock Divide function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDS.RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDM.RDMm | |
| RDC.RDCm | |

### (5) Operating procedure for Clock Divide Function

**Table 14-76 Operating procedure for Clock Divide Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-72 "TAUBnCMORm settings for Clock Divide Function" on page 581* and *Table 14-73 "TAUBnCMURm settings for Clock Divide Function" on page 581*<br><br>Set the value of the TAUBnCDRm register<br><br>Set the channel output mode by setting the control bits as described in *Table 14-74 "Control bit settings for Independent Channel Output Mode 1" on page 582* | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>TAUBnCNTm loads the TAUBnCDRm value. When TAUBnCMORm.MD0 is set to 1, INTTAUBnIm is generated and TAUBnTTOUTm toggles. |
| During operation | The value of TAUBnCDRm can be changed at any time.<br>The TAUBnCNTm register can be read at all times. | When a TAUBnTTINm input edge is detected, TAUBnCNTm counts down. When the counter reaches $0000_H$:<br>• TAUBnCNTm reloads the TAUBnCDRm value and continues count operation<br>• INTTAUBnIm is generated<br>• TAUBnTTOUTm toggles.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and both it and TAUBnTTOUTm retain their current values.<br><br>When TAUBnTOE.TOEm is 0, TAUBnTTOUTm output is initialized to the value set by TAUBnTO.TOm. |

Restart

### (6) Specific timing diagrams

#### (a) TAUBnCDRm = $0000_H$

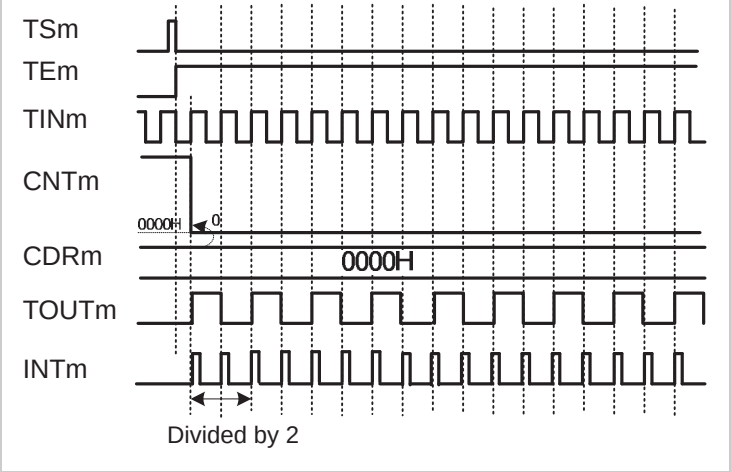



**Figure 14-62 TAUBnCDRm = $0000_H$, TAUBnCMORm.MD0 = 1, TAUBnCMURm.TIS[1:0] = 01**

- If TAUBnCDRm is $0000_H$, TAUBnCNTm is also always $0000_H$.
- INTTAUBnIm is generated every count clock, resulting in TAUBnTTOUTm toggling every count clock.

#### (b) Restart

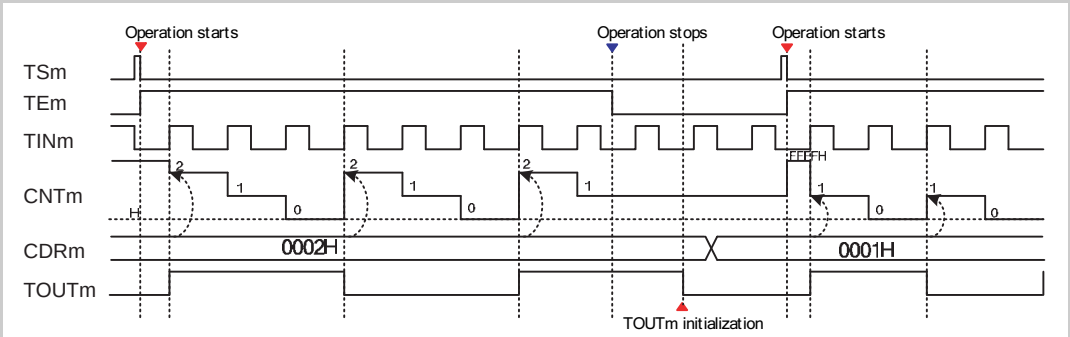



**Figure 14-63 Restart, TAUBnCMORm.MD0 = 1, TAUBnCMURm.TIS[1:0] = 01**

To reset the value of TAUBnTTOUTm:

- Set TAUBnTOE.TOEm = 0 when the counter is stopped (TAUBnTE.TEm = 0)
- Then write either 0 or 1 to TAUBnTO.TOm to set the new start value of TAUBnTTOUTm

**(c) Forced restart**

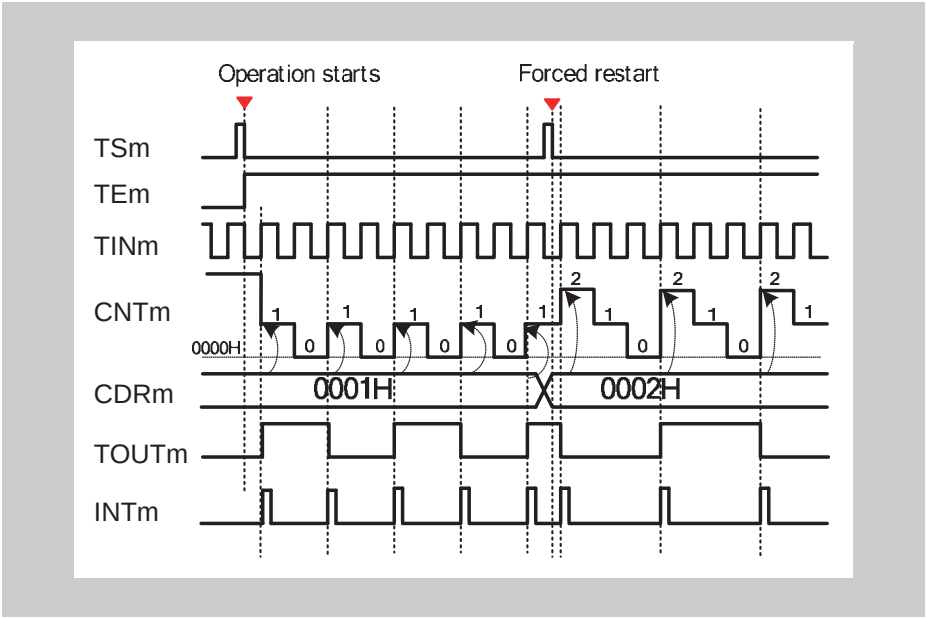



**Figure 14-64 Forced restart, TAUBnCMORm.MD0 = 1, TAUBnCMURm.TIS[1:0] = 01**

- The counter can be forcibly restarted (without stopping it first) by setting TAUBnTS.TSm = 1 during operation.
- The value of TAUBnCDRm is written to TAUBnCNTm and the count operation restarts.
- TAUBnTTOUTm restarts at the same level as before the forced restart.

### 14.17.3 TAUBnTTINm Input Position Detection Function

#### (1) Overview

**Summary** This function measures the duration between the function start and a TAUBnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Count Capture Mode, refer to *Table 14-77 "TAUBnCMORm settings for TAUBnTTINm Input Position Detection Function" on page 588*
- TAUBnTTOUTm is not used for this function

**Description** The counter is enabled by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. The counter starts to count from $0000_H$. When a valid TAUBnTTINm input stop edge is detected, the current TAUBnCNTm value is written to TAUBnCDRm and an interrupt (INTTAUBnIm) is generated. The counter continues to count from the current value until the next valid TAUBnTTINm input edge is detected.

When the counter reaches $FFFF_H$, the bit TAUBnCSRm.OVF is set to 1 and the counter restarts from $0000_H$. The value of TAUBnCSRm.OVF is reset by the CPU by TAUBnCSCm.CLOV = 1.

**Note** The input TAUBnTTINm is sampled at the frequency of the operation clock, specified by the TAUBnCMORm.CKS[1:0] bits. As a result, the output cycle of TAUBnTTOUTm has an error of ± 1 operation clock cycle.

**Conditions** If the TAUBnCMORm.MD0 bit is set to 0, the first interrupt after a start or restart is not generated. For details refer to *14.11 "TAUBnTTOUTm toggle and INTTAUBnIm Generation when Counter start is triggered (MD0-bit)" on page 498* .

#### (2) Equations

Function duration at a TAUBnTTINm input pulse =

count clock cycle × [($FFFF_H$+1 × TAUBnCSRm.OVF) + (TAUBnCDRm capture value + 1)]

### (3) Block diagram and general timing diagram

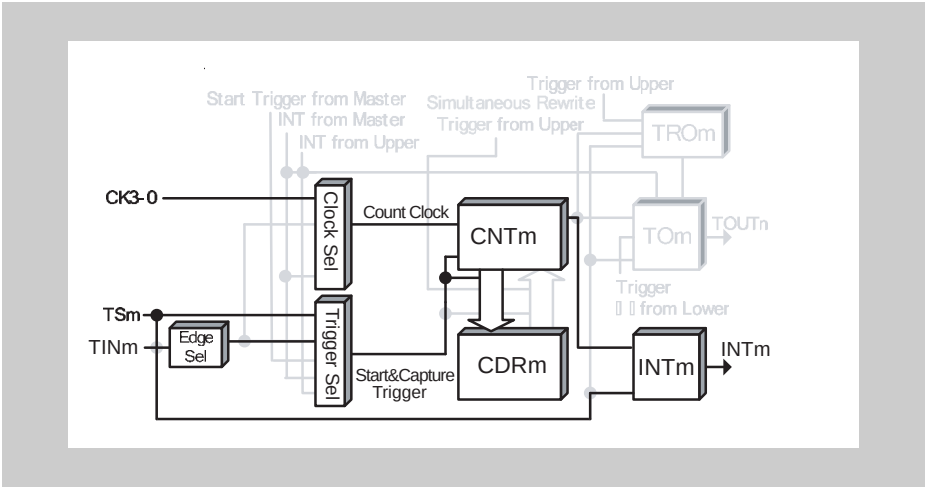



**Figure 14-65 Block diagram for TAUBnTTINm Input Position Detection Function**

The following settings apply to the general timing diagram:

- INTTAUBnIm not generated at operation start (TAUBnCMORm.MD0 = 0)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

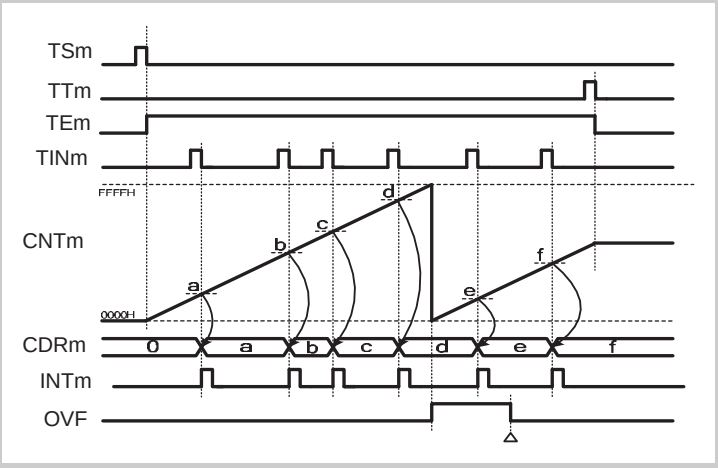



**Figure 14-66 General timing diagram for TAUBnTTINm Input Position Detection Function**

### (4) Register settings

#### (a) TAUBnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-77 TAUBnCMORm settings for TAUBnTTINm Input Position Detection Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid TAUBnTTINm input edge signal is used as the external capture trigger |
| COS[1:0] | 01: Overflow (TAUBnCSRm.OVF) set upon counter overflow and cleared by a CPU instruction |
| MD[4:1] | 1011: Count Capture Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start<br>1: Generates INTTAUBnIm at operation start |

#### (b) TAUBnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-78 TAUBnCMURm settings for TAUBnTTINm Input Position Detection Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUBnRDE, TAUBnRDS, TAUBnRDM, and TAUBnRDC) cannot be used with the TAUBnTTINm Input Position Detection Function. Therefore, these registers must be set to 0.

**Table 14-79 Simultaneous rewrite settings for TAUBnTTINm Input Position Detection Function**

| Bit name | Setting |
|---|---|
| RDEm | 0: Disables simultaneous rewrite |
| RDSm | 0: When simultaneous rewrite is disabled (TAUBnRDE.RDEm = 0), set these bits to 0 |
| RDMm | |
| RDCm | |

### (5) Operating procedure for TAUBnTTINm Input Position Detection Function

**Table 14-80 Operating procedure for TAUBnTTINm Input Position Detection Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Set the TAUBnCMORm register and TAUBnCMURm registers as described in *Table 14-77 "TAUBnCMORm settings for TAUBnTTINm Input Position Detection Function" on page 588* and *Table 14-78 "TAUBnCMURm settings for TAUBnTTINm Input Position Detection Function" on page 588*<br><br>Set the value of the TAUBnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm to 1.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is set to 1 and the counter starts.<br>INTTAUBnIm is generated when TAUBnCMORm.MD0 is set to 1. |
| During operation | The TAUBnCMURm.TIS[1:0] bits can be changed at any time.<br>The TAUBnCDRm and TAUBnCSRm registers can be read at any time. | TAUBnCNTm starts to count up from $0000_H$.<br>When a TAUBnTTINm valid edge is detected:<br>• TAUBnCNTm transfers (captures) its value to TAUBnCDRm<br>• TAUBnTTINm is output.<br>• The counter value is not cleared to $0000_H$ and TAUBnCNTm continues count operation.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUBnTT.TTm to 1.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm stops and both it and TAUBnCSRm.OVF retain their current values. |

Restart →

### (6) Specific timing diagrams

#### (a) Operation stop and restart

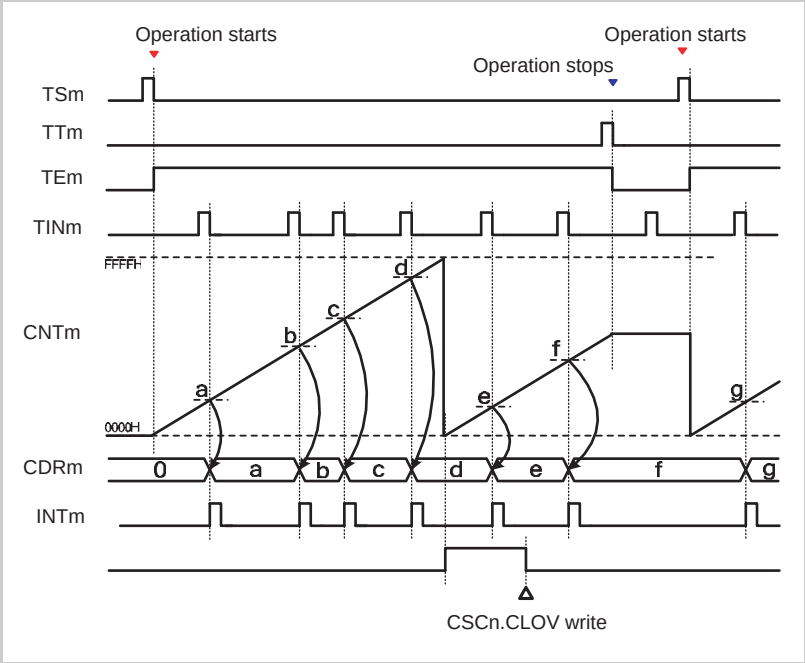



**Figure 14-67 Operation stop and restart, TAUBnCMORm.MD0 = 0, TAUBnCMURm.TIS[1:0] = 00**

- The counter can be stopped by setting TAUBnTT.TTm to 1, which in turn sets TAUBnTE.TEm to 0.
- TAUBnCNTm stops and the current value is retained.
- If the counter is stopped, valid TAUBnTTINm input edges are ignored.
- The counter can be restarted by setting TAUBnTS.TSm to 1. TAUBnCNTm restarts to count from $0000_H$.

## 14.18 Synchronous Channel Operation Functions

This section lists all the synchronous channel operation functions provided by the Timer Array Unit B. For a general overview of synchronous channel operation, see *14.4 “Functional Description” on page 472* .

## 14.19 Synchronous PWM Signal Functions Triggered at Regular Intervals

This chapter describes functions that generate PWM signals at regular intervals, that can be reset by a TAUBnTTINm input, with and without dead time.

- *14.19.1 “PWM Output Function”*
- *14.19.2 “Delay Pulse Output Function”*
- *14.19.3 “AD Conversion Trigger Output Function Type 1”*

### 14.19.1 PWM Output Function

#### (1) Overview

**Summary** This function generates multiple PWM outputs by using a master and multiple slave channels. It enables the pulse cycle (frequency) and the pulse width (duration) of the TAUBnTTOUTm to be set. The pulse cycle is set in the master channel. The pulse width is set in the slave channel.

**Prerequisites**

- Two channels
- The operation mode of the master channel must be set to Interval Timer Mode, refer to *Table 14-81 “TAUBnCMORm settings for the master channel of the PWM Output Function” on page 596*
- The operation mode of the slave channel(s) must be set to One Count Mode, refer to *Table 14-84 “TAUBnCMORm settings for the slave channel of the PWM Output Function” on page 598*
- TAUBnTTOUTm is not used for the master channel of this function
- The channel output mode of the slave channel(s) must be set to Synchronous Channel Output Mode 1 ( *14.9 “Channel Output Modes” on page 489* )

**Description** The counters are started by setting the channel trigger bits (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm = 1, enabling count operation. The current value of TAUBnCDRm is written to TAUBnCNTm and the counters start to count down from these values. INTTAUBnIm is generated on the master channel and TAUBnTTOUTm (slave) toggles.

- Master channel:

  When the counter of the master channel reaches $0000_H$, pulse cycle time has elapsed and INTTAUBnIm is generated. The counter reloads the TAUBnCDRm value and counts down.

- Slave channel(s)

  The INTTAUBnIm of the master channel triggers the counter of the slave channel(s). The current value of TAUBnCDRm (slave) is written to TAUBnCNTm (slave) and the counter starts to count down from this value. The TAUBnTTOUTm signal is set.

  When the counter reaches $0000_H$, i.e. duty time has elapsed, INTTAUBnIm is generated and the TAUBnTTOUTm signal is reset. The counter returns to $FFFF_H$ and awaits the next INTTAUBnIm of the master channel, and thus the start of the next pulse cycle.

The counter can be stopped by setting TAUBnTT.TTm to 1 for the master and slave channel(s), which in turn sets TAUBnTE.TEm to 0. TAUBnCNTm and TAUBnTTOUTm of master and slave channel(s) stop but retain their values. The counters can be restarted by setting TAUBnTS.TSm to 1.

**Note** If a forced restart is executed during operation, the counter value becomes invalid and TAUBnTTOUTm is not output as expected PWM signal.

**Conditions** Simultaneous rewrite can be used with this function. Please refer to *14.8 “Simultaneous Rewrite” on page 479*

### (2) Equations

Pulse cycle = (TAUBnCDRm (master) + 1) x count clock cycle

Duty cycle [%] = (TAUBnCDRm (slave) / (TAUBnCDRm (master) + 1)) × 100

– Duty cycle = 0 %

  TAUBnCDRm (slave) = $0000_H$

– Duty cycle = 100 %

  TAUBnCDRm (slave) ≥ TAUBnCDRm (master) + 1

### (3) Block diagram and general timing diagram

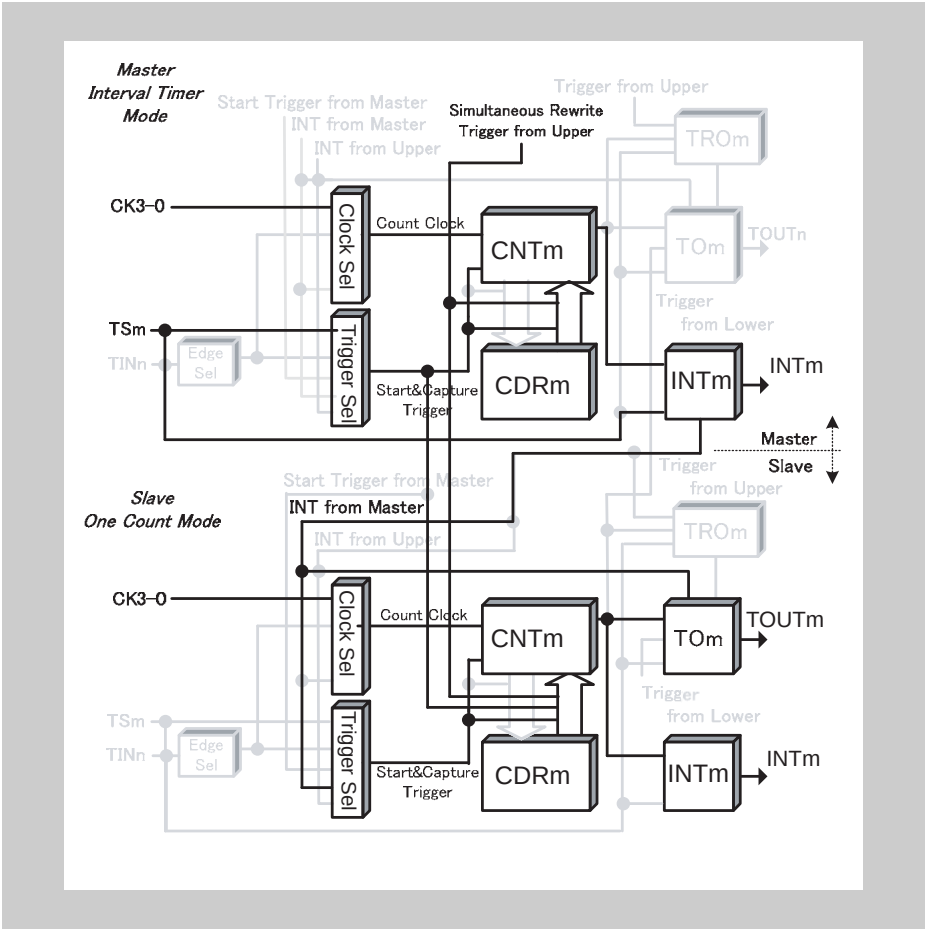



**Figure 14-68 Block diagram for PWM Output Function**

The following settings apply to the general timing diagram:

- Slave channel: Positive logic (TAUBnTOL.TOLm = 0)

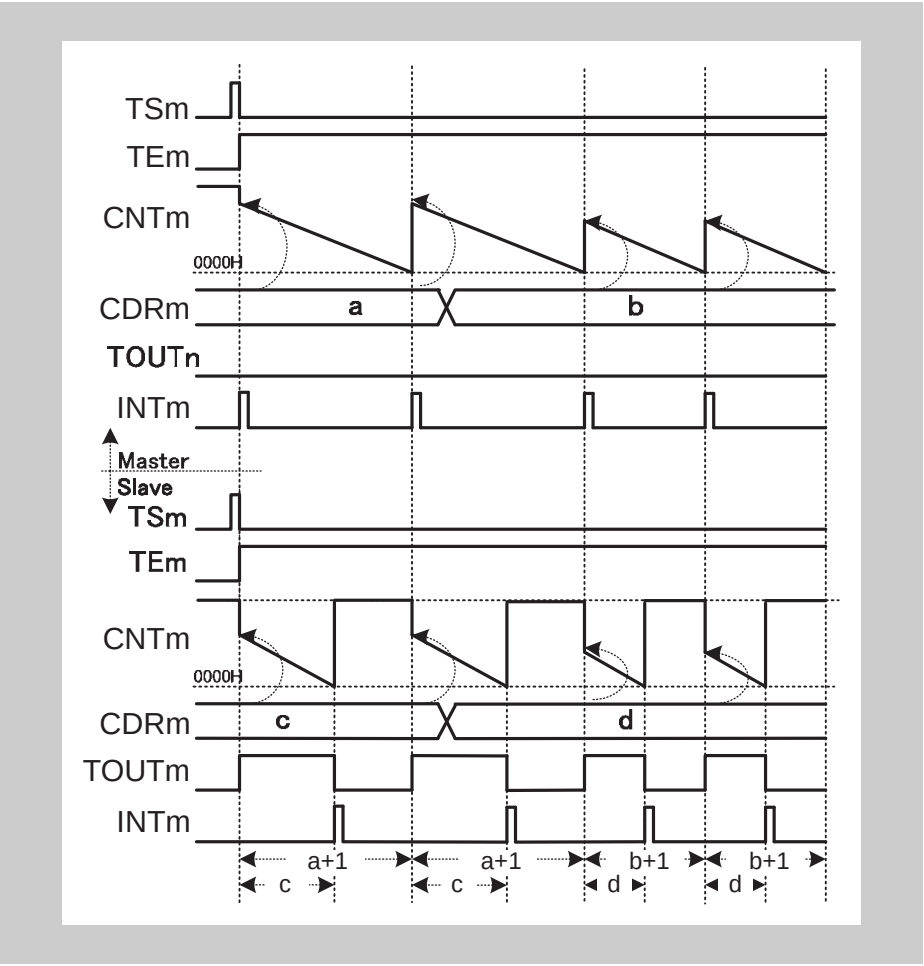



**Figure 14-69 General timing diagram for PWM Output Function**

**Note** The interval between the slave channel starting to count and an interrupt being generated is the value of corresponding TAUBnCDRm, whereas for the master channel the interval is the corresponding TAUBnCDRm + 1.

### (4) Register settings for the master channel

#### (a) TAUBnCMORm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-81 TAUBnCMORm settings for the master channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 1: Channel is master channel |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 1: Generates INTTAUBnIm at operation start |

#### (b) TAUBnCMURm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-82 TAUBnCMURm settings for the master channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

**(c) Channel output mode for the master channel**

The channel output mode is not used by this function. However, it can be used by other functions or in Direct Channel Output Mode.

**(d) Simultaneous rewrite for the master channel**

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 14-83 Simultaneous rewrite settings for the master channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is monitored for the simultaneous rewrite trigger<br>1: An upper channel outside the channel group is monitored for the simultaneous rewrite trigger |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (5) Register settings for the slave channel(s)

#### (a) TAUBnCMORm for the slave channel(s)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-84 TAUBnCMORm settings for the slave channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 100: INTTAUBnIm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start |

#### (b) TAUBnCMURm for the slave channel(s)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-85 TAUBnCMURm settings for the slave channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for the slave channel(s)

**Table 14-86 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 1: Synchronous channel operation |
| TOCm | 0: Operation mode 1 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

### (d) Simultaneous rewrite for the slave channel(s)

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 14-87 Simultaneous rewrite settings for the slave channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is monitored for the simultaneous rewrite trigger<br>1: An upper channel outside the channel group is monitored for the simultaneous rewrite trigger |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (6) Operating procedure for PWM Output Function

Table 14-88 Operating procedure for PWM Output Function

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Master channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *4 "Register settings for the master channel" on page 596*<br><br>Slave channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *5 "Register settings for the slave channel(s)" on page 598*<br><br>Set the values of the TAUBnCDRm registers of all channels | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm of the master and slave channels to 1 simultaneously.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm (master and slave channels) is set to 1 and the counters of the master and slave channels start.<br>INTTAUBnIm is generated on the master channel and TAUBnTTOUTm (slave) is set. |
| During operation | TAUBnCDRm can be changed at any time. TAUBnCNTm and TAUBnRSF.RSFm can be read at any time.<br><br>TAUBnRDT.RDTm can be changed during operation. | TAUBnCNTm of the master channel loads TAUBnCDRm and counts down. When the counter reaches $0000_H$:<br>• INTTAUBnIm (master) is generated<br>• TAUBnCNTm (master) reloads the TAUBnCDRm value and continues count operation<br>• TAUBnCNTm (slave) reloads the TAUBnCDRm value and counts down<br>• TAUBnTTOUTm (slave) is set<br>When TAUBnCNTm (slave) reaches $0000_H$:<br>• INTTAUBnIm (slave) is generated<br>• TAUBnTTOUTm (slave) is reset |
| Stop operation | Set TAUBnTT.TTm of the master and slave channels to 1 simultaneously.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values.<br><br>When TAUBnTOE.TOEm is 0, TAUBnTTOUTm output is initialized to the value set by TAUBnTO.TOm. |

Restart

### (7) Specific timing diagrams

#### (a) Duty cycle = 0 %

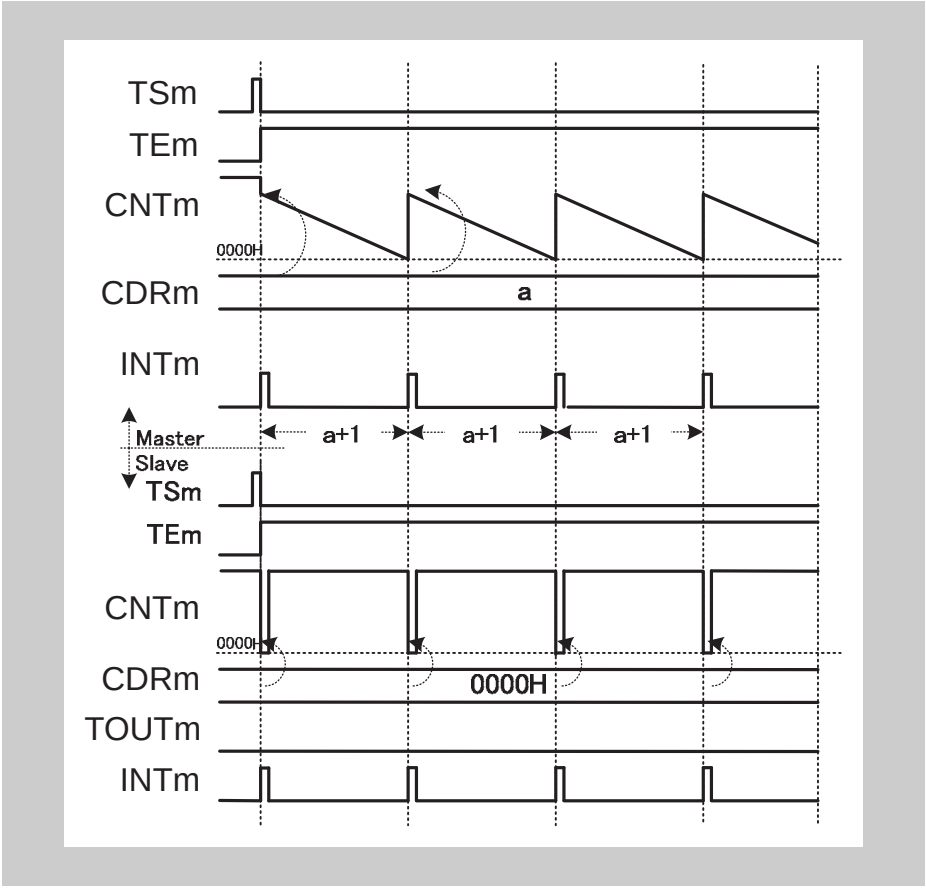



**Figure 14-70 TAUBnCDRm (slave) = $0000_H$, positive logic (TAUBnTOL.TOLm (slave) = 0)**

- Every time the master channel generates an interrupt (INTTAUBnIm), $0000_H$ is written to TAUBnCNTm (slave). Therefore, TAUBnCNTm (slave) cannot start to count and TAUBnTTOUTm remains at not active state.
- TAUBnCNTm (slave) generates an interrupt every time the value of TAUBnCDRm is reloaded. The slave and the master channel generate interrupts in the same cycle.

**(b) Duty cycle = 100 %**

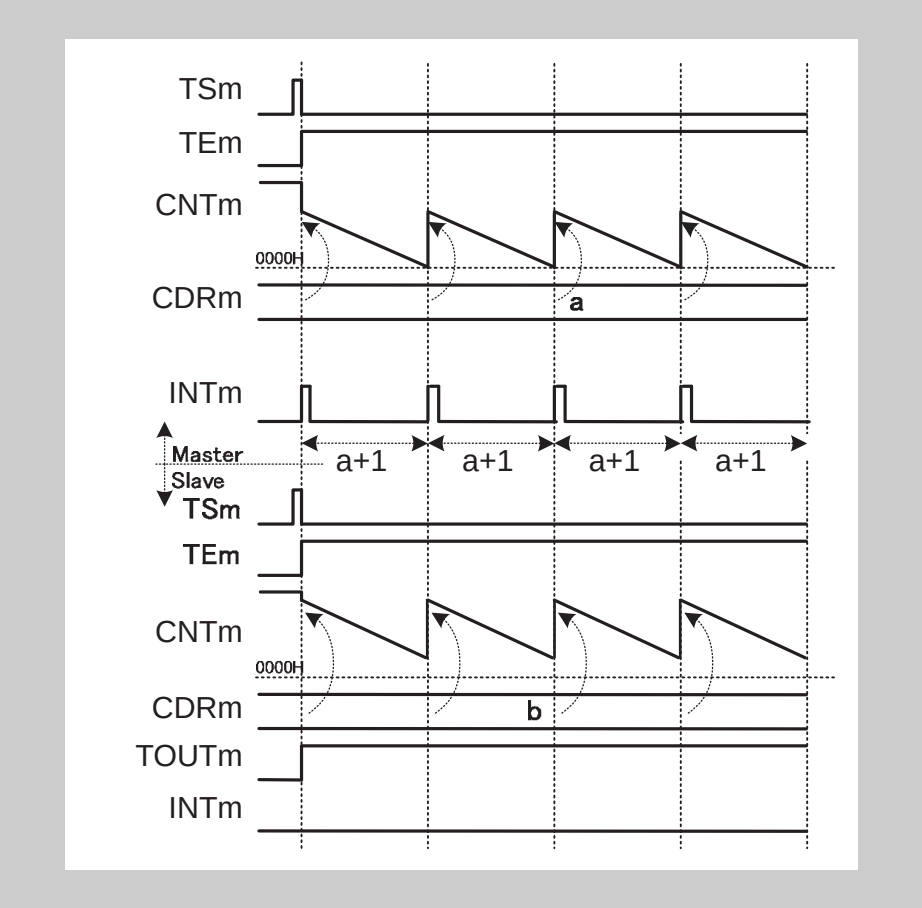



**Figure 14-71 TAUBnCDRm (slave) ≥ TAUBnCDRm (master) + 1, positive logic (TAUBnTOL.TOLm (slave) = 0)**

- If the value TAUBnCDRm (slave) is higher than the value TAUBnCDRm (master), the counter of the slave channel cannot reach $0000_H$ and cannot generate interrupts. The TAUBnTTOUTm remains at active state.

### (c) Stop and restart operation

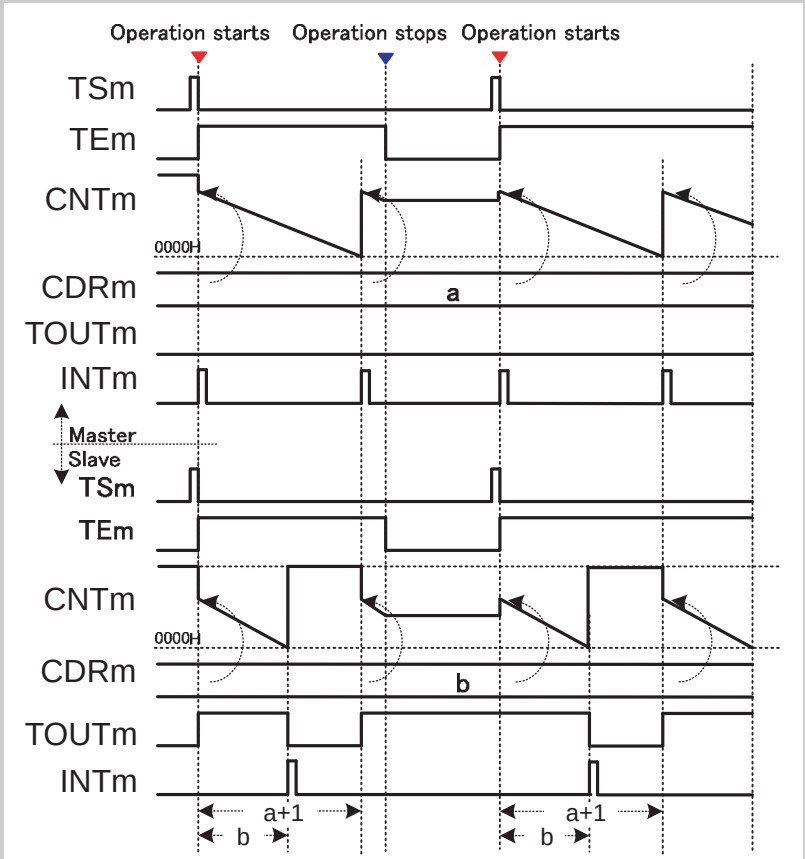



**Figure 14-72 Stop and restart operation, positive logic (TAUBnTOL.TOLm (slave) = 0)**

- The counter can be stopped by setting TAUBnTT.TTm of the master and slave channel(s) to 1, which in turn sets TAUBnTE.TEm to 0.
- TAUBnCNTm and TAUBnTTOUTm of all channels stop and the current values are retained. No interrupts are generated.
- The counter can be restarted by setting TAUBnTS.TSm of master and slave channel(s) to 1. TAUBnCNTm of master and slave channel reload the current values of TAUBnCDRm and start to count down from these values.

## 14.19.2 Delay Pulse Output Function

### (1) Overview

**Summary**

This function outputs two signals. The reference signal has a defined pulse width and pulse cycle specified using the master channel and slave channel 1. Slave channels 2 and 3 output the reference signal with a specified delay. The delay signal is identical to the reference signal, but delayed by amount specified in slave channel 2.

The signal values are specified in the following way:

- The pulse cycle is specified using the master channel.
- The duty cycle of the reference signal is specified using slave channel 1. The duty cycle of the delay signal is specified using slave channel 3.

  The values of TAUBnCDRm of these both channels have to be identical.
- The delay is specified in slave channel 2.

**Prerequisites**

- Four channels
- The operation mode of the master channel must be set to Interval Timer Mode, refer to *Table 14-89 “TAUBnCMORm settings for the master channel of the Delay Pulse Output Function” on page 608*
- The operation mode of slave channel 1 and 2 must be set to One Count Mode, refer to *Table 14-92 “TAUBnCMORm settings for slave channel 1 of the Delay Pulse Output Function” on page 610*
- The operation mode of slave channel 3 must be set to Pulse One Count Mode, refer to *Table 14-96 “TAUBnCMORm settings for slave channel 2 of the Delay Pulse Output Function” on page 612*
- TAUBnTTOUTm is not used for the master channel and slave channel 2
- The channel output mode of slave channel 1 must be set to Synchronous Channel Output Mode 1 (refer to *14.9 “Channel Output Modes” on page 489* )
- The channel output mode of slave channel 3 must be set to Independent Channel Output Mode 1 (refer to *14.9 “Channel Output Modes” on page 489* )

**Description**

The counters of the channel group are started by setting the channel trigger bit (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm, enabling count operation.

- Master channel:

  The current value of TAUBnCDRm is written to TAUBnCNTm and the counter starts to count down from this value. INTTAUBnIm is generated on the master channel.

  When the counter of the master channel reaches $0000_H$, pulse cycle time has elapsed and INTTAUBnIm is generated. The counter reloads the TAUBnCDRm value and counts down.
- Slave channels 1 and 2:

  When the slave channels 1 and 2 detect an interrupt from the master channel, they start to count down from the current value of TAUBnCDRm. The TAUBnTTOUTm signal (slave 1) is set.

– Slave channel 1:

When the counter of slave channel 1 reaches $0000_H$, duty time has elapsed, INTTAUBnIm is generated and the TAUBnTTOUTm signal is reset. The counter returns to $FFFF_H$ and awaits the next INTTAUBnIm of the master channel.

– Slave channel 2:

When the counter of slave channel 2 reaches $0000_H$, delay time has elapsed and INTTAUBnIm is generated. The counter returns to $FFFF_H$ and awaits the next INTTAUBnIm of the master channel.

INTTAUBnIm (slave 2) triggers the counter of slave channel 3

- Slave channel 3:

  When slave channel 3 detects an interrupt from slave channel 2, it starts to count down from the current value of TAUBnCDRm. INTTAUBnIm is generated and the TAUBnTTOUTm signal (slave 3) toggles.

  When the counter of slave channel 3 reaches $0000_H$, duty time has elapsed, INTTAUBnIm is generated and the TAUBnTTOUTm signal toggles again.

  The output from slave channel 3 is the delayed PWM pulse

The counter can be stopped by setting TAUBnTT.TTm to 1 for the master and slave channels, which in turn sets TAUBnTE.TEm to 0. TAUBnCNTm and TAUBnTTOUTm of master and slave channels stop but retain their values. The counters can be restarted by setting TAUBnTS.TSm to 1.

**Conditions** Simultaneous rewrite can be used with this function. Please refer to *14.8 “Simultaneous Rewrite” on page 479*

**Equations** Pulse cycle = (TAUBnCDRm (master) + 1) × count clock cycle

Duty cycle = (TAUBnCDRm (slave 1 and 3)) × count clock cycle

Delay = (TAUBnCDRm (slave 2) + 1) × count clock cycle

### (2) Block diagram and general timing diagram

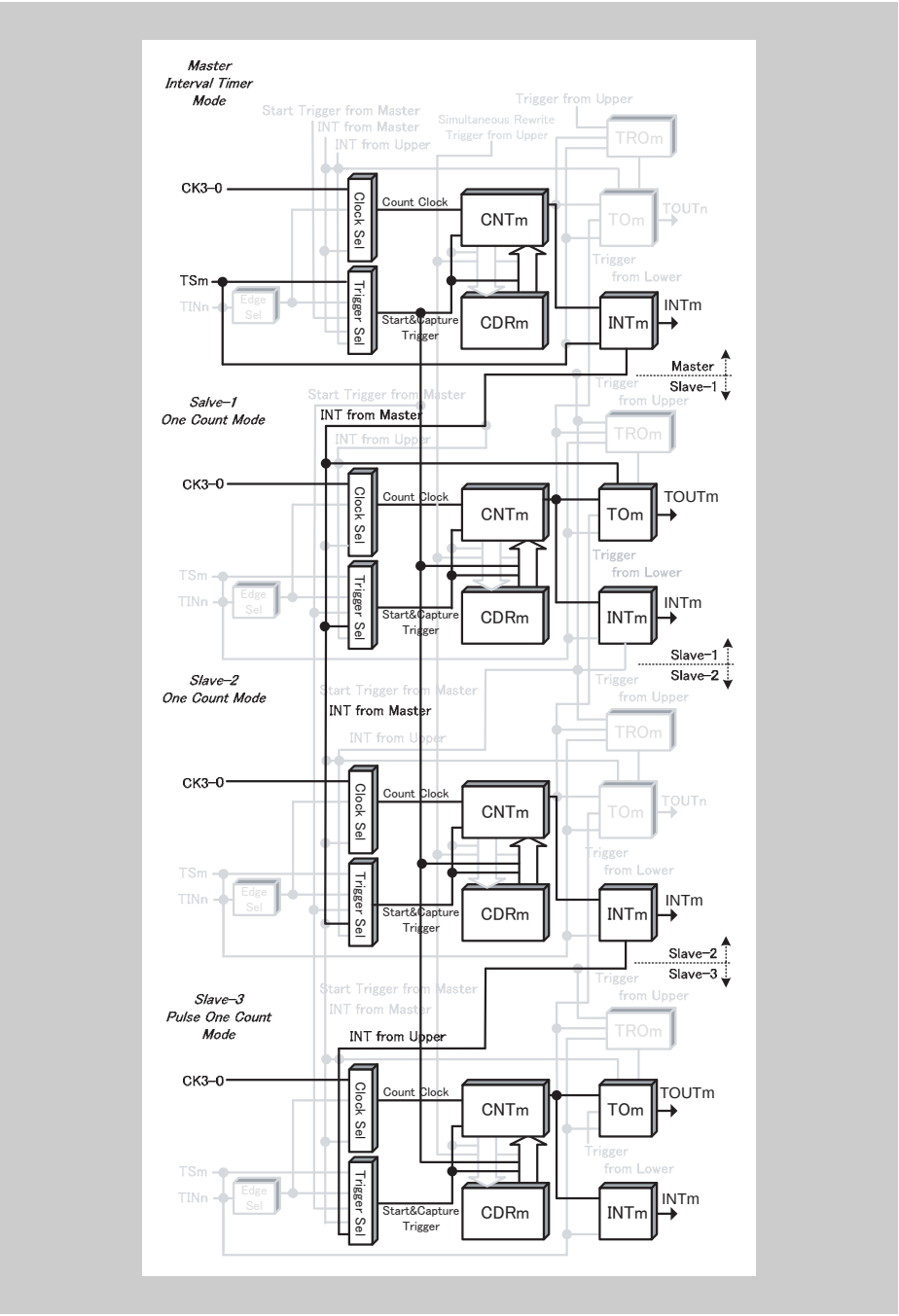



Figure 14-73 Block diagram for Delay Pulse Output Function

The following settings apply to the general timing diagram:

- All channels
  - INTTAUBnIm is generated at operation start (TAUBnCMORm.MD0 = 1)

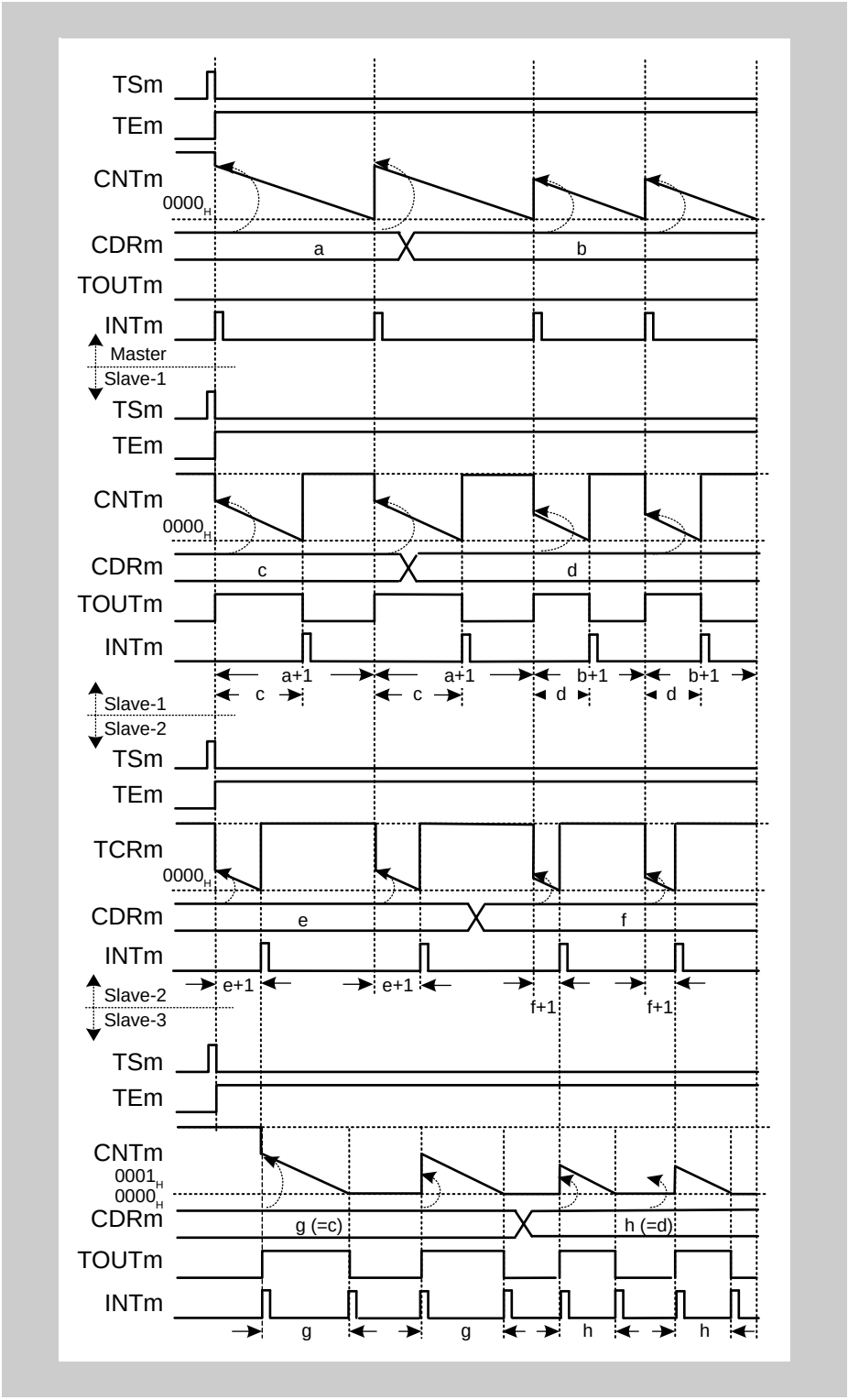



**Figure 14-74 General timing diagram for Delay Pulse Output Function**

### (3) Register settings for the master channel

#### (a) TAUBnCMORm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-89 TAUBnCMORm settings for the master channel of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 1: Channel is master channel |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 1: Generates INTTAUBnIm at operation start |

#### (b) TAUBnCMURm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-90 TAUBnCMURm settings for the master channel of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

**(c) Channel output mode for the master channel**

The channel output mode is not used by the master channel of this function. However, it can be used by other functions or in Direct Channel Output Mode.

**(d) Simultaneous rewrite for the master channel**

The simultaneous rewrite settings of the master and slave channels must be identical.

**Table 14-91 Simultaneous rewrite settings for the master channel of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (4) Register settings for slave channel 1

#### (a) TAUBnCMORm for slave channel 1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-92 TAUBnCMORm settings for slave channel 1 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 100: TAUBnTTOUTm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start (and enables start trigger during counting). |

#### (b) TAUBnCMURm for slave channel 1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-93 TAUBnCMURm settings for slave channel 1 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for slave channel 1

**Table 14-94 Control bit settings for slave channel 1 of the Synchronous Channel Output Mode 2**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 1: Synchronous channel operation |
| TOCm | 0: Operation mode 1 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

### (d) Simultaneous rewrite for slave channel 1

The simultaneous rewrite settings of the master and slave channels must be identical.

**Table 14-95 Simultaneous rewrite settings for slave channel 1 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (5) Register settings for slave channel 2

#### (a) TAUBnCMORm for slave channel 2

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-96 TAUBnCMORm settings for slave channel 2 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 100: TAUBnTTOUTm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 1: Generates INTTAUBnIm at operation start (and enables start trigger during counting) |

#### (b) TAUBnCMURm for slave channel 2

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-97 TAUBnCMURm settings for slave channel 2 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

**(c) Channel output mode for slave channel 2**

The channel output mode is not used by this function. However, it can be used by other functions or in Direct Channel Output Mode.

**(d) Simultaneous rewrite for slave channel 2**

The simultaneous rewrite settings of the master and slave channels must be identical.

**Table 14-98 Simultaneous rewrite settings for slave channel 2 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (6) Register settings for slave channel 3

#### (a) TAUBnCMORm for slave channel 3

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-99 TAUBnCMORm settings for slave channel 3 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 101: INTTAUBnIm of the upper channel (m-1) is the start trigger, regardless of the master setting |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 1010: Pulse One Count Mode |
| MD0 | 1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start (and enables start trigger during counting) |

#### (b) TAUBnCMURm for slave channel 3

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-100 TAUBnCMURm settings for slave channel 3 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for slave channel 3

**Table 14-101 Control bit settings for Synchronous Channel Output Mode 2**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 0: Independent channel output |
| TOCm | 0: Operation mode 1 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

### (d) Simultaneous rewrite for slave channel 3

The simultaneous rewrite settings of the master and slave channels must be identical.

**Table 14-102 Simultaneous rewrite settings for slave channel 3 of the Delay Pulse Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (7) Operating procedure for Delay Pulse Output Function

**Table 14-103 Operating procedure for Delay Pulse Output Function (1/2)**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Master channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *3 "Register settings for the master channel" on page 608*<br><br>Slave channel 1: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *4 "Register settings for slave channel 1" on page 610*<br><br>Slave channel 2: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *5 "Register settings for slave channel 2" on page 612*<br><br>Slave channel 3: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *6 "Register settings for slave channel 3" on page 614*<br><br>Set the values of the TAUBnCDRm registers of all channels | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |

**Table 14-103 Operating procedure for Delay Pulse Output Function (2/2)**

| | Operation | Status of TAUBn |
|---|---|---|
| Start operation | Set TAUBnTS.TSm of the master and slave channels to 1 simultaneously.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm (master and slave channels) is set to 1 and the counters of the master channel and slave channels 1 and 2 start.<br>INTTAUBnIm is generated on the master channel and TAUBnTTOUTm (slave 1) is set. |
| During operation | TAUBnCDRm can be changed at any time.<br>TAUBnCNTm and TAUBnRSF.RSFm can be read at any time.<br><br>TAUBnRDT.RDTm can be changed during operation. | TAUBnCNTm of the master channel and slave channels 1 and 2 load TAUBnCDRm and count down.<br><br>When the counter of the master channel reaches $0000_H$:<br>• INTTAUBnIm (master) is generated<br>• TAUBnCNTm (master) reloads the TAUBnCDRm value and continues count operation<br>• TAUBnCNTm (slave 1 and slave 2) reload the TAUBnCDRm value and start counting down<br>• TAUBnTTOUTm (slave 1) is set<br><br>When TAUBnCNTm (slave 1) reaches $0000_H$:<br>• INTTAUBnIm (slave 1) is generated<br>The output from slave channel 1 acts as the reference pulse<br>• TAUBnTTOUTm (slave 1) is reset<br><br>When TAUBnCNTm (slave 2) reaches $0000_H$:<br>• INTTAUBnIm (slave 2) is generated<br>• TAUBnTTOUTm (slave 3) toggles<br>• TAUBnCNTm (slave 3) reloads the TAUBnCDRm value and starts counting down<br><br>When TAUBnCNTm (slave 3) reaches $0000_H$:<br>• INTTAUBnIm (slave 3) is generated<br>• TAUBnTTOUTm (slave 3) toggles<br>The output from slave channel 3 is the delayed PWM pulse |
| Stop operation | Set TAUBnTT.TTm of the master and slave channels to 1 simultaneously.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values.<br><br>When TAUBnTOE.TOEm is 0, TAUBnTTOUTm output is initialized to the value set by TAUBnTO.TOm. |

Restart

**(8) Specific timing diagrams**

**(a) Duty cycle (slave 3) = 100 %**

The following values apply to the figure below:

- TAUBnCDRm (master) = $000A_H$
- TAUBnCDRm (slave 1) = $000B_H$
- TAUBnCDRm (slave 2) = $0000_H$
- TAUBnCDRm (slave 3) = $000B_H$

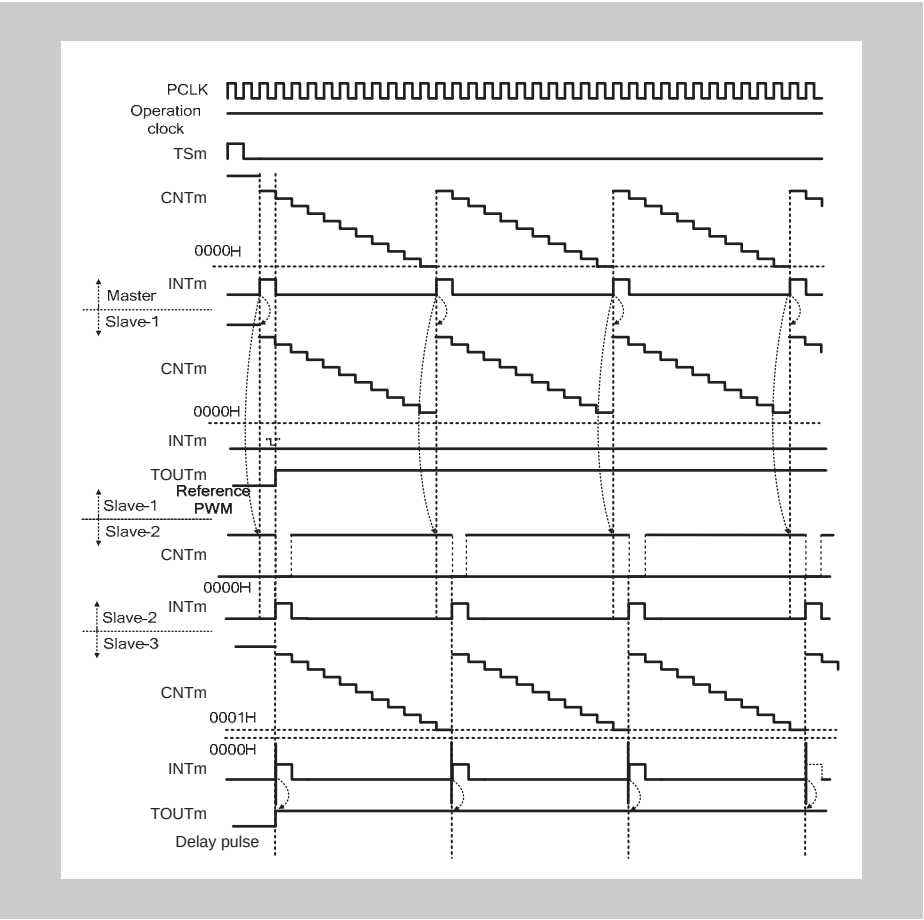



**Figure 14-75 Duty cycle (slave 3) = 100 %**

- If the value of TAUBnCDRm (slave 1 and 3) is higher than the value of TAUBnCDRm (master), the counter of the slave channels cannot reach $0000_H$ and cannot generate interrupts. TAUBnTTOUTm of channels 1 and 3 remain in the active state.

**(b) TAUBnTTOUTm (slave 1) = TAUBnTTOUTm (slave 3)**

The following values apply to the figure below:

- TAUBnCDRm (master) = $000A_H$
- TAUBnCDRm (slave 1) = $0005_H$
- TAUBnCDRm (slave 2) = $0000_H$
- TAUBnCDRm (slave 3) = $0005_H$

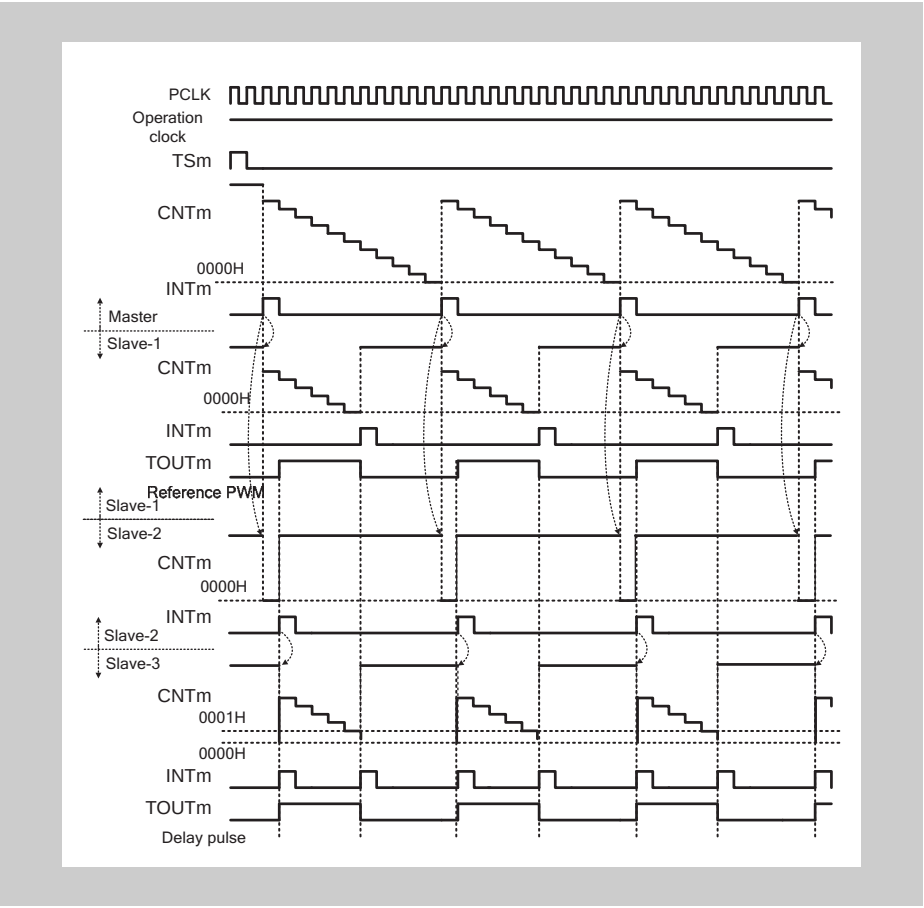



**Figure 14-76 TAUBnTTOUTm (slave 1) = TAUBnTTOUTm (slave 3)**

- If TAUBnCDRm (slave 2) = $0000_H$, the counter of slave channel 3 starts counting one count clock later than the counter of slave channel 1. The reference pulse and the delay pulse are output with a delay of one clock count.

## 14.19.3 AD Conversion Trigger Output Function Type 1

### (1) Overview

**Summary** This function is identical to *14.19.1 “PWM Output Function” on page 593* except that TAUBnTTOUTm is not output.

This is achieved by setting the channel output mode of the slave to Direct Channel Output Mode.

### (2) Block diagram and general timing diagram

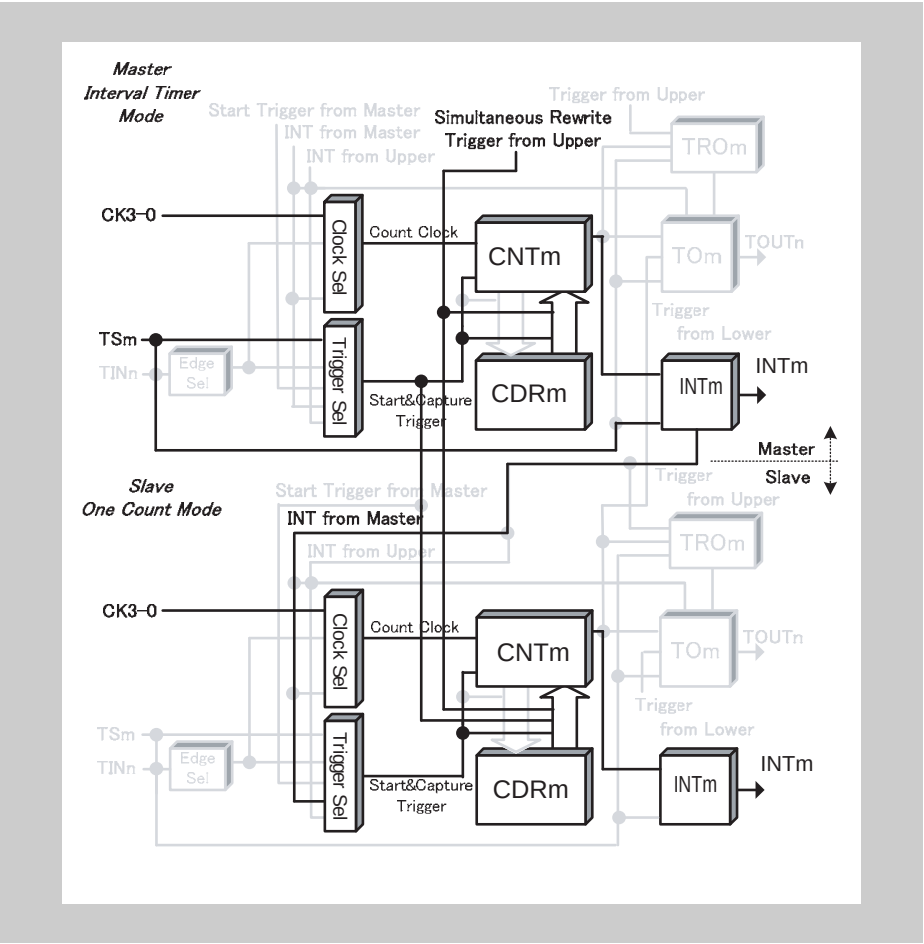



**Figure 14-77 Block diagram for AD Conversion Trigger Output Function Type 1**

### (3) General timing diagram

The following settings apply to the general timing diagram:

- Slave channel: Positive logic (TAUBnTOL.TOLm = 0)

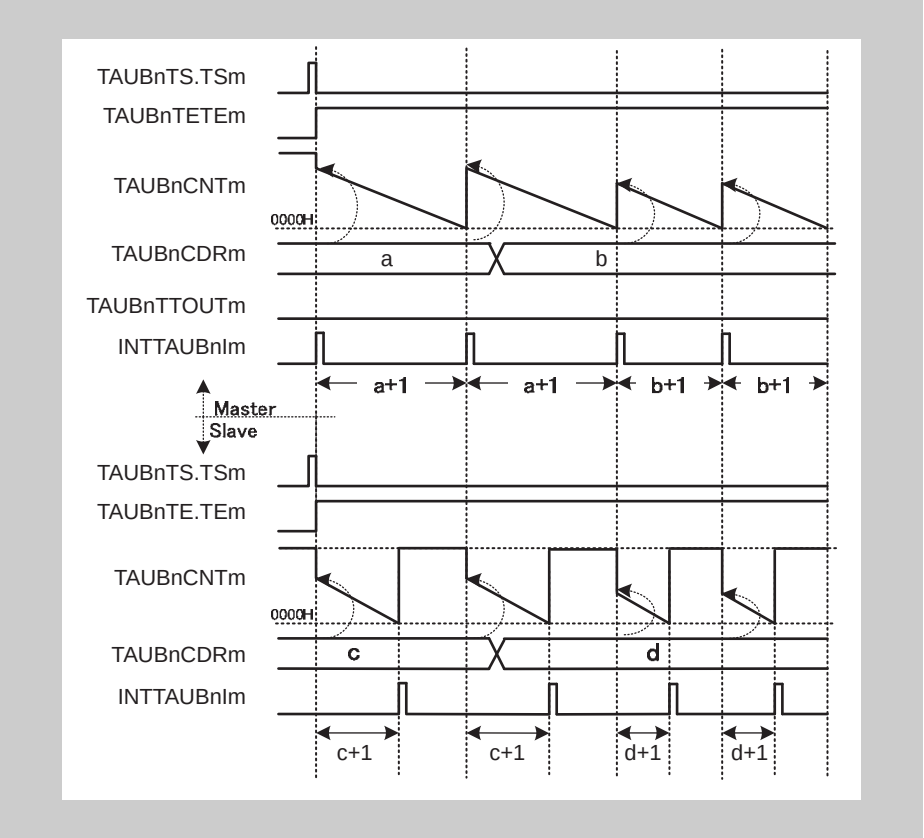



**Figure 14-78 General timing diagram for AD Conversion Trigger Output Function Type 1**

## 14.20 Synchronous PWM Signal Functions Triggered by an External Signal

This chapter describes functions that generate PWM signals and which are triggered by an external signal.

- *14.20.1 “One-Shot Pulse Output Function”*

### 14.20.1 One-Shot Pulse Output Function

#### (1) Overview

**Summary** This function outputs a signal pulse with a defined pulse width and a specific delay time compared to an external input signal pulse by using a master and a slave channel. The delay time is specified using the master channel. The pulse width is specified using the slave channel.

**Prerequisites**

- Two channels
- The operation mode of the master channel must be set to One Count Mode, refer to *Table 14-104 “TAUBnCMORm settings for the master channel of the One-Shot Pulse Output Function” on page 626*
- The operation mode of the slave channel must be set to Pulse One Count Mode, refer to *Table 14-107 “TAUBnCMORm settings for the slave channel of the One-Shot Pulse Output Function” on page 628*
- TAUBnTTOUTm is not used for the master channel of this function
- The channel output mode of the slave channel must be set to Synchronous Channel Output Mode 2 (refer to *14.9 “Channel Output Modes” on page 489* )
- TAUBnTTINm (master) has to be detected while TAUBnCNTm (master) and TAUBnCNTm (slave) await a trigger. Furthermore, the slave is only triggered by an interrupt from the master channel and not by TAUBnTTINm.

**Description** The counters are enabled by setting the channel trigger bits (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm, enabling count operation.

- Master channel:

  When the next valid TAUBnTTINm input edge is detected, the current value of TAUBnCDRm is written to TAUBnCNTm. The counter starts to count down from this value. If TAUBnCMORm.MD0 = 0, a trigger (TAUBnTTINm) which is detected within the delay time is ignored.
  When the counter of the master channel reaches $0000_H$, INTTAUBnIm is generated. The counter returns to $FFFF_H$ and awaits the next valid TAUBnTTINm input edge.

- Slave channel

  The INTTAUBnIm of the master channel triggers the counter of the slave channel. The current value of TAUBnCDRm (slave) is written to TAUBnCNTm (slave) and the counter starts to count down from this value. An interrupt is generated and the TAUBnTTOUTm signal is set.
  When the counter reaches $0001_H$, INTTAUBnIm is generated and the TAUBnTTOUTm signal is reset. The counter remains at $0001_H$ and awaits the next INTTAUBnIm of the master channel.

The counter can be stopped by setting TAUBnTT.TTm to 1 for the master and slave channel, which in turn sets TAUBnTE.TEm to 0. TAUBnCNTm and TAUBnTTOUTm of master and slave channel stop but retain their values. The counters can be restarted by setting TAUBnTS.TSm to 1.

The counter of the master channel can be restarted without stopping it first (forced restart) by setting TAUBnTS.TSm to 1 during operation.

**Notes**

1. If a forced restart of the slave channel is executed during operation, the width of the output signal does not correspond to the value of TAUBnCDRm (slave).
2. The input TAUBnTTINm is sampled at the frequency of the operating clock, specified by TAUBnCMORm.CKS[1:0] bits. As a result, the output cycle of TAUBnTTOUTm has an error of ± 1 operation clock cycle.

**Conditions**

- If TAUBnCMORn.MD0 of the master channel is set to 0, during counting detected TAUBnTTINm input edges are ignored.
- Simultaneous rewrite can be used with this function. Please refer to *14.8 "Simultaneous Rewrite" on page 479*

**Equations**

Delay to input pulse = (TAUBnCDRm (master) + 1) × count clock cycle

Pulse width = (TAUBnCDRm (slave)) × count clock cycle

### (2) Block diagram and general timing diagram

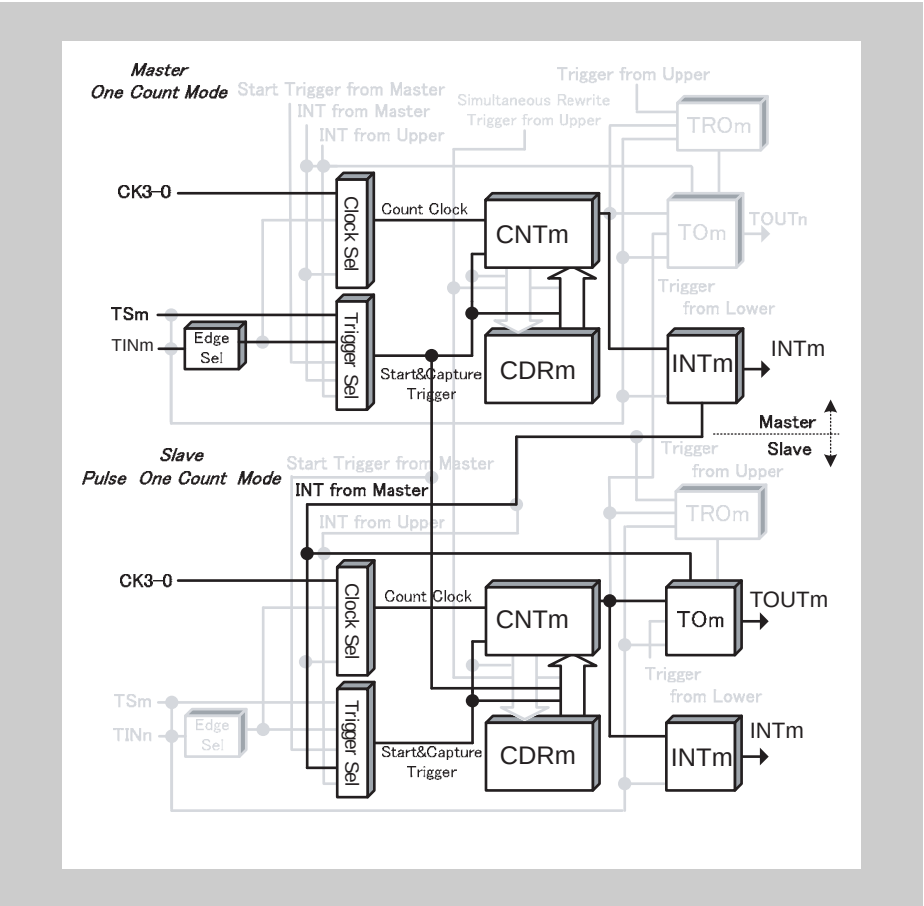



**Figure 14-79 Block diagram for One-Shot Pulse Output Function**

The following settings apply to the general timing diagram:

- Start trigger detection disabled during counting (TAUBnCMORm.MD0 = 0)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

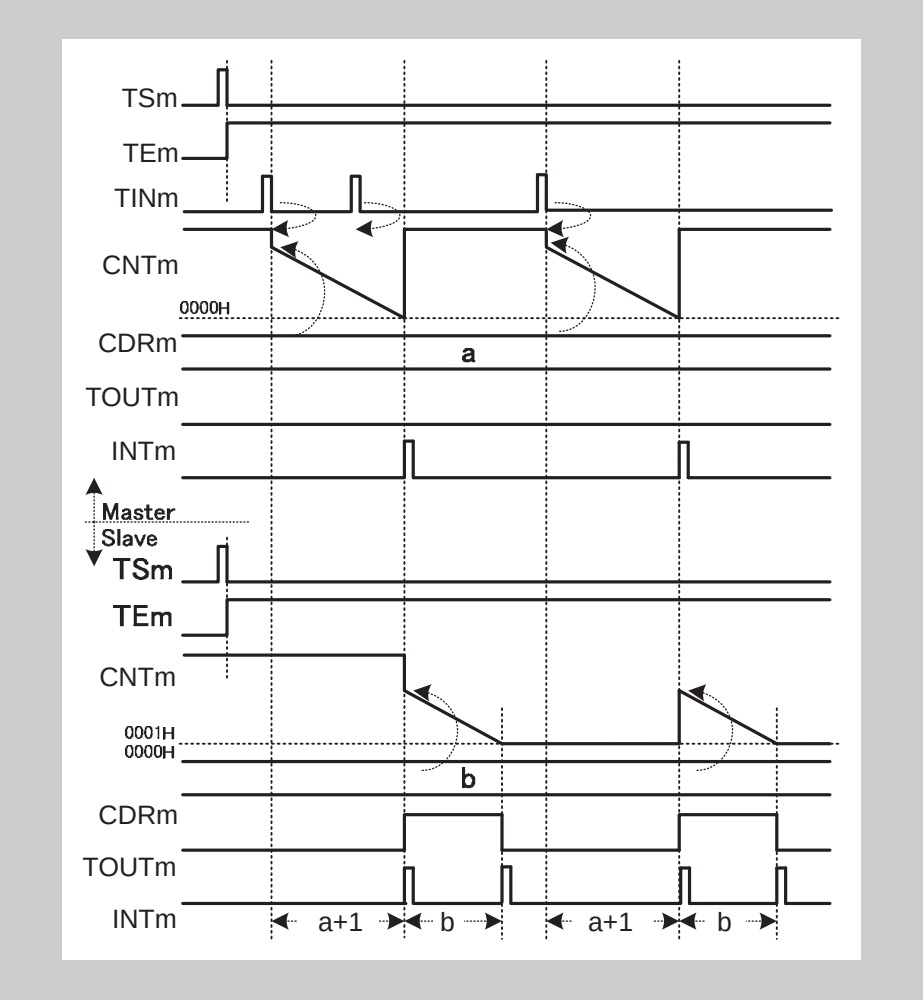



**Figure 14-80 General timing diagram for One-Shot Pulse Output Function**

### (3) Register settings for the master channel

#### (a) TAUBnCMORm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-104 TAUBnCMORm settings for the master channel of the One-Shot Pulse Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 1: Channel is master channel |
| STS[2:0] | 001: Valid TAUBnTTINm input edge signal is used as the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 0: Disables start trigger detection during counting<br>1: Enables start trigger detection during counting<br>The value of the MD0 bit of the master and slave channel must be identical. |

#### (b) TAUBnCMURm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-105 TAUBnCMURm settings for the master channel of the One-Shot Pulse Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection<br>11: Setting prohibited |

**(c) Channel output mode for the master channel**

The channel output mode is not used by this function. However, it can be used by other functions or in Direct Channel Output Mode.

**(d) Simultaneous rewrite for the master channel**

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 14-106 Simultaneous rewrite settings for the master channel of the One-Shot Pulse Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (4) Register settings for the slave channel

#### (a) TAUBnCMORm for the slave channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-107 TAUBnCMORm settings for the slave channel of the One-Shot Pulse Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 100: TAUBnTTOUTm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 1010: Pulse One Count Mode |
| MD0 | 0: Disables start trigger detection during counting<br>1: Enables start trigger detection during counting<br>The value of the MD0 bit of the master and slave channel must be identical. |

#### (b) TAUBnCMURm for the slave channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-108 TAUBnCMURm settings for the slave channel of the One-Shot Pulse Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for the slave channel

**Table 14-109 Control bit settings for Synchronous Channel Output Mode 2**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 0: Independent channel output |
| TOCm | 1: Operation mode 2 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

### (d) Simultaneous rewrite for the slave channel

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 14-110 Simultaneous rewrite settings for the slave channel of the One-Shot Pulse Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is the control channel for simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (5) Operating procedure for One-Shot Pulse Output Function

**Table 14-111 Operating procedure for One-Shot Pulse Output Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Master channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *3 "Register settings for the master channel" on page 626*<br><br>Slave channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *4 "Register settings for the slave channel" on page 628*<br><br>Set the values of the TAUBnCDRm registers of all channels | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm of the master and slave channels to 1 simultaneously.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm (master and slave channels) is set to 1 and the master channel awaits a TAUBnTTINm input.<br>INTTAUBnIm is generated on the master channel. |
| During operation | TAUBnCDRm can be changed at any time. TAUBnCNTm and TAUBnRSF.RSFm can be read at any time.<br><br>TAUBnRDT.RDTm can be changed during operation. | When a valid TAUBnTTINm input edge is detected, TAUBnCNTm of the master channel loads TAUBnCDRm and counts down. When the counter reaches $0000_H$:<br>• INTTAUBnIm (master) is generated<br>• TAUBnCNTm (master) reloads the TAUBnCDRm value and continues count operation<br>• TAUBnCNTm (slave) reloads the TAUBnCDRm value and starts to count down<br>• INTTAUBnIm (slave ) is generated<br>• TAUBnTTOUTm (slave) is set<br>When TAUBnCNTm (slave) reaches $0001_H$:<br>• INTTAUBnIm (slave) is generated<br>• TAUBnTTOUTm (slave) is reset<br><br>If a TAUBnTTINm input is detected on the master channel while the counter is counting, the input is ignored when TAUBnCMORm.MD0 = 0. |
| Stop operation | Set TAUBnTT.TTm of the master and slave channels to 1 simultaneously.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values.<br><br>When TAUBnTOE.TOEm is 0, TAUBnTTOUTm output is initialized to the value set by TAUBnTO.TOm. |

Restart

### (6) Specific timing diagrams

#### (a) TAUBnCDRm (master) = $0000_H$

The following settings apply to this diagram:

- Start trigger detection disabled during counting (TAUBnCMORm.MD0 = 0)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

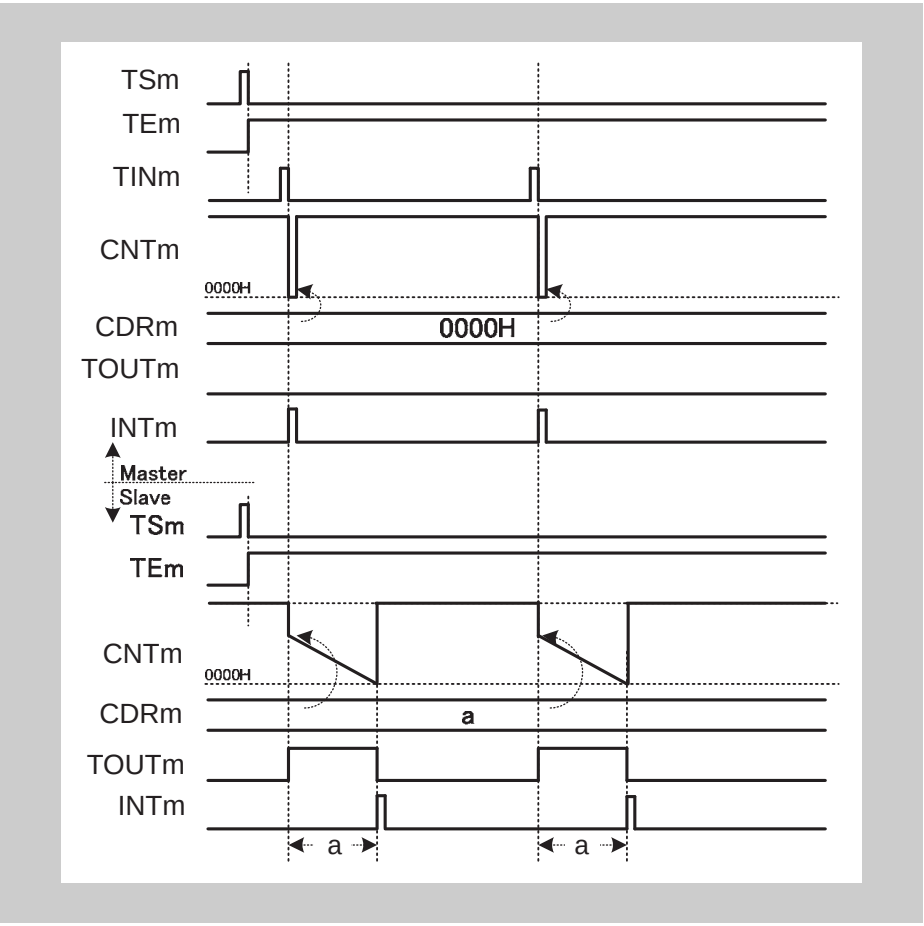



**Figure 14-81 TAUBnCDRm (master) = $0000_H$**

- When a valid TAUBnTTINm input edge is detected, the value $0000_H$ is written to TAUBnCNTm (master). The counter is set to $0000_H$ for one count and returns to $FFFF_H$.

  Thus the slave channel starts to count down one count clock later to TAUBnTTINm (master).

**(b) TAUBnCDRm (slave) = $0000_H$**

The following settings apply to this diagram:

- Start trigger detection disabled during counting (TAUBnCMORm.MD0 = 0)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

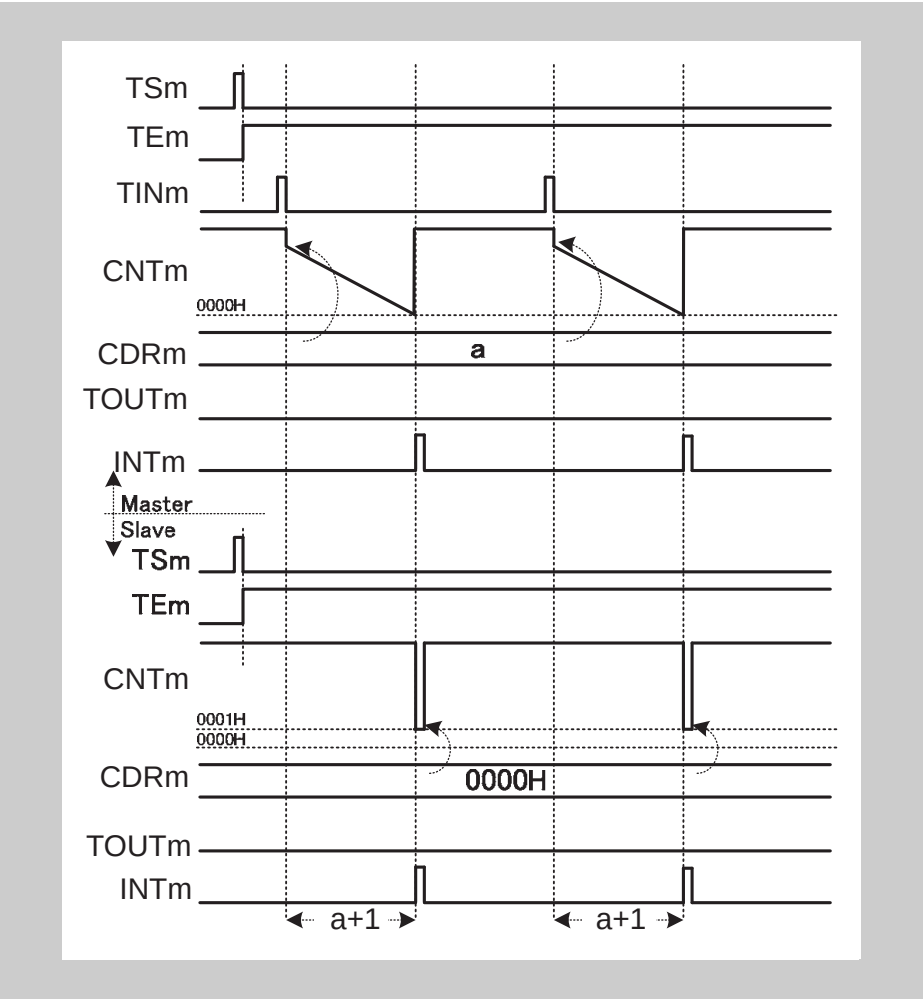



**Figure 14-82 TAUBnCDRm (slave) = $0000_H$**

- The counter of the slave channel reloads the value $0000_H$ and returns to returns to $FFFF_H$ one clock count later.

  TAUBnTTOUTm remains at not active state, because the pulse width is zero.

**(c) TAUBnCMORm.MD0 = 1**

The following settings apply to this diagram:

- Start trigger detection enabled during counting (TAUBnCMORm.MD0 = 1)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

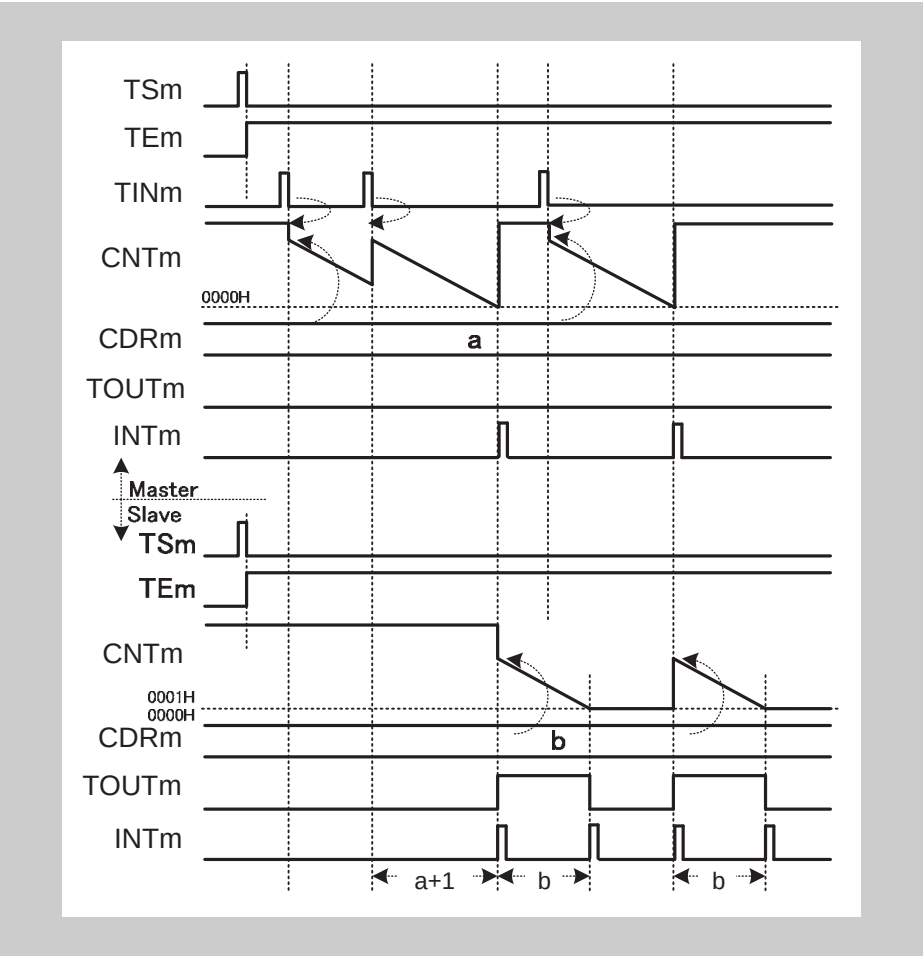



**Figure 14-83 TAUBnCMORm.MD0 = 1**

- If a valid TAUBnTTINm input edge is detected while the counter of the master channel counts down, TAUBnCNTm reloads the value of TAUBnCDRm. The counter restarts to count down.

  This means the delay is extended by the value of TAUBnCNTm at the time a valid TAUBnTTINm input edge is detected.

**(d) Restarting the master channel while the slave channel is counting**

The following settings apply to this diagram:

- Start trigger detection disabled during counting (TAUBnCMORm.MD0 = 0)
- Falling edge detection (TAUBnCMURm.TIS[1:0] = $00_B$)

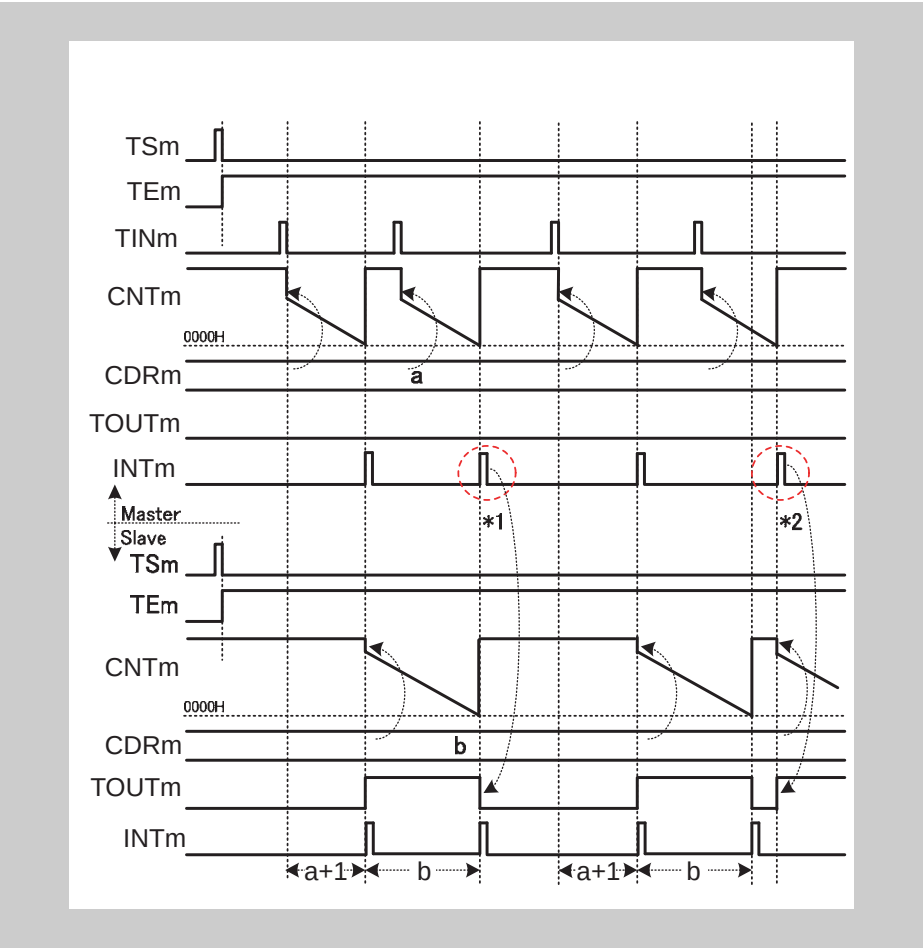



**Figure 14-84 Interval of TAUBnTTINm ≤ delay time + pulse width +1**

- If the master channel generates an interrupt before the counter of the slave channel has reached $0001_H$ or exactly when $0001_H$ is reached (*1), the interrupt (master) is ignored.
- If an interrupt of the master channel occurs when the counter of the slave channel awaits the next trigger, the value of TAUBnCDRm (slave) is reloaded. An interrupt is generated and TAUBnTTOUTm toggles. If TAUBnCNTm (master) has started to count down while the TAUBnCNTm (slave) is still counting (*2), TAUBnTTOUTm is not output with the expected delay time.
- To generate the correct one-shot pulse, the start trigger for the master channel must be detected while the master and slave channels are waiting for the start trigger, and not while they are counting.

## 14.21 Synchronous Triangle PWM Functions

This chapter describes functions that generate a triangle PWM output.

- *14.21.1 “Triangle PWM Output Function”*
- *14.21.2 “Triangle PWM Output Function with Dead Time”*
- *14.21.3 “AD Conversion Trigger Output Function Type 2”*

### 14.21.1 Triangle PWM Output Function

#### (1) Overview

**Summary** This function generates multiple triangle PWM outputs by using a master and one or more slave channels. It enables the pulse cycle (frequency) and the duty cycle of TAUBnTTOUTm to be set using the master and slave channel(s) respectively.

The slave channel generates a carrier cycle from two pulse cycles. The first pulse of the master channel controls the down status and the second pulse controls the up status of the slaves counter.

Counting up and down TAUBnCNTm (slave) means that signal duration of TAUBnTTOUTm (slave) is double that of the difference between TAUBnCDRm (master) +1 and TAUBnCDRm (slave).

**Prerequisites**

- Two channels
- The operation mode of the master channel must be set to Interval Timer Mode, refer to *Table 14-112 “TAUBnCMORm settings for the master channel of the Triangle PWM Output Function” on page 640*
- The operation mode of the slave channel(s) must be set to Up Down Count Mode, refer to *Table 14-116 “TAUBnCMORm settings for the slave channel of the Triangle PWM Output Function” on page 642*
- The channel output mode of the master channel must be set to Independent Channel Output Mode 1 (refer to *14.9 “Channel Output Modes” on page 489* )
- The channel output mode of the slave channel(s) must be set to Synchronous Channel Output Mode 2 (refer to *14.9 “Channel Output Modes” on page 489* )
- The following settings establish TAUBnTTOUTm at high level for the down status of the carrier cycle.
  - If the TAUBnCMORm.MD0 (master) bit is set to 0,TAUBnTO.TOm must be set to 1 while TAUBnTOE.TOEm is 0.
  - If the TAUBnCMORm.MD0 (master) bit is set to 1, TAUBnTO.TOm must be set to 0 while TAUBnTOE.TOEm is 0.

**Description** The counters are started by setting the channel trigger bit (TAUBnTS.TSm) to 1 for every channel. This in turn sets TAUBnTE.TEm, enabling count operation. The current values of TAUBnCDRm (master and slave) are written to TAUBnCNTm (master and slave) and the counters start to count down from these values. Depending on the setting of the master channel TAUBnCMORm.MD0 bit an interrupt is generated and TAUBnTTOUTm signal of the master toggles.

- Master channel:

  When the counter of the master channel reaches $0000_H$, pulse cycle time has elapsed, INTTAUBnIm is generated and the TAUBnTTOUTm signal toggles. TAUBnCNTm then reloads the TAUBnCDRm value and counts down.

- Slave channel:

  The INTTAUBnIm of the master channel triggers the counter of the slave channel:

  – If the slave counter currently counts down, it changes count direction .

  – If the slave counter currently counts up, the value of TAUBnCDRm is reloaded and the counter counts down.

  When the counter of the slave channel reaches $0001_H$ while counting up or down, INTTAUBnIm is generated and the TAUBnTTOUTm (slave) signal toggles:

  – It is set in the count-down status

  – It is reset in the count-up status

  The counter continues to count down or up and awaits the next INTTAUBnIm of the master channel.

TAUBnTTOUTm can be switched between positive and negative phase setting TAUBnTOL.TOLm during operation.

The counters can be stopped by setting TAUBnTT.TTm to 1 for the master and slave channel(s), which in turn sets TAUBnTE.TEm to 0. TAUBnCNTm and TAUBnTTOUTm of master and slave channel(s) stop but retain their values.

**Note** If a forced restart is executed during operation, TAUBnTTOUTm is not output as a triangle PWM signal.

**Conditions** Simultaneous rewrite can be used with this function. Please refer to *14.8 “Simultaneous Rewrite” on page 479*

**(2) Equations**

Pulse cycle = (TAUBnCDRm (master) + 1) x count clock cycle

Carrier cycle (down/up) = (TAUBnCDRm (master) + 1) × 2 × count clock cycle

Duty cycle [%] =
[(TAUBnCDRm (master) + 1 - TAUBnCDRm (slave) )/ (TAUBnCDRm (master) + 1)] × 100

– Duty cycle = 100 %

  TAUBnCDRm (slave) = $0000_H$

– Duty cycle = 0 %

  TAUBnCDRm (slave) ≥ TAUBnCDRm (master) + 1

### (3) Block diagram and general timing diagram

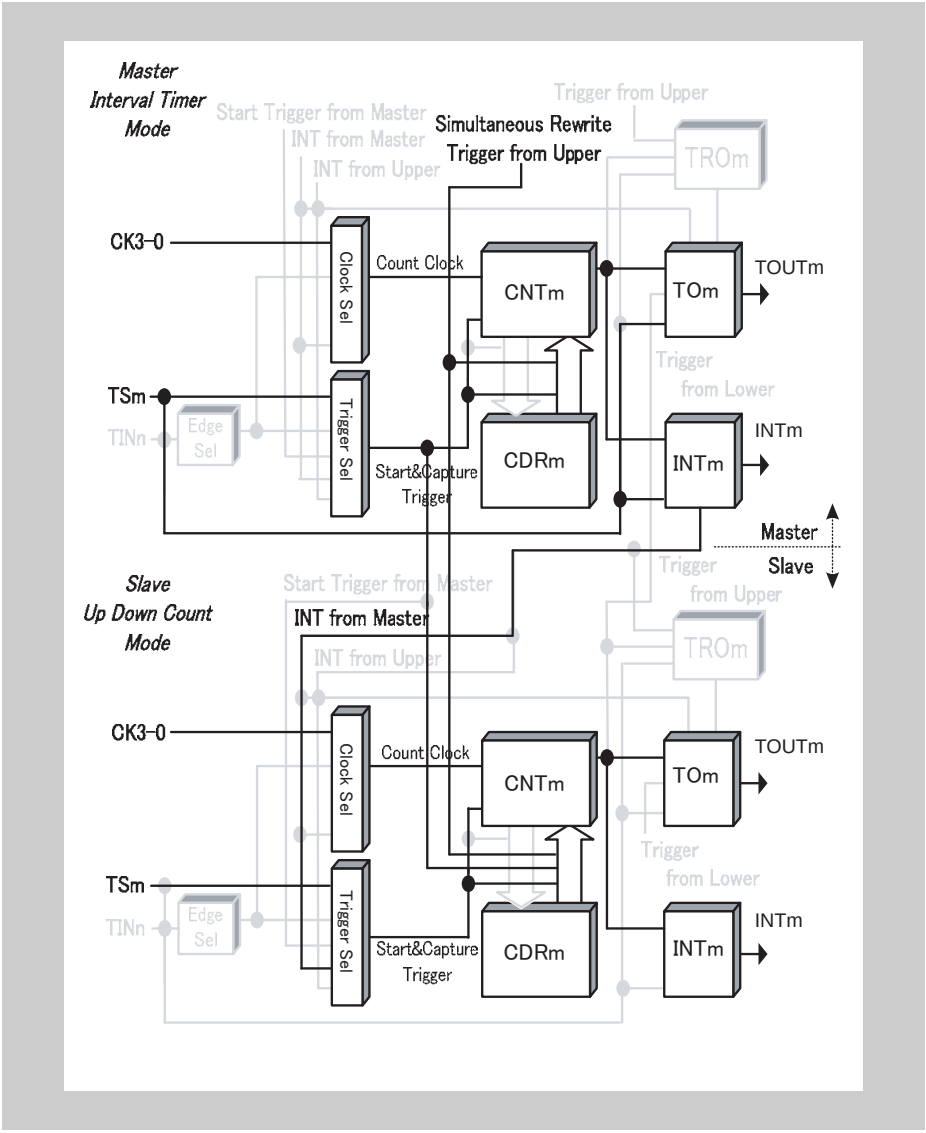



**Figure 14-85 Block diagram for Triangle PWM Output Function**

The following settings apply to the general timing diagram:

- Master channel
  - INTTAUBnIm is generated at operation start (TAUBnCMORm.MD0 = 1)

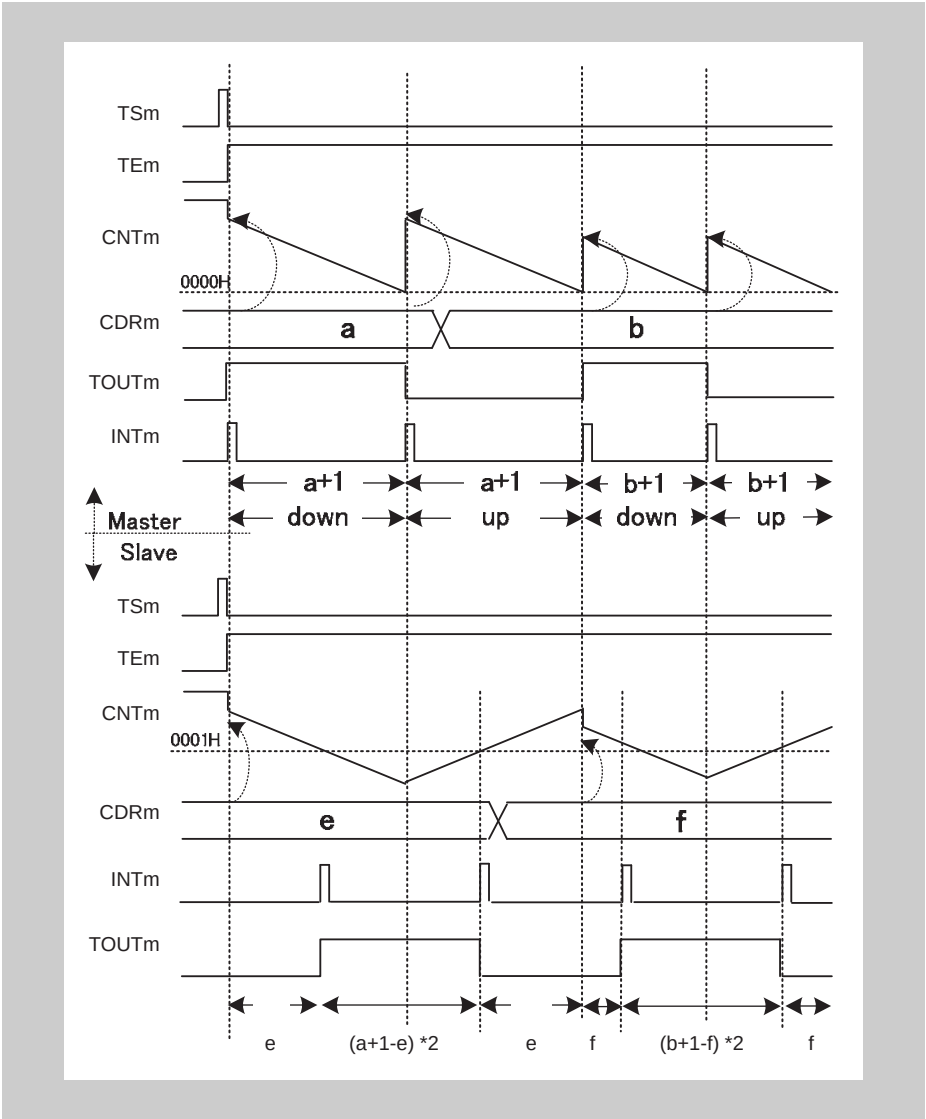



**Figure 14-86 General timing diagram for Triangle PWM Output Function**

### (4) Register settings for the master channel

#### (a) TAUBnCMORm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-112 TAUBnCMORm settings for the master channel of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 1: Channel is master channel |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUBnIm not generated and TAUBnTTOUTm does not toggle at operation start<br>1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start |

#### (b) TAUBnCMURm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-113 TAUBnCMURm settings for the master channel of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for the master channel

**Table 14-114 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 0: Independent channel output |
| TOCm | 0: Operation mode 1 (= Toggle mode if TAUBnTOM.TOMm = 0) |
| TOLm | 0: Positive logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

### (d) Simultaneous rewrite for the master channel

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 14-115 Simultaneous rewrite settings for the master channel of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is monitored for the simultaneous rewrite trigger<br>1: An upper channel outside the channel group is monitored for the simultaneous rewrite trigger |
| RDM.RDMm | 1: The simultaneous rewrite trigger signal is generated when the master channel starts counting and the corresponding slave channel is at the peak of a triangular wave |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (5) Register settings for the slave channel(s)

#### (a) TAUBnCMORm for the slave channel(s)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-116 TAUBnCMORm settings for the slave channel of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 111: The up/down output trigger signal TAUBnTUDSm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 1001: Up Down Count Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start |

#### (b) TAUBnCMURm for the slave channel(s)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-117 TAUBnCMURm settings for the slave channel of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for the slave channel(s)

**Table 14-118 Control bit settings for Synchronous Channel Output Mode 2**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 1: Synchronous channel operation |
| TOCm | 1: Operation mode 2 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

### (d) Simultaneous rewrite for the slave channel(s)

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 14-119 Simultaneous rewrite settings for the slave channel of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is monitored for the simultaneous rewrite trigger<br>1: An upper channel is monitored for the simultaneous rewrite trigger |
| RDM.RDMm | 1: The simultaneous rewrite trigger signal is generated when the master channel starts counting and the corresponding slave channel is at the peak of a triangular wave |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (6) Operating procedure for Triangle PWM Output Function

**Table 14-120 Operating procedure for Triangle PWM Output Function**

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Master channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *4 "Register settings for the master channel" on page 640*<br><br>Slave channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *5 "Register settings for the slave channel(s)" on page 642*<br><br>Set the values of the TAUBnCDRm registers of all channels | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUBnTS.TSm of the master and slave channels to 1 simultaneously.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm (master and slave channels) is set to 1 and the counters of the master and slave channels start.<br>INTTAUBnIm is generated on the master channel, depending on TAUBnCMORm.MD0. |
| During operation | TAUBnCDRm can be changed at any time.<br>TAUBnCNTm and TAUBnRSF.RSFm can be read at any time.<br><br>TAUBnRDT.RDTm can be changed during operation. | TAUBnCNTm of the master and slave channel loads TAUBnCDRm and counts down. When the counter of the master channel reaches $0000_H$:<br>• INTTAUBnIm (master) is generated<br>• TAUBnTTOUTm (master) toggles<br>• TAUBnCNTm (master) reloads the TAUBnCDRm value and continues count operation.<br>• TAUBnCNTm (slave) reloads the TAUBnCDRm value or counts in the reverse direction.<br>When TAUBnCNTm of the slave = $0001_H$:<br>• INTTAUBnIm (slave) is generated<br>• TAUBnTTOUTm (slave) is set (in count-down status) or reset (in count-up status) |
| Stop operation | Set TAUBnTT.TTm of the master and slave channels to 1 simultaneously.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values.<br><br>When TAUBnTOE.TOEm is 0, TAUBnTTOUTm is initialized to the value set by TAUBnTO.TOm. |

Restart

### (7) Specific timing diagrams

#### (a) Duty cycle = 0 %

The following settings apply to the general timing diagram:

- Master channel:
  - INTTAUBnIm is generated at operation start (TAUBnCMORm.MD0 = 1)
  - TAUBnCDRm = a = $5_H$
- Slave channel:
  - TAUBnCDRm = $6_H$

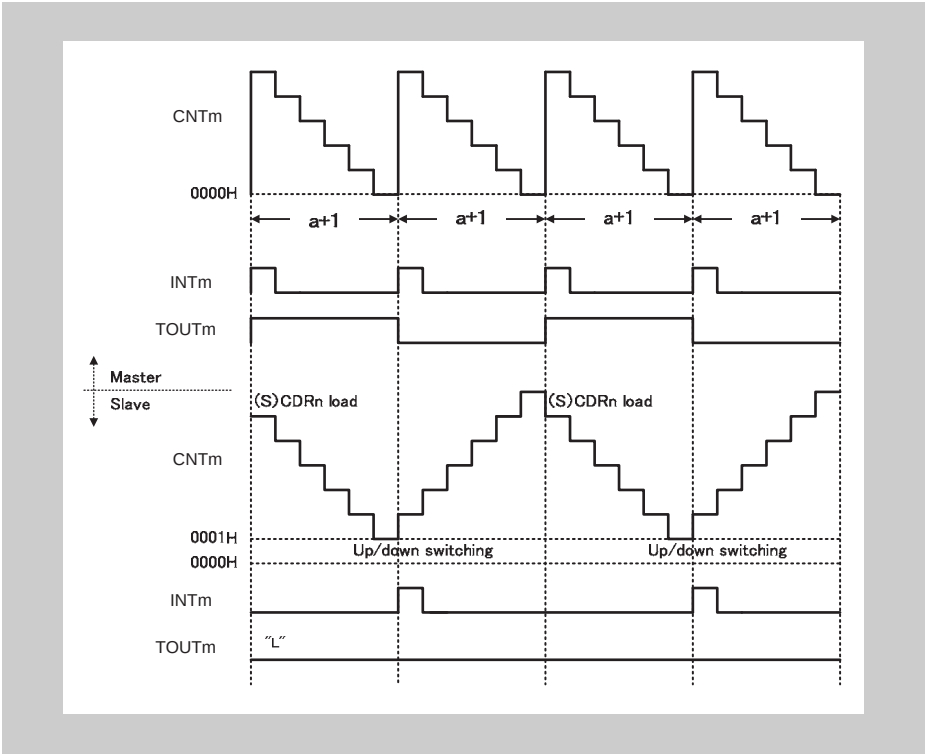



**Figure 14-87 TAUBnCDRm (slave) ≥ TAUBnCDRm (master) + 1**

- If TAUBnCDRm (slave) ≥ TAUBnCDRm (master) the counter of slave channel cannot reach $0001_H$ during *counting down*. The set signal is never detected, so TAUBnTTOUTm remains at low state.

**(b) Duty cycle = 100 %**

The following settings apply to the general timing diagram:

- Master channel:
  - INTTAUBnIm is generated at operation start (TAUBnCMORm.MD0 = 1)
  - TAUBnCDRm = a = $5_H$
- Slave channel:
  - TAUBnCDRm = $0_H$

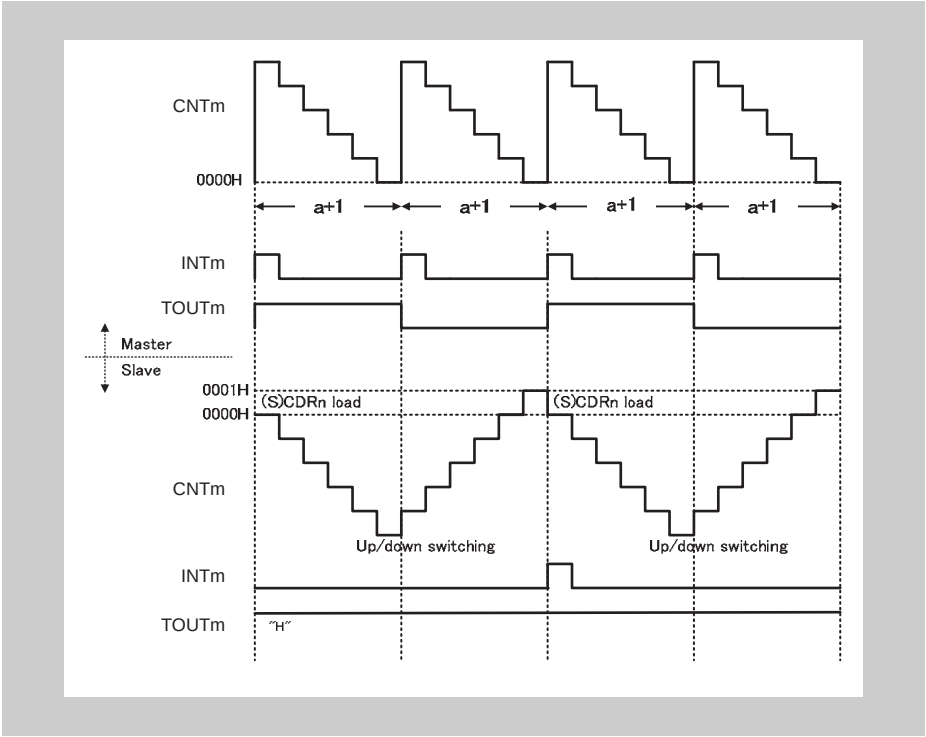



**Figure 14-88 TAUBnCDRm (slave) = $0000_H$**

- If TAUBnCDRm (slave) = $0000_H$ the counter of slave channel cannot reach $0001_H$ during *counting up*. The reset signal is never detected, so TAUBnTTOUTm remains at high state.

## 14.21.2 Triangle PWM Output Function with Dead Time

### (1) Overview

**Summary** This function generates multiple triangle PWM outputs with a defined dead time by using a master and two or more slave channels. The resulting PWM signals are output via TAUBnTTOUTm of the slave channels 2 and 3. It enables the pulse cycle (frequency) and the duty cycle of TAUBnTTOUTm to be set using the master and slave channel(s) respectively.

Slave channel 2 generates a carrier cycle from two pulse cycles from the master channel. The first pulse controls the down status and the second pulse controls the up status of the slaves counter.

Counting up and down TAUBnCNTm (slave 2) means that signal duration of TAUBnTTOUTm (slave) is double that of the difference between TAUBnCDRm (master) +1 and TAUBnCDRm (slave 2).

An interrupt on slave 2 causes TAUBnTTOUTm of the slave channels to toggle. Depending on the settings of TAUBnTDL.TDLm, delay time is added to positive or negative logic side of the signal (i.e. whether TAUBnTTOUTm toggles immediately or after dead time has elapsed). The duration of the dead time is specified by slave channel 3.

**Prerequisites**

- Three channels
- The operation mode of the master channel must be set to Interval Timer Mode, refer to *Table 14-122 “TAUBnCMORm settings for the master channel of the Triangle PWM Output Function with Dead Time” on page 652*
- Slave channel 1 is not used for this function. This ensures that slave channel 2 is an odd channel, and slave channel 3 is an even channel.
- The operation mode of slave channel 2 must be set to Up Down Mode, refer to *Table 14-126 “TAUBnCMORm settings for slave channel 2 of the Triangle PWM Output Function with Dead Time” on page 654*
  Furthermore, slave channel 2 must be an even channel
- The operation mode of slave channel 3 must be set to One Count Mode, refer to *Table 14-130 “TAUBnCMORm settings for slave channel 3 of the Triangle PWM Output Function with Dead Time” on page 656*
  Furthermore, slave channel 3 must be an odd channel
- The channel output mode of the master channel must be set to Independent Channel Output Mode 1 (refer to *14.9 “Channel Output Modes” on page 489* )
- The channel output mode of the slave channels 2 and 3 must be set to Synchronous Channel Output Mode 2 with Dead Time Output (refer to *14.9 “Channel Output Modes” on page 489* )
- The following settings establish TAUBnTTOUTm at high level for the down status of the carrier cycle.
  - If the TAUBnCMORm.MD0 (master) bit is set to 0,TAUBnTO.TOm must be set to 1 while TAUBnTOE.TOEm is 0.
  - If the TAUBnCMORm.MD0 (master) bit is set to 1, TAUBnTO.TOm must be set to 0 while TAUBnTOE.TOEm is 0.

**Note** Slave channel 1 is not used for Triangle PWM Output Function with Dead Time.

**Description** The counters are started by setting the channel trigger bits (TAUBnTS.TSm) to 1. This in turn sets TAUBnTE.TEm, enabling count operation. The current values of TAUBnCDRm is written to TAUBnCNTm and the counters start to

count down from these values. Depending on the setting of the master channel TAUBnCMORm.MD0 bit an interrupt is generated and TAUBnTTOUTm signal of the master toggles.

- Master channel:

  When the counter of the master channel reaches $0000_H$, INTTAUBnIm is generated and the TAUBnTTOUTm signal toggles. The counter reloads the TAUBnCDRm value and counts down.

- Slave channel 2:

  The INTTAUBnIm of the master channel triggers the counter of the slave channel 2:

  – If the slave counter currently counts down, it changes count direction.

  – If the slave counter currently counts up, the value of TAUBnCDRm is reloaded and the counter counts down.

  The counter continues to count down or up and awaits the next INTTAUBnIm of the master channel.

- Slave channel 3:

  INTTAUBnIm of slave channel 2 triggers the counter of slave channel 3. The current value of TAUBnCDRm (slave 3) is written to TAUBnCNTm (slave 3) and the counter starts to count down from this value.
  When the counter reaches $0000_H$, dead time has elapsed, INTTAUBnIm is generated. The counter returns to $FFFF_H$ and awaits the next INTTAUBnIm of slave channel 2.

An interrupt on slave channel 2 causes TAUBnTTOUTm to toggle as follows:

- It is set by the interrupt when slave channel 2 is counting down
- It is reset by the interrupt when slave channel 2 is counting up.

However, the TAUBnTDL.TDLm settings of the corresponding channel specify whether it is set/reset immediately, or after dead time has elapsed, as shown in *Table 14-121 “Behavior of TAUBnTTOUTm when an interrupt occurs on slave channel 2” on page 649* .

The TAUBnTOL.TOLm settings specify whether set corresponds to a high signal (TAUBnTOL.TOLm = 0) or a low signal (TAUBnTOL.TOLm = 1).

In Triangle PWM Output Function with Dead Time, TAUBnTTOUTm can be switched between positive and negative phase by setting TAUBnTOL.TOLm during operation.

The counter can be stopped by setting TAUBnTT.TTm to 1 for the master and slave channel(s), which in turn sets TAUBnTE.TEm to 0. TAUBnCNTm and TAUBnTTOUTm of master and slave channel(s) stop but retain their values.

TAUBnCDRm value of slave channel 2 can be set to $0000_H$ to output 100 % TAUBnTTOUTm.

**Note** If a forced restart is executed during operation, TAUBnTTOUTm is not output as a triangle PWM signal.

**Conditions** Simultaneous rewrite can be used with this function. Please refer to *14.8 "Simultaneous Rewrite" on page 479*

TAUBnTOL.TOLm bits should be set before the counter starts, and slave channels 2 and 3 should have opposite TAUBnTOL.TOLm settings or opposite TAUBnTDL.TDLm settings.

**Table 14-121 Behavior of TAUBnTTOUTm when an interrupt occurs on slave channel 2**

| TAUBnTDL.TDLm | Count direction of slave channel 2 when interrupt is generated | TAUBnTTOUTm toggles |
|---|---|---|
| 0 | Down | Set after dead time has elapsed |
| | Up | Reset immediately |
| 1 | Down | Set immediately |
| | Up | Reset after dead time has elapsed |

**Notes**

1. When the set and reset condition for TAUBnTTOUTm occur simultaneously:
   - If TAUBnTOL.TOLm = 0, the reset condition has priority
   - If TAUBnTOL.TOLm = 1, the set condition has priority .
2. To generate a two-phase PWM output with added dead time, the settings of TAUBnTOL.TOLm bit of slave channels 2 and 3 must not be changed.

### (2) Equations

Pulse cycle = (TAUBnCDRm (master) + 1) × count clock cycle

Carrier cycle (down/up) = (TAUBnCDRm (master) + 1) × 2 × count clock cycle

Duty cycle [%] =
[(TAUBnCDRm (master) + 1 – TAUBnCDRm (slave) )/ (TAUBnCDRm (master) + 1)] × 100

– Duty cycle = 100 %

  TAUBnCDRm (slave) = $0000_H$

– Duty cycle = 0 %

  TAUBnCDRm (slave) ≥ TAUBnCDRm (master) + 1

PWM signal width (positive phase) = [((TAUBnCDRm (master) + 1 – TAUBnCDRm (slave 2) × 2) – (TAUBnCDRm (slave 3) + 1)] × count clock cycle

PWM signal width (negative phase) = [((TAUBnCDRm (master) + 1 – TAUBnCDRm (slave 2) × 2) + (TAUBnCDRm (slave 3) + 1)] × count clock cycle

### (3) Block diagram and general timing diagram

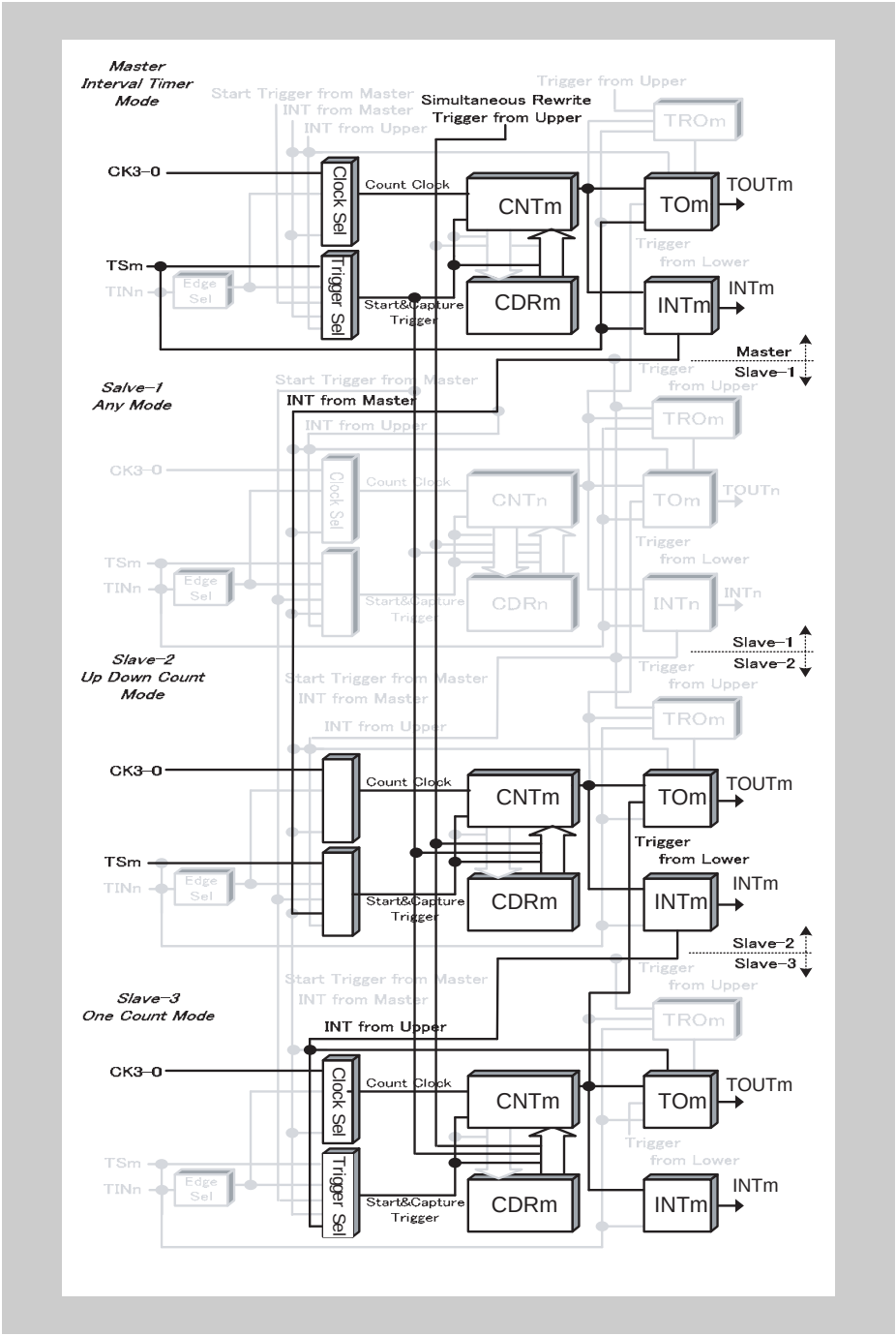



**Figure 14-89 Block diagram for Triangle PWM Output Function with Dead Time**

The following settings apply to the general timing diagram:

- Master channel:
  - INTTAUBnIm is generated at operation start (TAUBnCMORm.MD0 = 1)
- Slave channel 2:
  - INTTAUBnIm not generated at operation start (TAUBnCMORm.MD0 = 0)
  - TAUBnTDL.TDLm = 0
  - Positive logic (TAUBnTOL.TOLm = 0)
- Slave channel 3:
  - INTTAUBnIm is generated at operation start (TAUBnCMORm.MD0 = 1)
  - TAUBnTDL.TDLm = 1
  - Negative logic (TAUBnTOL.TOLm = 1)

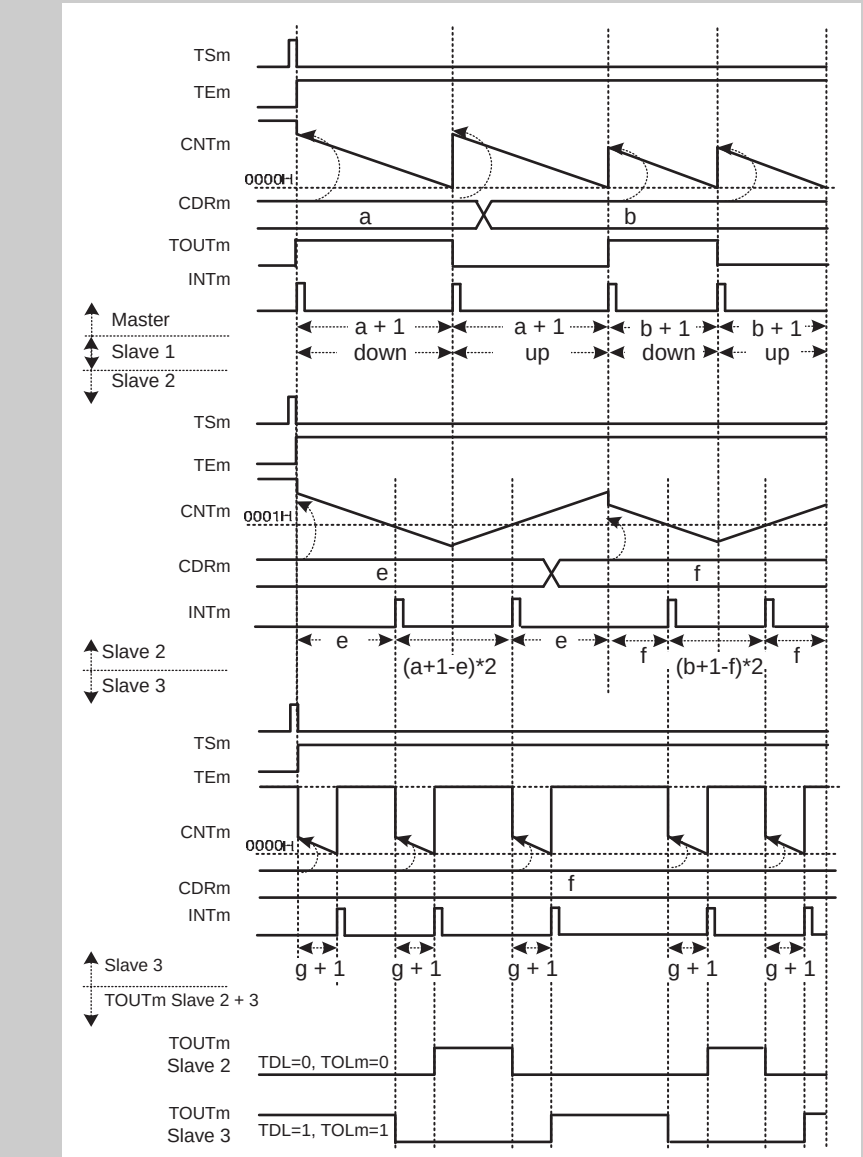



**Figure 14-90 General timing diagram for Triangle PWM Output Function with Dead Time**

### (4) Register settings for the master channel

#### (a) TAUBnCMORm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-122 TAUBnCMORm settings for the master channel of the Triangle PWM Output Function with Dead Time**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 1: Channel is master channel |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUBnIm not generated and TAUBnTTOUTm does not toggle at operation start<br>1: Generates INTTAUBnIm and toggles TAUBnTTOUTm at operation start |

#### (b) TAUBnCMURm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-123 TAUBnCMURm settings for the master channel of the Triangle PWM Output Function with Dead Time**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for the master channel

**Table 14-124 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 0: Independent channel output |
| TOCm | 0: Operation mode 1 (= Toggle mode if TAUBnTOM.TOMm = 0) |
| TOLm | 0: Positive logic |
| TDEm | 0: Disables dead time operation |
| TDLm | 0: When dead time operation is disabled (TAUBnTDE.TDEm = 0), set these bits to 0 |

### (d) Simultaneous rewrite for the master channel

The simultaneous rewrite settings of the master and slave channels must be identical.

**Table 14-125 Simultaneous rewrite settings for the master channel of the Triangle PWM Output Function with Dead Time**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is monitored for the simultaneous rewrite trigger<br>1: An upper channel outside the channel group is monitored for the simultaneous rewrite trigger |
| RDM.RDMm | 1: The simultaneous rewrite trigger signal is generated when the master channel starts counting and the corresponding slave channel is at the peak of a triangular wave |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (5) Register settings for slave channel 2

#### (a) TAUBnCMORm for slave channel 2

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-126 TAUBnCMORm settings for slave channel 2 of the Triangle PWM Output Function with Dead Time**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 111: The up/down output trigger signal TAUBnTUDSm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 1001: Up Down Count Mode |
| MD0 | 0: INTTAUBnIm not generated at operation start |

#### (b) TAUBnCMURm for slave channel 2

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-127 TAUBnCMURm settings for slave channel 2 of the Triangle PWM Output Function with Dead Time**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for slave channel 2

**Table 14-128 Control bit settings for Synchronous channel Output Mode 2 with Dead Time Output**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 1: Synchronous channel operation |
| TOCm | 1: Operation mode 2 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 1: Enables dead time operation |
| TDLm | 0: Dead time is added to the positive phase<br>1: Dead time is added to the negative phase |

### (d) Simultaneous rewrite for slave channel 2

The simultaneous rewrite settings of the master and slave channels must be identical.

**Table 14-129 Simultaneous rewrite settings for slave channel 2 of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is monitored for the simultaneous rewrite trigger<br>1: An upper channel outside the channel group is monitored for the simultaneous rewrite trigger |
| RDM.RDMm | 1: The simultaneous rewrite trigger signal is generated when the master channel starts counting and the corresponding slave channel is at the peak of a triangular wave |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (6) Register settings for slave channel 3

#### (a) TAUBnCMORm for slave channel 3

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:1] | | | | MD0 |

**Table 14-130 TAUBnCMORm settings for slave channel 3 of the Triangle PWM Output Function with Dead Time**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS0 | 0: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 111: The up/down output trigger signal TAUBnTUDSm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 1: Enables start trigger detection during counting |

#### (b) TAUBnCMURm for slave channel 3

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — | — | — | — | — | — | — | TIS[1:0] | |

**Table 14-131 TAUBnCMURm settings for slave channel 3 of the Triangle PWM Output Function with Dead Time**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for slave channel 3

**Table 14-132 Control bit settings for Synchronous channel Output Mode 2 with Dead Time Output**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 1: Synchronous channel operation |
| TOCm | 1: Operation mode 2 |
| TOLm | 0: Positive logic<br>1: Inverted logic |
| TDEm | 1: Enables dead time operation |
| TDLm | 0: Dead time is added to the positive phase<br>1: Dead time is added to the negative phase |

### (d) Simultaneous rewrite for slave channel 3

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 14-133 Simultaneous rewrite settings for slave channel 3 of the Triangle PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDS.RDSm | 0: The master channel is monitored for the simultaneous rewrite trigger<br>1: An upper channel outside the channel group is monitored for the simultaneous rewrite trigger |
| RDM.RDMm | 1: The simultaneous rewrite trigger signal is generated when the master channel starts counting and the corresponding slave channel is at the peak of a triangular wave |
| RDC.RDCm | 0: Channel is not monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger. If TAUBnRDS.RDSm = 0, the master channel is monitored for the simultaneous rewrite trigger regardless of the value of this bit. |

### (7) Operating procedure for Triangle PWM Output Function with Dead Time

Table 14-134 Operating procedure for Triangle PWM Output Function with Dead Time (1/2)

| | Operation | Status of TAUBn |
|---|---|---|
| Initial channel setting | Master channel: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *4 "Register settings for the master channel" on page 652*<br><br>Slave channel 2: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *5 "Register settings for slave channel 2" on page 654*<br><br>Slave channel 3: set the TAUBnCMORm and TAUBnCMURm registers and the channel output mode as described in *6 "Register settings for slave channel 3" on page 656*<br><br>Set the values of the TAUBnCDRm registers of all channels | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |

**Table 14-134 Operating procedure for Triangle PWM Output Function with Dead Time (2/2)**

| | Operation | Status of TAUBn |
|---|---|---|
| Start operation | Set TAUBnTS.TSm of the master and slave channels to 1 simultaneously.<br>TAUBnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm (master and slave channels) is set to 1 and the counters of the master and slave channels start.<br>INTTAUBnIm is generated on the master channel., depending on the setting of TAUBnCMORm.MD0. |
| During operation | TAUBnCDRm can be changed at any time.<br>TAUBnCNTm and TAUBnRSF.RSFm can be read at any time.<br><br>TAUBnRDT.RDTm can be changed during operation. | TAUBnCNTm of the master channel and slave channel 2 load TAUBnCDRm and count down.<br>When the counter of the master channel reaches $0000_H$:<br>• INTTAUBnIm (master) is generated<br>• TAUBnCNTm (master) reloads the TAUBnCDRm value and continues count operation<br>• TAUBnCNTm (slave 2) reloads the TAUBnCDRm value or counts in the reverse direction<br>When TAUBnCNTm (slave 2) reaches $0001_H$:<br>• INTTAUBnIm (slave 2) is generated<br>• TAUBnTTOUTm is reset (TAUBnTOL.TOLm = 0)<br>• TAUBnCNTm of slave channel 3 reloads the TAUBnCDRm value and counts down<br>When TAUBnCNTm of slave channel 3 = $0000_H$:<br>• INTTAUBnIm is generated<br>• TAUBnTTOUTm is set (TAUBnTOL.TOLm = 0)<br>Whether TAUBnTTOUTm (slave 2 and 3) are set/reset with INTTAUBnIm (slave 2) simultaneously or after dead time has elapsed depends on the settings in TAUBnTDL.TDLm and TAUBnTOL.TOLm (refer to *Table 14-121 "Behavior of TAUBnTTOUTm when an interrupt occurs on slave channel 2" on page 649* ). |
| Stop operation | Set TAUBnTT.TTm of the master and slave channels to 1 simultaneously.<br>TAUBnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUBnTE.TEm is cleared to 0 and the counter stops.<br>TAUBnCNTm and TAUBnTTOUTm stop and retain their current values.<br><br>When TAUBnTOE.TOEm is 0, TAUBnTTOUTm of slave channels 2 and 3 is initialized to the value set by TAUBnTO.TOm. |

Restart

### (8) Specific timing diagrams

#### (a) Duty cycle = 0 %

The following settings apply to the diagram below:

- Slave channel 2:
  - Positive logic (TAUBnTOL.TOLm = 0)
- Slave channel 3:
  - Negative logic (TAUBnTOL.TOLm = 1)

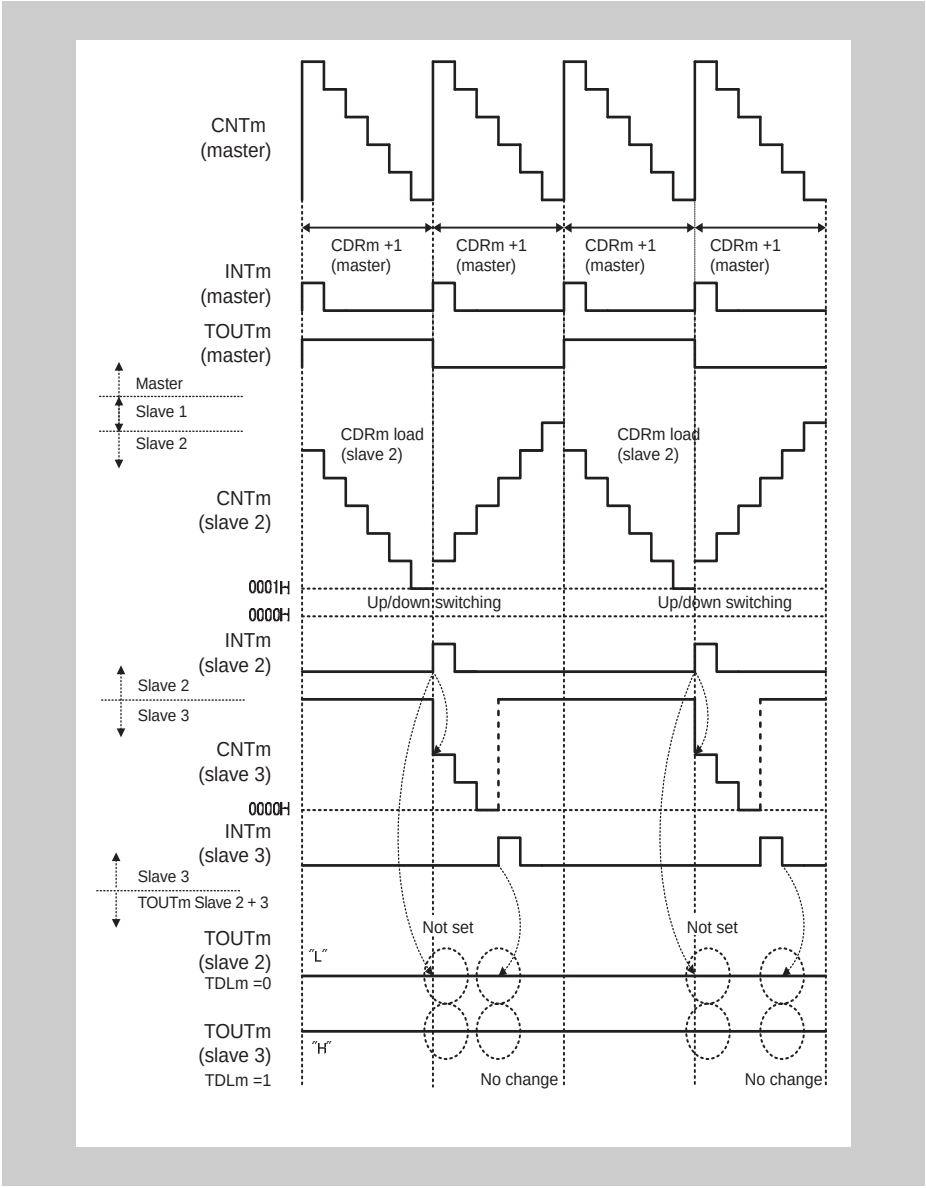



**Figure 14-91 TAUBnCDRm (slave) ≥ TAUBnCDRm (master) + 1**

- If TAUBnCDRm (slave 2) ≥ TAUBnCDRm (master) the counter of slave channel cannot reach $0000_H$ during counting down. Therefore TAUBnTTOUTm cannot toggle, i.e. it remains at its initial state. The interrupt from slave channel 2 occurs during count up, therefore it is a reset signal.

**(b) Duty cycle = 100 %**

The following settings apply to the diagram below:

- Slave channels 2:
  - Positive logic (TAUBnTOL.TOLm = 0)
- Slave channels 3:
  - Negative logic (TAUBnTOL.TOLm = 1)

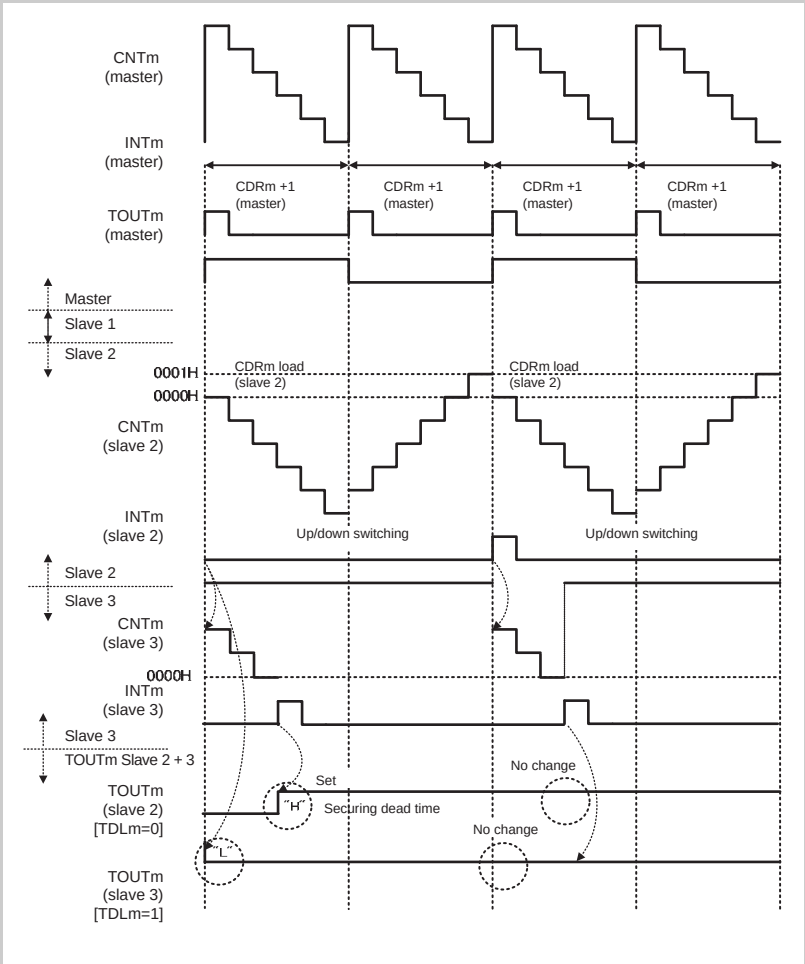



**Figure 14-92 TAUBnCDRm (slave) = $0000_H$**

- If TAUBnCDRm (slave 2) = $0000_H$ the counter of slave channel cannot reach $0001_H$ while counting up and therefore cannot generate an INTTAUBnIm while counting up.
  - The set conditions for a channel in which TAUBnTDL.TDLm = 0 are met after dead time has elapsed. TAUBnTTOUTm toggles but remains in the new state because the reset conditions never occur for such a channel.
  - Slave channel 3 in the diagram above is set when the counter starts. However, the reset conditions for a channel in which TAUBnTDL.TDLm = 1 never occur so TAUBnTTOUTm remains in its initial state for such a slave channel. In slave 3 in the diagram above, the initial state is high because TAUBnTOL.TOLm = 1.

**(c) TAUBnTTOUTm (slave 2) = 0 % and TAUBnTTOUTm (slave 3) > 0 %**

The following settings apply to the diagram below:

- Slave channel 2:
  - Positive logic (TAUBnTOL.TOLm = 0)
- Slave channel 3:
  - Negative logic (TAUBnTOL.TOLm = 1)

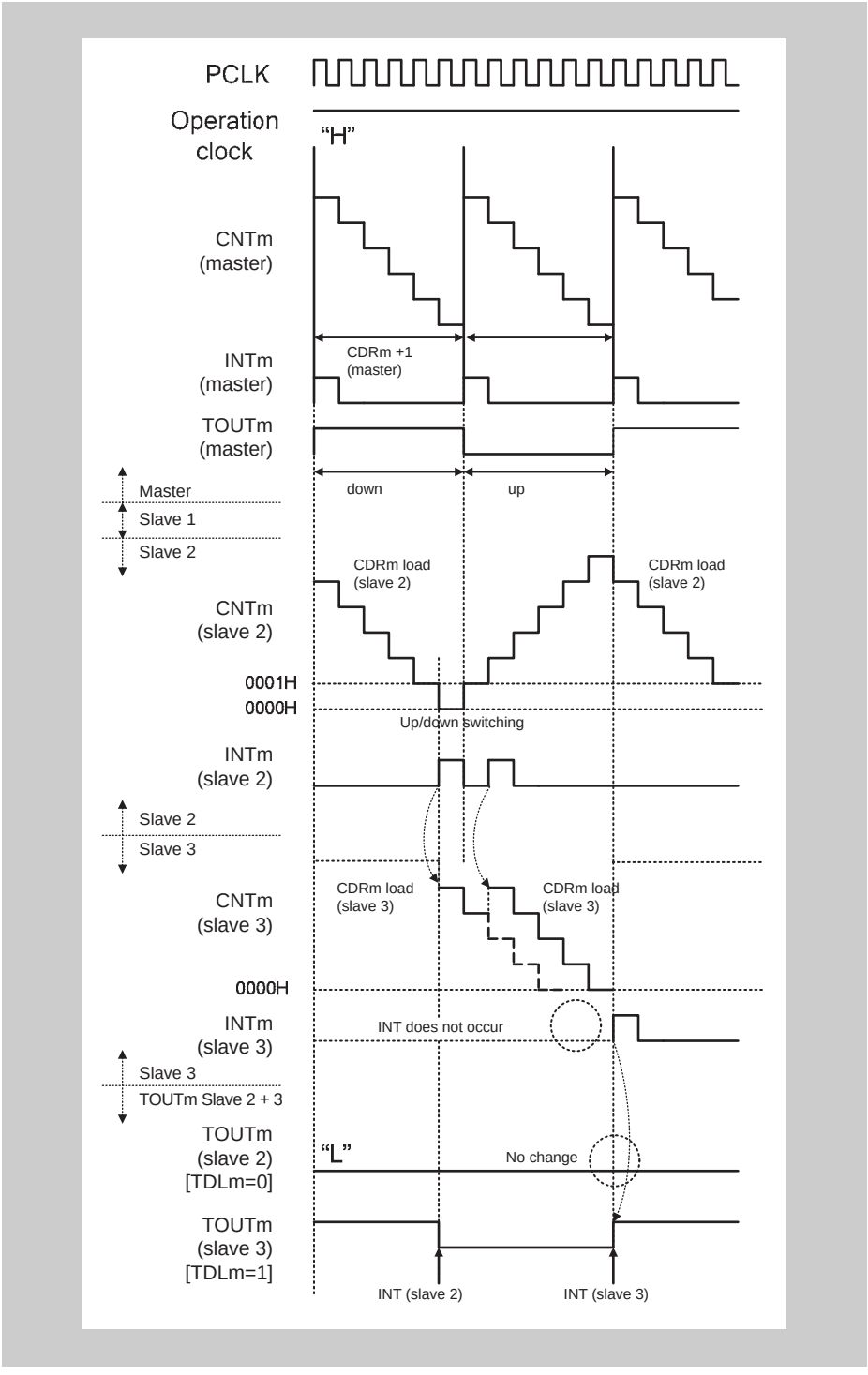



**Figure 14-93 TAUBnCDRm (master) = $0005_H$, TAUBnCDRm (slave 2) = $0005_H$, TAUBnCDRm (slave 3) = $0004_H$**

- When the counter of slave channel 2 reaches $0000_H$, INTTAUBnIm (slave 2) is generated. The counter of slave channel 3 starts to count down.
- If another INTTAUBnIm (slave 2) is generated while the counter of slave channel 3 is still counting down, the value of TAUBnCDRm (slave 3) is reloaded and the counter restarts counting down from this value.
- In the diagram above, the first interrupt on channel 2 occurs while the counter is counting down, and the second whilst it is counting up.
- After the first interrupt, a slave for which TAUBnTDL.TDLm = 0 waits for dead time to elapse before setting. However, before the dead time has elapsed, another interrupt occurs on slave 2, this time while the counter is counting up. This acts as a reset signal, meaning that a channel for which TAUBnTDL.TDLm = 0 always remains inactive.
- TAUBnTTOUTm of a slave channel for which TAUBnTDL.TDLm = 1 is set and reset as normal when the corresponding INTTAUBnIm is generated.

**(d) TAUBnTTOUTm (slave 2) > 0 % and TAUBnTTOUTm (slave 3) = 100 %**

The following settings apply to the diagram below:

- Slave channel 2:
  - Positive logic (TAUBnTOL.TOLm = 0)
- Slave channel 3:
  - Negative logic (TAUBnTOL.TOLm = 1)

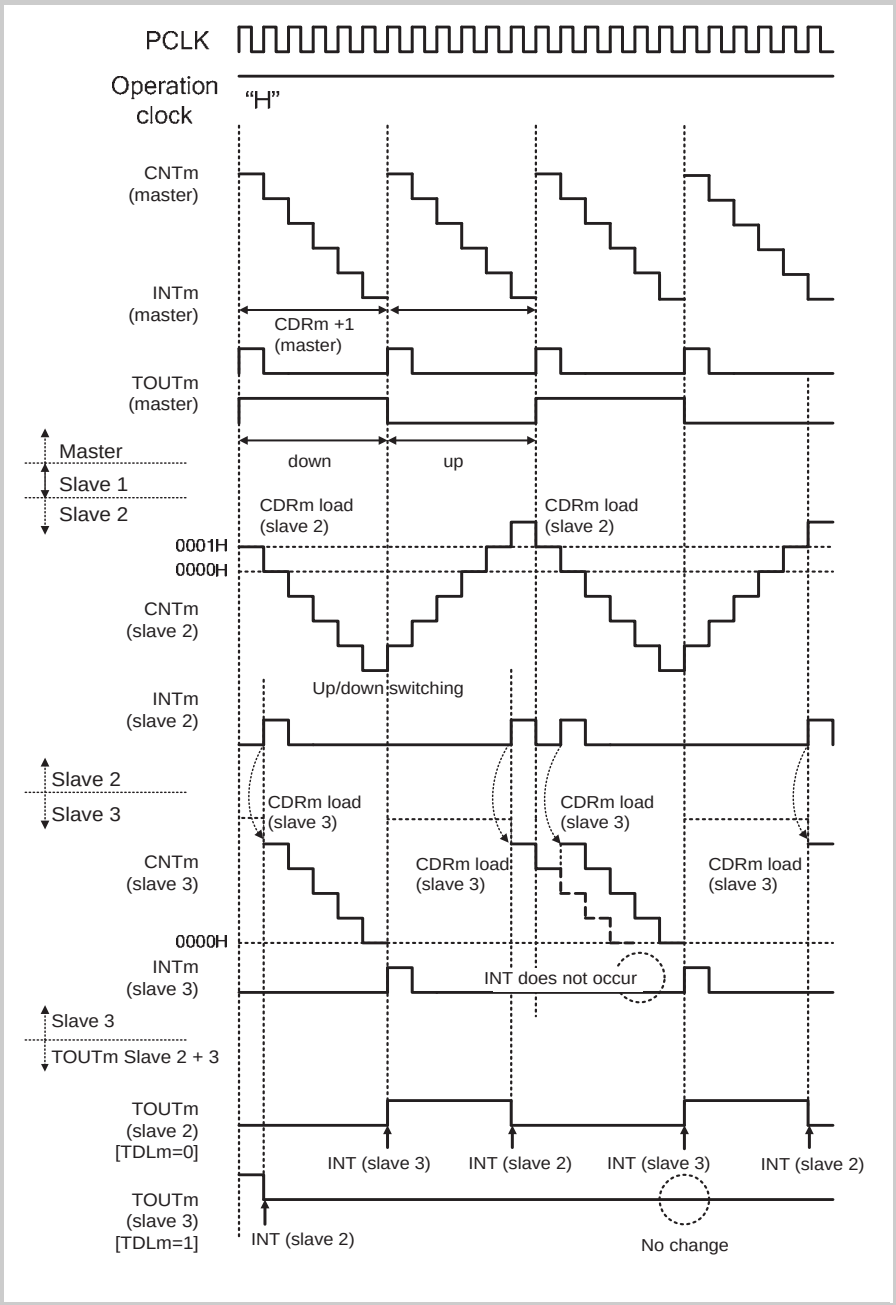



**Figure 14-94 TAUBnCDRm (master) = $0005_H$, TAUBnCDRm (slave 2) = $0002_H$, TAUBnCDRm (slave 3) = $0004_H$**
**PWM signal width (negative phase) ≥ Carrier cycle**

- After the second interrupt, a slave for which TAUBnTDL.TDLm = 1 waits for dead time to elapse before resetting. However, before the dead time has elapsed, another interrupt occurs on slave 2, this time while the counter is counting up. This acts as a set signal, meaning that a channel for which TAUBnTDL.TDLm = 1 always remains active.
- TAUBnTTOUTm of a slave channel for which TAUBnTDL.TDLm = 0 is set and reset as normal when the corresponding INTTAUBnIm is generated.

**(e) Inhibited INTTAUBnIm to set TAUBnTTOUTm positive phase period**

The following settings apply to the diagram below:

- Slave channel 2:
  - Positive logic (TAUBnTOL.TOLm = 0)
- Slave channel 3:
  - Negative logic (TAUBnTOL.TOLm = 1)

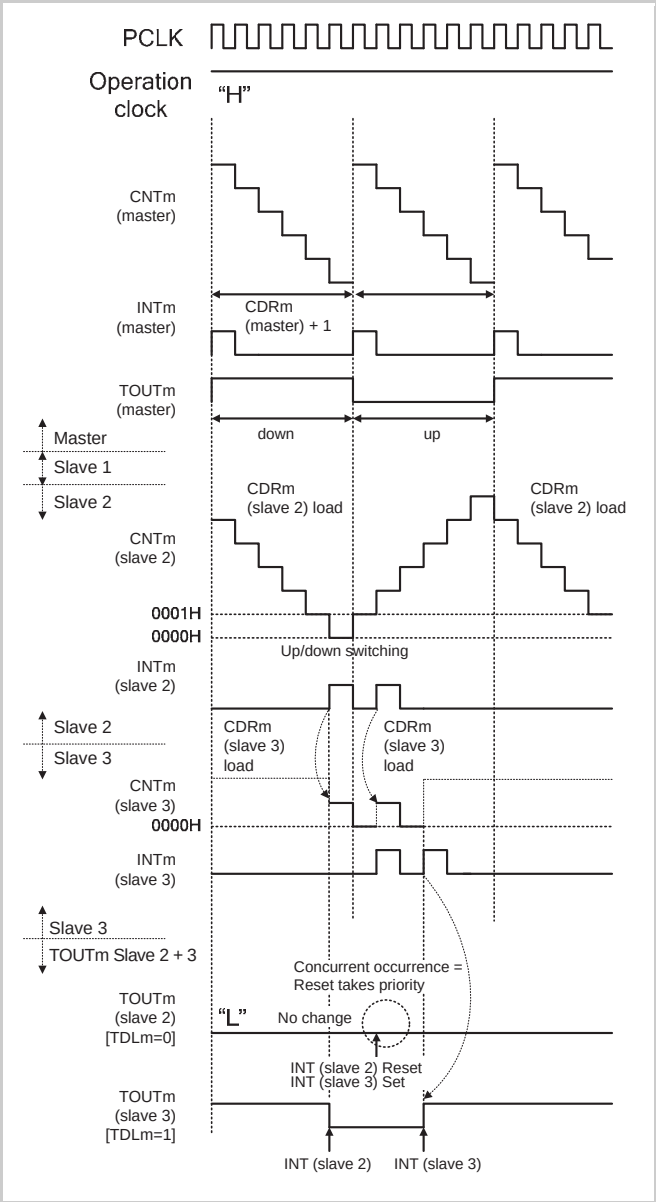



**Figure 14-95 TAUBnCDRm (master) = $0005_H$, TAUBnCDRm (slave 2) = $0005_H$, TAUBnCDRm (slave 3) = $0001_H$ PWM signal width (positive phase) = 0**

- The counter of slave channel 3 reaches $0000_H$ and generates an INTTAUBnIm to set the TAUBnTTOUTm of slave channel for which TAUBnTDL.TDLm = 0 (slave channel 2 in this example).
- If channel 2 generates an INTTAUBnIm to reset TAUBnTTOUTm simultaneously, this reset signal has priority (assuming TAUBnTOL.TOLm = 0, otherwise the set signal has priority).
- Therefore, TAUBnTTOUTm of a slave channel for which TAUBnTDL.TDLm = 0 remains in its initial state.

**(f) Inhibited INTTAUBnIm to set TAUBnTTOUTm negative phase period**

The following settings apply to the diagram below:

- Slave channel 2:
  - Positive logic (TAUBnTOL.TOLm = 0)
- Slave channel 3:
  - Negative logic (TAUBnTOL.TOLm = 1)

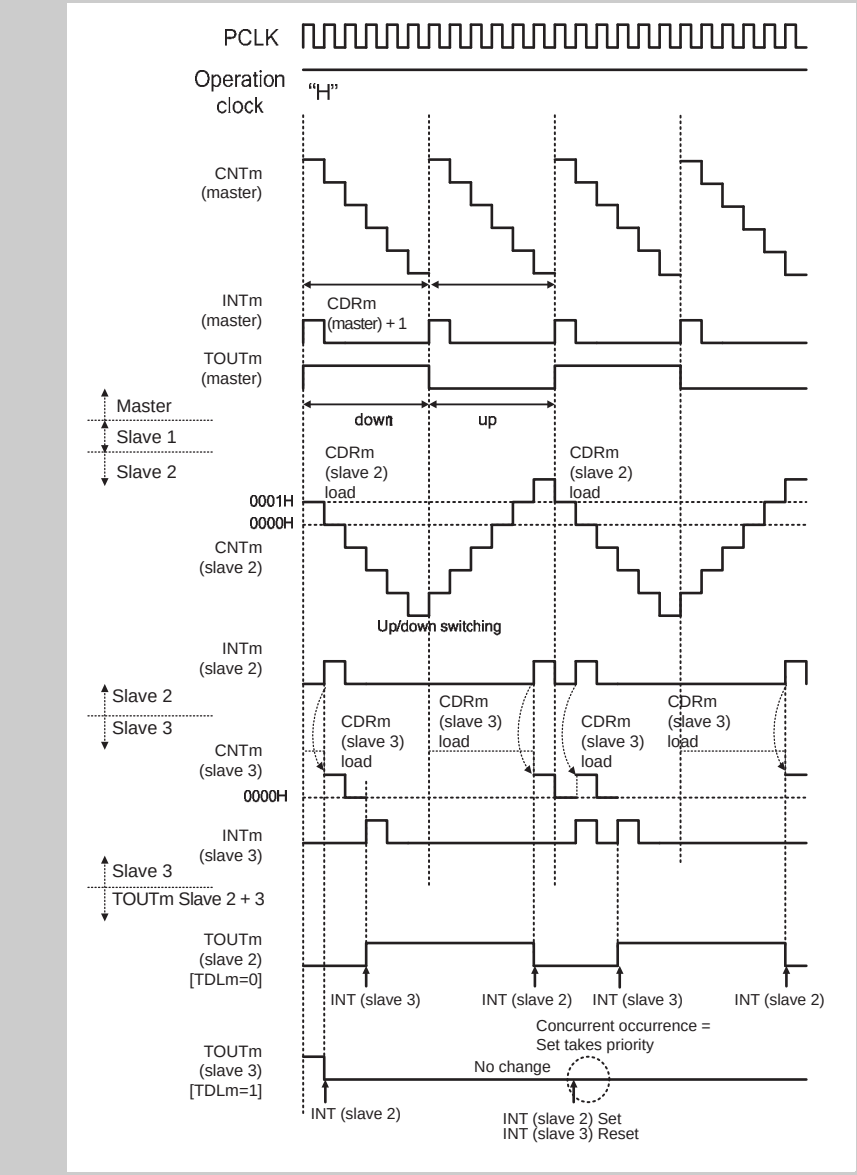



**Figure 14-96 TAUBnCDRm (master) = $0005_H$, TAUBnCDRm (slave 2) = $0001_H$, TAUBnCDRm (slave 3) = $0001_H$ PWM signal width (negative phase) = carrier cycle**

- The counter of slave channel 3 reaches $0000_H$ and generates an INTTAUBnIm to set the TAUBnTTOUTm of slave channel for which TAUBnTDL.TDLm = 1 (slave 3 in this example).
- If channel 2 generates an INTTAUBnIm to reset TAUBnTTOUTm simultaneously, the set signal has priority (assuming TAUBnTOL.TOLm = 1, otherwise the reset signal has priority).
- Therefore, TAUBnTTOUTm of slave channel for which TAUBnTDL.TDLm = 1 remains in its initial state.

## 14.21.3 AD Conversion Trigger Output Function Type 2

### (1) Overview

**Summary** This function is identical to *14.21.1 “Triangle PWM Output Function” on page 636* except that TAUBnTTOUTm is not output.

This is achieved by setting the channel output mode of the slave to Direct Channel Output Mode.

### (2) Block diagram and general timing diagram

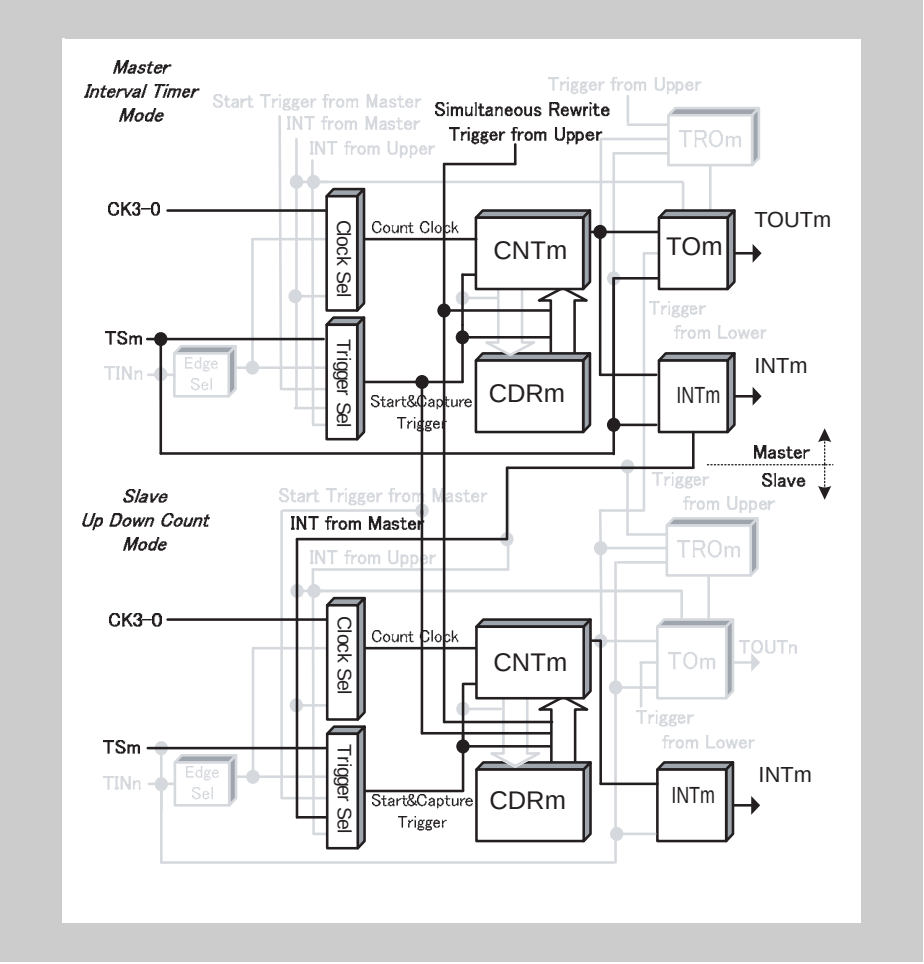



**Figure 14-97 Block diagram for AD Conversion Trigger Output Function Type 2**

The following settings apply to the general timing diagram:

- Master channel
  - INTTAUBnIm is generated at operation start (TAUBnCMORm.MD0 = 1)

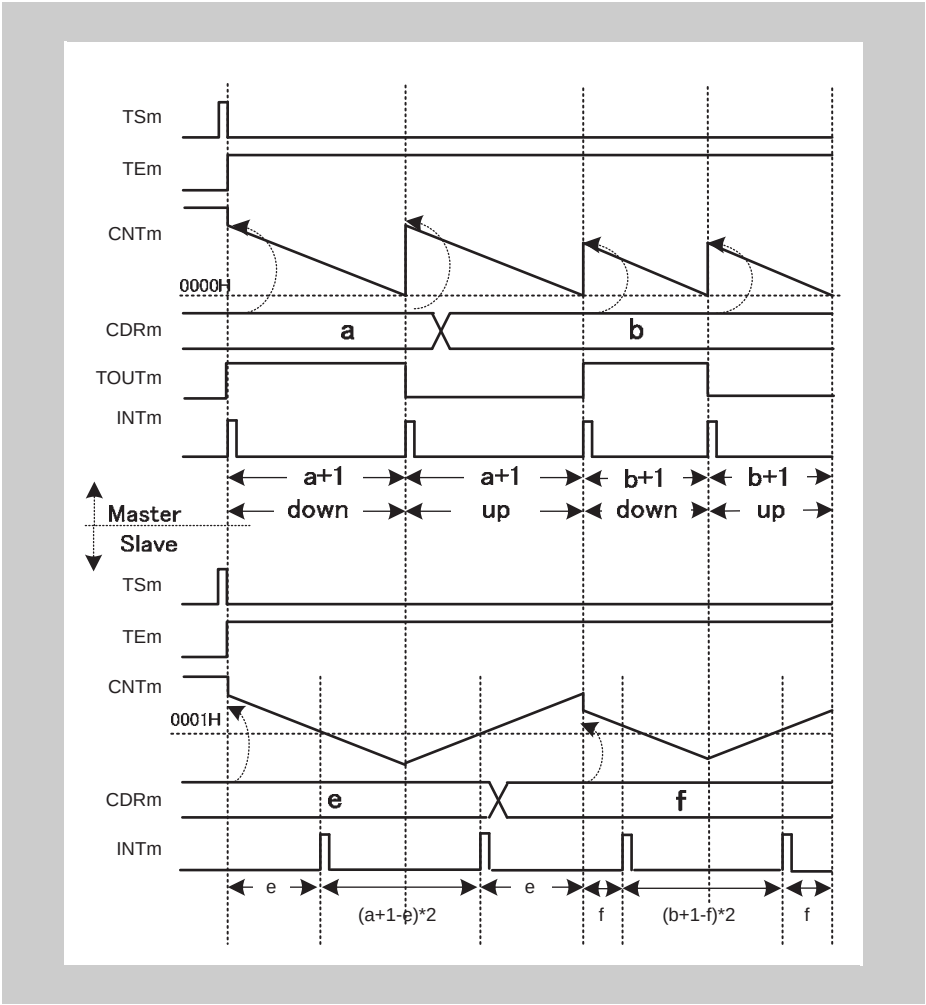



**Figure 14-98 General timing diagram for AD Conversion Trigger Output Function Type 2**

## 14.22 Registers

This section contains a description of all the registers of the 16-bit Timer Array Unit B.

### 14.22.1 TAUBn registers overview

The TAUBn is controlled and operated by the registers in the following table. Where there is one register per channel, this is indicated by an “m”, where m runs from 0 to 15.

**Table 14-135 TAUBn registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| **TAUBn prescaler registers** | | |
| TAUBn prescaler clock select register | TAUBnTPS | <TAUBn_base> + $240_H$ |
| **TAUBn control registers** | | |
| TAUBn channel data register m | TAUBnCDRm | <TAUBn_base> + m × $4_H$ |
| TAUBn channel counter register m | TAUBnCNTm | <TAUBn_base> + $80_H$ + m × $4_H$ |
| TAUBn channel mode OS register m | TAUBnCMORm | <TAUBn_base> + $200_H$ + m × $4_H$ |
| TAUBn channel mode user register m | TAUBnCMURm | <TAUBn_base> + $C0_H$ + m × $4_H$ |
| TAUBn channel status register m | TAUBnCSRm | <TAUBn_base> + $140_H$ + m × $4_H$ |
| TAUBn channel status clear trigger register m | TAUBnCSCm | <TAUBn_base> + $180_H$ + m × $4_H$ |
| TAUBn channel start trigger register | TAUBnTS | <TAUBn_base> + $1C4_H$ |
| TAUBn channel enable status register | TAUBnTE | <TAUBn_base> + $1C0_H$ |
| TAUBn channel stop trigger register | TAUBnTT | <TAUBn_base> + $1C8_H$ |
| **TAUBn output registers** | | |
| TAUBn channel output enable register | TAUBnTOE | <TAUBn_base> + $5C_H$ |
| TAUBn channel output register | TAUBnTO | <TAUBn_base> + $58_H$ |
| TAUBn channel output mode register | TAUBnTOM | <TAUBn_base> + $248_H$ |
| TAUBn channel output configuration register | TAUBnTOC | <TAUBn_base> + $24C_H$ |
| TAUBn channel output active level register | TAUBnTOL | <TAUBn_base> + $040_H$ |
| TAUBn channel dead time output enable register | TAUBnTDE | <TAUBn_base> + $250_H$ |
| TAUBn channel dead time output level register | TAUBnTDL | <TAUBn_base> + $54_H$ |
| **TAUBn reload data registers** | | |
| TAUBn channel reload data enable register | TAUBnRDE | <TAUBn_base> + $260_H$ |
| TAUBn channel reload data mode register | TAUBnRDM | <TAUBn_base> + $264_H$ |
| TAUBn channel reload data control CH select register | TAUBnRDS | <TAUBn_base> + $268_H$ |
| TAUBn channel reload data control register | TAUBnRDC | <TAUBn_base> + $26C_H$ |
| TAUBn channel reload data trigger register | TAUBnRDT | <TAUBn_base> + $44_H$ |
| TAUBn channel reload status register | TAUBnRSF | <TAUBn_base> + $48_H$ |
| **TAUBn emulation register** | | |
| TAUBn emulation register | TAUBnEMU | <TAUBn_base> + $290_H$ |

**<TAUBn_base>** The <TAUBn_base> addresses of the registers are defined in the first section of this chapter under the keyword “Register addresses”.

## 14.22.2 TAUBn prescaler registers details

### (1) TAUBnTPS - TAUBn prescaler clock select register

This register specifies the PCLK prescalers for clocks CK0, CK1, CK2, and CK3 for all channels.

**Access** This register can be read/written in 16-bit units.

**Address** <TAUBn_base> + $240_H$

**Initial Value** $FFFF_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PRS3[3:0] | | | | PRS2[3:0] | | | | PRS1[3:0] | | | | PRS0[3:0] | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-136 TAUBnTPS register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 12 | PRS3[3:0] | Specifies the CK3 clock.<br>**PRS3[3:0] / CK3 clock**<br>$0000_B$ / PCLK/$2^0$<br>$0001_B$ / PCLK/$2^1$<br>$0010_B$ / PCLK/$2^2$<br>$0011_B$ / PCLK/$2^3$<br>... / ...<br>$1110_B$ / PCLK/$2^{14}$<br>$1111_B$ / PCLK/$2^{15}$<br>These bits can only be rewritten when all counters using CK3 are stopped (TAUBnTE.TEm = 0). |
| 11 to 8 | PRS2[3:0] | Specifies the CK2 clock.<br>**PRS2[3:0] / CK2 clock**<br>$0000_B$ / PCLK/$2^0$<br>$0001_B$ / PCLK/$2^1$<br>$0010_B$ / PCLK/$2^2$<br>$0011_B$ / PCLK/$2^3$<br>... / ...<br>$1110_B$ / PCLK/$2^{14}$<br>$1111_B$ / PCLK/$2^{15}$<br>These bits can only be rewritten when all counters using CK2 are stopped (TAUBnTE.TEm = 0). |

**Table 14-136 TAUBnTPS register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>7 to 4</td><td>PRS1[3:0]</td><td>Specifies the CK1 clock.
<table>
<tr><th>PRS1[3:0]</th><th>CK1 clock</th></tr>
<tr><td>$0000_B$</td><td>PCLK/$2^0$</td></tr>
<tr><td>$0001_B$</td><td>PCLK/$2^1$</td></tr>
<tr><td>$0010_B$</td><td>PCLK/$2^2$</td></tr>
<tr><td>$0011_B$</td><td>PCLK/$2^3$</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>$1110_B$</td><td>PCLK/$2^{14}$</td></tr>
<tr><td>$1111_B$</td><td>PCLK/$2^{15}$</td></tr>
</table>
These bits can only be rewritten when all counters using CK1 are stopped (TAUBnTE.TEm = 0).</td></tr>
<tr><td>3 to 0</td><td>PRS0[3:0]</td><td>Specifies the CK0 clock.
<table>
<tr><th>PRS0[3:0]</th><th>CK0 clock</th></tr>
<tr><td>$0000_B$</td><td>PCLK/$2^0$</td></tr>
<tr><td>$0001_B$</td><td>PCLK/$2^1$</td></tr>
<tr><td>$0010_B$</td><td>PCLK/$2^2$</td></tr>
<tr><td>$0011_B$</td><td>PCLK/$2^3$</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>$1110_B$</td><td>PCLK/$2^{14}$</td></tr>
<tr><td>$1111_B$</td><td>PCLK/$2^{15}$</td></tr>
</table>
These bits can only be rewritten when all counters using CK0 are stopped (TAUBnTE.TEm = 0).</td></tr>
</table>

**Note** The TAUBn clock input PCLK is specified in the first section of this chapter under the keyword “Clock supply”.

### 14.22.3 TAUBn control registers details

#### (1) TAUBnCDRm - TAUBn channel data register

This register functions either as a compare register or as a capture register, depending on the operation mode specified in TAUBnCMORm.MD[4:0].

**Access** This register can be read/written in 16-bit units.

- In capture mode, only reading is possible. Write operation is ignored.
- In compare mode, reading and writing is possible.

**Address** <TAUBn_base> + $0_H$ + m × $4_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CDR[15:0] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-137 TAUBnCDRm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | CDR[15:0] | Data register for the capture/compare value. |

### (2) TAUBnCNTm - TAUBn channel counter register

This register is the channel m counter register.

**Access** This register can be read in 16-bit units.

**Address** <TAUBn_base> + $80_H$ + m × $4_H$

**Initial Value** $0000_H$ or $FFFF_H$ The initial value depends on the operation mode, see *Table 14-139 "TAUBnCNTm read values after the counter is re-enabled" on page 676* .

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNT[15:0] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 14-138 TAUBnCNTm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | CNT[15:0] | 16-bit counter value. |

The read value depends on the counter, the operation mode change, and the values of the TAUBnTS.TSm and TAUBnTT.TTm bits.

The *initial* counter read value depends on the operation mode and how the counter was stopped:

- by a reset
- by a counter stop trigger (TAUBnTT.TTm = 1)

The following table lists the initial counter read values after the counter has stopped (TAUBnTE.TEm= 0 ) and re-enabled (TAUBnTS.TSm = 1).

The table also contains the counter read value one count after the counter is enabled (TAUBnTS.TSm = 1) for modes where the counter waits for a start trigger.

**Table 14-139 TAUBnCNTm read values after the counter is re-enabled**

| Mode name | Count method (up/down) | TAUBnCNTm value | | |
|---|---|---|---|---|
| | | After reset | After stop trigger | After one count |
| Interval Timer mode | Count down | $FFFF_H$ | Stop value | — |
| Judge mode | Count down | $FFFF_H$ | Stop value | — |
| Capture mode | Count up | $0000_H$ | Stop value | — |
| Event Count mode | Count down | $FFFF_H$ | Stop value | — |
| One Count mode | Count down | $FFFF_H$ | Stop value | $FFFF_H$ |
| Capture & One Count mode | Count up | $0000_H$ | Stop value | Captured value + 1 (TAUBnCDRm) |
| Judge & One Count mode | Count down | $FFFF_H$ | Stop value | TAUBnCNTm value - 1 |
| Up Down Count mode | Count up/down | $FFFF_H$ | Stop value | — |
| Pulse One Count mode | Count down | $FFFF_H$ | Stop value | $0000_H$ |
| Count Capture Mode | Count up | $0000_H$ | Stop value | — |
| Gate Count Mode | Count down | $FFFF_H$ | Stop value | Stop value |
| Capture & Gate Count Mode | Count up | $0000_H$ | Stop value | Stop value |

**Note** If the operation mode is changed while the counter is stopped, the initial counter value after counter restart is undefined. The operation mode is changed by register TAUBnCMORm.MD[4:0].

### (3) TAUBnCMORm - TAUBn channel mode OS register

This register controls channel m operation. It specifies, for example, the operation clock, count clock, and master/slave function.

**Access** This register can be read/written in 16-bit units. It can only be written when the counter is stopped (TAUBnTE.TEm = 0).

**Address** <TAUBn_base> + $200_H$ + m × $4_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | — | CCS0 | MAS | STS[2:0] | | | COS[1:0] | | — | MD[4:0] | | | | |
| R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W |

**Table 14-140 TAUBnCMORm register contents (1/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 15,14 | CKS[1:0] | Selects the operation clock.<br>The operation clock is used for the TAUBnTTINm input edge detection circuit. It can also be used as the count clock depending on bit TAUBnCMORm.CCS0. |
| 12 | CCS0 | Selects the count clock for TAUBnCNTm counter:<br>0: Operation clock as specified by TAUBnCMORm.CKS[1:0]<br>1: Valid edge of TAUBnTTINm input signal |
| 11 | MAS | Specifies the channel as master or slave channel during synchronous channel operation:<br>0: Slave<br>1: Master<br>This bit is only valid for even channels (CHm_even). For odd channels (CHm_odd), it is fixed to 0. |

| CKS1 | CKS0 | Selected operation clock |
|---|---|---|
| 0 | 0 | CK0 |
| 0 | 1 | CK1 |
| 1 | 0 | CK2 |
| 1 | 1 | CK3 |

**Table 14-140 TAUBnCMORm register contents (2/3)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>10 to 8</td><td>STS[2:0]</td><td>Selects the external start trigger:
<table>
<tr><th>STS2</th><th>STS1</th><th>STS0</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Software trigger</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Valid edge of the TAUBnTTINm input signal. TAUBnCMURm.TIS[1:0] specifies the valid edge.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Valid edge of the TAUBnTTINm input signal is the start trigger and the reverse edge is the stop (capture) trigger</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Setting prohibited</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>INTTAUBnI of the master channel</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>INTTAUBnI of the upper channel (m-1), regardless of the master setting</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Dead-time output signal TAUBnTTDL of the TAUBnTTOUTm generation unit</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Up/down output trigger signal TAUBnTUDSm of the master channel.</td></tr>
</table></td></tr>
<tr><td>7, 6</td><td>COS[1:0]</td><td>Specifies when the capture register TAUBnCDRm and the overflow flag TAUBnCSRm.OVF of channel m are updated.<br>These bits are only valid if channel m is in capture mode.
<table>
<tr><th>COS1</th><th>COS0</th><th>Capture register</th><th>TAUBnCSRm.OVF</th></tr>
<tr><td>0</td><td>0</td><td rowspan="2">Updated upon detection of a TAUBnTTINm input valid edge.</td><td>Updated (cleared or set) upon detection of a TAUBnTTINm input valid edge:<br>• If a counter overflow has occurred since the last valid edge detection, TAUBnCSRm.OVF is set.<br>• If no counter overflow has occurred since the last valid edge detection, TAUBnCSR.OVF is cleared.</td></tr>
<tr><td>0</td><td>1</td><td>Set upon counter overflow and cleared by a CPU instruction.</td></tr>
<tr><td>1</td><td>0</td><td rowspan="2">Updated upon detection of a TAUBnTTINm input valid edge and upon counter overflow:<br>• TAUBnTTINm input valid edge: Counter value is written to TAUBnCDRm<br>• Overflow: $FFFF_H$ is written to TAUBnCDRm. The next TAUBnTTINm input valid edge detection is ignored.</td><td>Not set.</td></tr>
<tr><td>1</td><td>1</td><td>Set upon counter overflow and cleared by a CPU instruction.</td></tr>
</table></td></tr>
</table>

**Table 14-140 TAUBnCMORm register contents (3/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 4 to 0 | MD[4:0] | Specifies the operation mode. |

| MD4 | MD3 | MD2 | MD1 | MD0 | Description |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1/0 | Interval Timer mode |
| 0 | 0 | 0 | 1 | 1/0 | Judge mode |
| 0 | 0 | 1 | 0 | 1/0 | Capture mode |
| 0 | 0 | 1 | 1 | 0 | Event Count mode |
| 0 | 1 | 0 | 0 | 1/0 | One Count mode |
| 0 | 1 | 0 | 1 | 1/0 | Setting prohibited |
| 0 | 1 | 1 | 0 | 0 | Capture & One Count mode |
| 0 | 1 | 1 | 1 | 1/0 | Judge & One Count mode |
| 1 | 0 | 0 | 0 | 0 | Setting prohibited |
| 1 | 0 | 0 | 1 | 0 | Up Down Count mode |
| 1 | 0 | 1 | 0 | 1/0 | Pulse One Count mode |
| 1 | 0 | 1 | 1 | 1/0 | Count Capture mode |
| 1 | 1 | 0 | 0 | 0 | Gate Count mode |
| 1 | 1 | 0 | 1 | 0 | Capture & Gate Count mode |

| Mode | Role of the MD0 bit |
|---|---|
| Interval Timer mode<br>Capture mode<br>Count Capture mode | Specifies whether an INTTAUBnIm is generated when the counter is triggered:<br>0: No INTTAUBnIm generated<br>1: INTTAUBnIm generated |
| Event Count mode<br>Up Down Count mode | This bit must be set to 0:<br>0: No INTTAUBnIm generated when the counter is triggered<br>1: — |
| One Count mode<br>Gate Count mode | Enables/disables start trigger detection during counting:<br>0: Disabled<br>1: Enabled<br>INTTAUBnIm and TAUBnTTOUTm are not output when the counter is triggered. |
| Pulse One Count mode | Enables/disables start trigger detection during counting:<br>0: Disabled<br>1: Enabled<br>INTTAUBnIm and TAUBnTTOUTm are output when the counter is triggered. |
| Capture & One Count mode<br>Capture & Gate Count mode | This bit must be set to 0:<br>0: No INTTAUBnIm generated when the counter is triggered. Start trigger is disabled during counting.<br>1: — |
| Judge mode<br>Judge One Count mode | Specifies when INTTAUBnIm is generated:<br>0: When TAUBnCNTm ≤ TAUBnCDRm<br>1: When TAUBnCNTm > TAUBnCDRm |

### (4) TAUBnCMURm - TAUBn channel mode user register

This register specifies the type of valid edge detection used for the TAUBnTTINm input.

**Access** This register can be read/written in 8-bit units.

**Address** <TAUBn_base> + $C0_H$ + m × $4_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | TIS[1:0] | |
| R | R | R | R | R | R | R/W | R/W |

**Table 14-141 TAUBnCMURm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1, 0 | TIS[1:0] | Specifies the valid edge of the TAUBnTTINm input:<br>• To detect rising and falling edges when TAUBnCMORm.STS[2:0] is not set to $010_B$, set TAUBnCMURm.TIS[1:0] = $10_B$.<br>• Edge detection for TAUBnTTINm input signals is performed based on the operation clock selected by TAUBnCMORm.CKS[1:0]. |

| TIS1 | TIS0 | Description |
|---|---|---|
| 0 | 0 | Falling edge |
| 0 | 1 | Rising edge |
| 1 | 0 | Rising and falling edges (low-width measurement selection).<br>Start trigger: falling edge<br>Stop trigger (capture): rising edge |
| 1 | 1 | Rising and falling edges (high-width measurement selection).<br>Start trigger: rising edge<br>Stop trigger (capture): falling edge<br>This setting is only valid if the start trigger selection is set to TAUBnCMORm.STS[2:0] = $010_B$ |

### (5) TAUBnCSRm - TAUBn channel status register

This register indicates the count direction and the overflow status of channel m's counter.

**Access** This register can be read in 8-bit units.

**Address** <TAUBn_base> + $140_H$ + m × $4_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | CSF | OVF |
| R | R | R | R | R | R | R | R |

**Table 14-142 TAUBnCSRm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | CSF | Indicates the count direction:<br>0: Counts up<br>1: Counts down<br>The read value of this bit is only valid in the following mode:<br>• Up Down Count mode<br>For channel 0 this bit is fixed to 0. |
| 0 | OVF | Indicates the counter overflow status:<br>0: No overflow occurred<br>1: Overflow occurred<br>This bit is only used in the following modes:<br>• Capture mode<br>• Capture & One Count mode<br>• Count Capture mode<br>• Capture & Gate Count mode<br><br>The function of this bit depends on the setting of control bits TAUBnCMORm.COS[1:0].<br>OVF is not be set when TAUBnCMORm.COS[1:0] = $10_B$. |

### (6) TAUBnCSCm - TAUBn channel status clear register

This register is a trigger register for clearing the overflow flag TAUBnCSRm.OVF of a channel m.

**Access** This register can be written in 8-bit units. It is always read as $0000_H$.

**Address** <TAUBn_base> + $180_H$ + m × $4_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | 0 | CLOV |
| R | R | R | R | R | R | R | W |

**Table 14-143 TAUBnCSCm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | CLOV | 0: No function<br>1: Clears the overflow flag TAUBnCSRm.OVF |

### (7) TAUBnTS - TAUBn channel start trigger register

This register enables the counter for each channel.

**Access** This register can be written in 16-bit units. It is always read as $0000_H$.

**Address** <TAUBn_base> + $1C4_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TS 15 | TS 14 | TS 13 | TS 12 | TS 11 | TS 10 | TS 09 | TS 08 | TS 07 | TS 06 | TS 05 | TS 04 | TS 03 | TS 02 | TS 01 | TS 00 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 14-144 TAUBnTS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TSm | Enables the counter for channel m:<br>0: No function<br>1: Enables the counter and sets TAUBnTE.TEm = 1. When the counter is enabled, this bit immediately returns to 0.<br>TAUBnTE.TEm = 1 only *enables* counter. Whether the counter *starts* depends on the selected operation mode. |

### (8) TAUBnTE - TAUBn channel enable status register

This register indicates whether counter is enabled or disabled.

**Access** This register can be read in 16-bit units.

**Address** <TAUBn_base> + $1C0_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TE15 | TE14 | TE13 | TE12 | TE11 | TE10 | TE09 | TE08 | TE07 | TE06 | TE05 | TE04 | TE03 | TE02 | TE01 | TE00 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 14-145 TAUBnTE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TEm | Indicates whether counter for channel m is enabled or disabled:<br>0: Counter disabled<br>1: Counter enabled<br>Setting TAUBnTS.TSm to 1 or trigger input detection TAUBnTSSTm = 1 sets this bit to 1.<br>Setting TAUBnTT.TTm to 1 resets this bit to 0. |

### (9) TAUBnTT - TAUBn channel stop trigger register

This register stops the counter for each channel.

**Access** This register can be written in 16-bit units. It is always read as $0000_H$.

**Address** <TAUBn_base> + $1C8_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TT15 | TT14 | TT13 | TT12 | TT11 | TT10 | TT09 | TT08 | TT07 | TT06 | TT05 | TT04 | TT03 | TT02 | TT01 | TT00 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 14-146 TAUBnTT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TTm | Stops the counter of channel m:<br>0: No function<br>1: Stops the counter and sets TAUBnTE.TEm = 0. When the counter has stopped, this bit immediately returns to 0.<br>TAUBnCNTm stops counting and TAUBnCNTm, TAUBnTO.TOm, and TAUBnTTOUTm all retain the values they had before the counter was stopped. |

## 14.22.4 TAUBn output registers details

### (1) TAUBnTOE - TAUBn channel output enable register

This register enables and disables Direct Channel Output Mode.

**Access** This register can be read/written in 16-bit units.

**Address** <TAUBn_base> + $5C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TOE 15 | TOE 14 | TOE 13 | TOE 12 | TOE 11 | TOE 10 | TOE 09 | TOE 08 | TOE 07 | TOE 06 | TOE 05 | TOE 04 | TOE 03 | TOE 02 | TOE 01 | TOE 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-147 TAUBnTOE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TOEm | Enables/disables Direct Channel Output Mode:<br>0: Enables Direct Channel Output Mode (TAUBnTTOUT m output is controlled by the application software)<br>1: Disables Direct Channel Output Mode (TAUBnTTOUT m output is controlled by the timer) |

### (2) TAUBnTOM - TAUBn channel output mode register

This register specifies the output mode of each channel.

**Access** This register can be read/written in 16-bit units. It can only be written when the counter is stopped (TAUBnTE.TEm = 0).

**Address** <TAUBn_base> + $248_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TOM 15 | TOM 14 | TOM 13 | TOM 12 | TOM 11 | TOM 10 | TOM 09 | TOM 08 | TOM 07 | TOM 06 | TOM 05 | TOM 04 | TOM 03 | TOM 02 | TOM 01 | TOM 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-148 TAUBnTOM register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TOMm | Specifies the timer controlled channel output mode, if Direct Channel Output Mode is disabled (TAUBnTOE.TOEm = 1):<br>0: Independent Channel Output Mode<br>1: Synchronous Channel Output Mode<br>The output mode depends on several channel output control bits, as can be seen in *Table 14-16 “Channel output modes” on page 490* . |

### (3) TAUBnTOC - TAUBn channel output configuration register

This register specifies the output mode of each channel in combination with TAUBnTOMm.

**Access** This register can be read/written in 16-bit units. It can only be written when the counter is stopped (TAUBnTE.TEm = 0).

**Address** <TAUBn_base> + $24C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TOC15 | TOC14 | TOC13 | TOC12 | TOC11 | TOC10 | TOC09 | TOC08 | TOC07 | TOC06 | TOC05 | TOC04 | TOC03 | TOC02 | TOC01 | TOC00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-149 TAUBnTOC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TOCm | Specifies the output mode:<br>0: Operation mode 1<br>1: Operation mode 2<br>The output mode also depends on TAUBnTOM.TOMm, as can be seen in the following table. |

| TOMm | TOCm | Description |
|---|---|---|
| 0 | 0 | Toggle mode: TAUBnTTOUTm toggles when INTTAUBnIm occurs. |
| | 1 | Set/reset mode: TAUBnTTOUTm set when INTTAUBnIm occurs upon count start and reset when INTTAUBnIm occurs due to detection of a match between TAUBnCNTm and TAUBnCDRm, or, in One-Shot Pulse Output Function when the counter is 0. |
| 1 | 0 | Synchronous Channel Operation Mode 1: TAUBnTTOUTm set when INTTAUBnI occurs on the master channel and reset when INTTAUBnI occurs on the slave channel. |
| | 1 | Synchronous Channel Operation Mode 2: TAUBnTTOUTm set when INTTAUBnIm occurs while the slave channel is counting down and reset when INTTAUBnIm occurs while the slave channel is counting up. |

### (4) TAUBnTDE - TAUBn channel dead time output enable register

This register enables/disables dead time operation for each channel.

**Access** This register can be read/written in 16-bit units. It can only be written when the counter is stopped (TAUBnTE.TEm = 0).

**Address** <TAUBn_base> + $250_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TDE 15 | TDE 14 | TDE 13 | TDE 12 | TDE 11 | TDE 10 | TDE 09 | TDE 08 | TDE 07 | TDE 06 | TDE 05 | TDE 04 | TDE 03 | TDE 02 | TDE 01 | TDE 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-150 TAUBnTDE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TDEm | Enables/disables dead time control operation of channel m:<br>0: Disables dead time operation<br>1: Enables dead time operation<br>The same settings must be set for the even and the odd slave channel that comprise a set.<br>These bits only apply when:<br>• TAUBnTOE.TOEm, TAUBnTOM.TOMm, and TAUBnTOC.TOCm = 1. |

### (5) TAUBnTDL - TAUBn channel dead time output level register

This register selects the phase period to which dead time is added.

**Access** This register can be read/written in 16-bit units.

**Address** <TAUBn_base> + $54_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TDL 15 | TDL 14 | TDL 13 | TDL 12 | TDL 11 | TDL 10 | TDL 09 | TDL 08 | TDL 07 | TDL 06 | TDL 05 | TDL 04 | TDL 03 | TDL 02 | TDL 01 | TDL 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-151 TAUBnTDL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TDLm | Selects the phase period to which dead time is added:<br>0: Positive phase period<br>1: Negative phase period<br>These bits only apply when:<br>• TAUBnTOE.TOEm, TAUBnTOM.TOMm, TAUBnTOC.TOCm, and TAUBnTDE.TDEm = 1. |

## 14.22.5 TAUBn channel output level registers details

### (1) TAUBnTO - TAUBn channel output register

This register specifies and reads the level of TAUBnTTOUTm.

**Access** This register can be read/written in 16-bit units.

**Address** <TAUBn_base> + $58_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TO15 | TO14 | TO13 | TO12 | TO11 | TO10 | TO09 | TO08 | TO07 | TO06 | TO05 | TO04 | TO03 | TO02 | TO01 | TO00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-152 TAUBnTO register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TOm | Specifies/reads the level of TAUBnTTOUTm:<br>0: Low<br>1: High<br>Only TOm bits for which Independent Channel Output function is disabled (TAUBnTOEm = 0) can be written. |

### (2) TAUBnTOL - TAUBn channel output level register

This register specifies the output logic of the channel output bit (TAUBnTO.TOm).

**Access** This register can be read/written in 16-bit units.

**Address** <TAUBn_base> + $040_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TOL 15 | TOL 14 | TOL 13 | TOL 12 | TOL 11 | TOL 10 | TOL 09 | TOL 08 | TOL 07 | TOL 06 | TOL 05 | TOL 04 | TOL 03 | TOL 02 | TOL 01 | TOL 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-153 TAUBnTOL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | TOLm | Specifies the output logic of the channel m output bit (TAUBnTO.TOm):<br>0: Positive logic (active high)<br>1: Inverted logic (active low)<br>These bits apply in all channel output modes except Direct Channel Output Mode and channel output modes with real-time output. |

## 14.22.6 TAUBn simultaneous rewrite register details

### (1) TAUBnRDE - TAUBn channel reload data enable register

This register enables and disables simultaneous rewrite of the data register TAUBnCDRm. It also enables simultaneous rewrite of the data register TAUBnTOLm for the PWM output function and the triangle PWM output function.

**Access** This register can be read/written in 16-bit units. It can only be written when TAUBnTE.TEm = 0.

**Address** <TAUBn_base> + $260_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RDE 15 | RDE 14 | RDE 13 | RDE 12 | RDE 11 | RDE 10 | RDE 09 | RDE 08 | RDE 07 | RDE 06 | RDE 05 | RDE 04 | RDE 03 | RDE 02 | RDE 01 | RDE 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-154 TAUBnRDE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | RDEm | Enables/disables simultaneous rewrite of the data register of channel m:<br>0: Disables simultaneous rewrite<br>1: Enabled simultaneous rewrite |

### (2) TAUBnRDM - TAUBn channel reload data mode register

This register selects when the signal that controls simultaneous rewrite is loaded.

**Access** This register can be read/written in 16-bit units. It can only be written when TAUBnTE.TEm = 0.

**Address** <TAUBn_base> + $264_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RDM 15 | RDM 14 | RDM 13 | RDM 12 | RDM 11 | RDM 10 | RDM 09 | RDM 08 | RDM 07 | RDM 06 | RDM 05 | RDM 04 | RDM 03 | RDM 02 | RDM 01 | RDM 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-155 TAUBnRDM register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | RDMm | Selects when the signal that triggers simultaneous is generated:<br>0: When the master channel counter starts counting<br>1: At the top of a triangle wave cycle<br>These bits only apply when TAUBnRDE.RDEm = 1 and TAUBnRDS.RDSm = 0. |

### (3) TAUBnRDS - TAUBn channel reload data control channel select register

This register selects the control channel for simultaneous rewrite.

**Access** This register can be read/written in 16-bit or 1-bit units. It can only be written when TAUBnTE.TEm = 0.

**Address** <TAUBn_base> + $268_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RDS 15 | RDS 14 | RDS 13 | RDS 12 | RDS 11 | RDS 10 | RDE 09 | RDE 08 | RDE 07 | RDE 06 | RDE 05 | RDE 04 | RDE 03 | RDE 02 | RDE 01 | RDE 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-156 TAUBnRDS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | RDSm | Specifies which channel is monitored for the simultaneous rewrite trigger:<br>0: Master channel<br>1: Another upper channel |

### (4) TAUBnRDC - TAUBn channel reload data control register

This register specifies the channel that generates the INTTAUBnIm signal that triggers simultaneous rewrite.

**Access** This register can be read/written in 16-bit units. It can only be written when TAUBnTE.TEm = 0

**Address** <TAUBn_base> + $26C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RDC 15 | RDC 14 | RDC 13 | RDC 12 | RDC 11 | RDC 10 | RDC 09 | RDC 08 | RDC 07 | RDC 06 | RDC 05 | RDC 04 | RDC 03 | RDC 02 | RDC 01 | RDC 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-157 TAUBnRDC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | RDCm | Specifies whether the channel is monitored for an INTTAUBnIm signal that is used as the simultaneous rewrite trigger:<br>0: Channel is not monitored<br>1: Channel is monitored<br>These bits only apply when TAUBnRDS.RDSm = 1. |

### (5) TAUBnRDT - TAUBn channel reload data trigger register

This register triggers the simultaneous rewrite pending state.

**Access** This register can be written in 16-bit units. It is always read as $0000_H$.

**Address** <TAUBn_base> + $044_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RDT 15 | RDT 14 | RDT 13 | RDT 12 | RDT 11 | RDT 10 | RDT 09 | RDT 08 | RDT 07 | RDT 06 | RDT 05 | RDT 04 | RDT 03 | RDT 02 | RDT 01 | RDT 00 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 14-158 TAUBnRDT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | RDTm | Triggers the simultaneous rewrite pending state:<br>0: No function<br>1: Simultaneous rewrite pending state is triggered. The simultaneous rewrite pending flag (TAUBnRSFm) is set to 1. The system waits for the simultaneous rewrite trigger. TAUBnRDT.RDTm immediately returns to 0. |

### (6) TAUBnRSF - TAUBn channel reload status register

This flag register indicates that simultaneous rewrite is possible.

**Access** This register can be read in 16-bit units.

**Address** <TAUBn_base> + $048_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RSF 15 | RSF 14 | RSF 13 | RSF 12 | RSF 11 | RSF 10 | RSF 09 | RSF 08 | RSF 07 | RSF 06 | RSF 05 | RSF 04 | RSF 03 | RSF 02 | RSF 01 | RSF 00 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 14-159 TAUBnRSF register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | RSFm | Indicates the simultaneous rewrite status:<br>0: Simultaneous rewrite disabled<br>1: Simultaneous rewrite enabled |

## 14.22.7 TAUBn emulation register

### (1) TAUBnEMU - TAUB emulation register

This register controls whether the TAUBn can be stopped during emulation, for instance upon a breakpoint hit.

**Access** This register can be read/written in 8-bit units only when TAUBnTE.TEm = 0.

**Address** <TAUBn_base> + $290_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TAUBn SVSDIS | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 14-160 TAUBnEMU register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | TAUBn SVSDIS | Emulation control<br>0: TAUBn can be stopped during emulation<br>1: TAUBn continuous operating during emulation |

# Chapter 15 Timer Array Unit J (TAUJ)

This chapter contains a generic description of the Timer Array Unit J (TAUJ).

The first section describes all V850E2/Fx4-G specific properties, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 15.1 V850E2/Fx4-G TAUJ Features

**Instances**

This microcontroller has following number of instances of the Timer Array Unit J.

**Table 15-1 Instances of TAUJ**

| Timer Array Unit J | |
|---|---|
| Instance | 1 |
| Name | TAUJ0 |

**Instances index n**

Throughout this chapter, the individual instances of a Timer Array Unit J is identified by the index “n” (n = 0), for example, TAUJnTOM for the TAUJn channel output mode register.

**Channel index m**

The Timer Array Unit J has 4 channels. Throughout this chapter, the individual channels are identified by the index “m” (m = 0 to 3), thus a certain channel is denoted as CHm.
The even numbered channels (m = 0, 2) are denoted as CHm_even.
The odd numbered channels (m = 1, 3) are denoted as CHm_odd.

**Register addresses**

All TAUJ n register addresses are given as address offsets to the individual base address <TAUJ n_base>.
The <TAUJ n_base> address of each TAUJn are listed in the following table:

**Table 15-2 Register base addresses <TAUJn_base>**

| TAUJn instance | <TAUJn_base> address |
|---|---|
| TAUJ 0 | FF81 $1000_H$ |

**Clock supply** All Timer Array Units J provide one clock input.

**Table 15-3 TAUJn clock supply**

| TAUJn instance | TAUJn clock | Connected to |
|---|---|---|
| TAUJ0 | PCLK | Clock Controller CKSCLK_A03 |

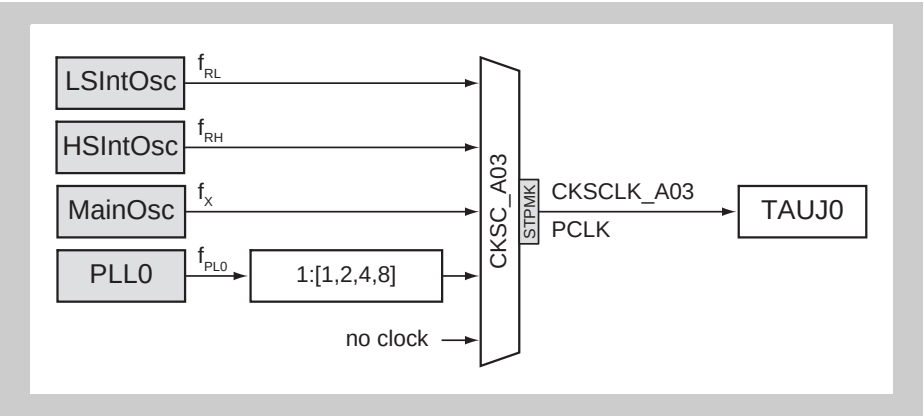



**Figure 15-1 TAUJ clock supply**

**Interrupts and DMA** The Time Array Unit J can generaste the following interrupt and DMA requests:

**Table 15-4 TAUJn interrupt and DMA requests**

| TAUJn signals | Function | Connected to |
|---|---|---|
| **TAUJ0:** | | |
| INTTAUJ0I0 | Channel 0 interrupt | Interrupt Controller INTTAUJ0I0[a]<br>DMA Controller trigger 44<br>Wake-up Sequencer INTTAUJI0 |
| INTTAUJ0I1 | Channel 1 interrupt | Interrupt Controller INTTAUJ0I1[a]<br>DMA Controller trigger 45<br>Wake-up Sequencer INTTAUJI1 |
| INTTAUJ0I2 | Channel 2 interrupt | Interrupt Controller INTTAUJ0I2[a]<br>DMA Controller trigger 46<br>Wake-up Sequencer INTTAUJI2 |
| INTTAUJ0I3 | Channel 3 interrupt | Interrupt Controller INTTAUJ0I3[ab]<br>DMA Controller trigger 47<br>TAUB0 TAUB0TTIN0,<br>TAUB0TTIN15[c] |

a) These interrupts can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

b) These signals can be used to as a trigger source to start the A/D Converter. Refer to the section *"H/W Trigger Expansion"* in the chapter *"A/D Converter (ADAA)"*.

c) Refer to section *"TAUB Input Selections"* in the *"Timer Array Unit B (TAUB)"* for details.

**TAUJ H/W reset**

The Time Array Units J and their registers are initialized by the following reset signal:

**Table 15-5 TAUJn reset signal**

| TAUJn | Reset signal |
|---|---|
| TAUJn | • Reset Controller SYSRES |

**I/O signals**

The I/O signals of the Timer Array Unit J are listed in the following table.

**Table 15-6 TAUJn I/O signals**

| TAUJ signal | Function | Connected to |
|---|---|---|
| **TAUJ0:** | | |
| TAUJ0TTIN0 to TAUJ0TTIN3 | Channel 0 to 3 input | Port TAUJ0I0[a] to TAUJ0I3[a] |
| TAUJ0TTOUT0 to TAUJ0TTOUT3 | Channel 0 to 3 output | Port TAUJ0O0 to TAUJ0O3 |

a) These input signals are passed through a noise filter, refer to the section *"Port Filters"* in the chapter *"Port Functions"*.

## 15.2 Functional Overview

**Features summary**

The TAUJ has the following functions:

- 4 channels
- 32-bit counter and 32-bit data register per channel
- Independent channel operation
- Synchronous channel operation (master and slave operation)
- Generation of different types of output signal
- Counter can be triggered by external signal
- Interrupt generation

The following figure shows the main components of the TAUJ:

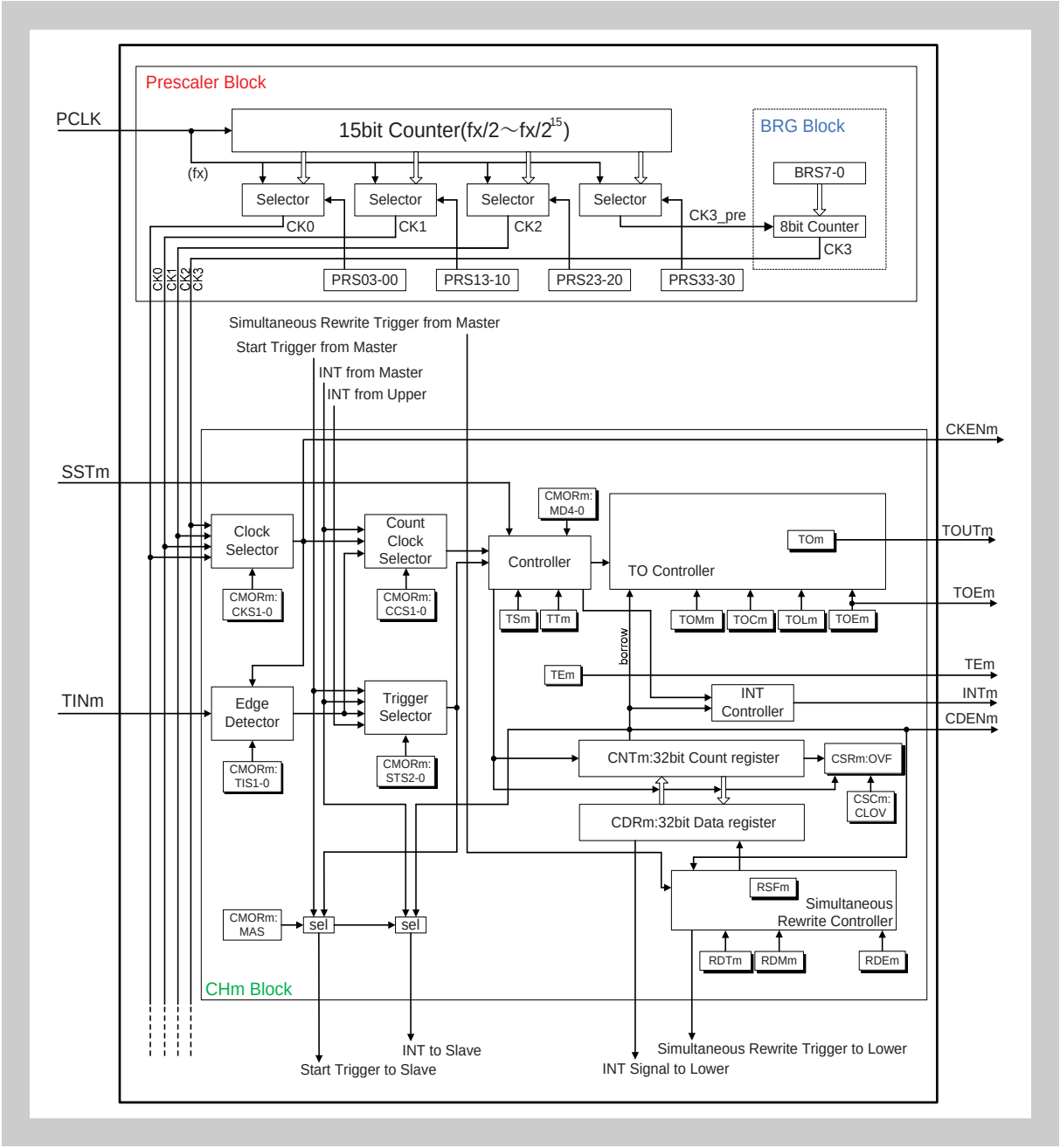



**Figure 15-2 Block diagram of the TAUJ**

The prefix “TAUJn” has been omitted from the register names for the sake of clarity in the above figure.

### 15.2.1 Terms

In this chapter, the following terms are used:

- **Independent / synchronous channel operation**

  Independent or synchronous channel operation describes the dependency of channels on each other:

  – If a channel operates independent of all other channels, this is called independent channel operation.

  – If a channel operates depending on other channels, this is called synchronous channel operation.

- **Channel group**

  In synchronous channel operation, all channels that depend on each other are referred to as a “channel group”.

  A channel group has one master channel and one or more slave channels.

- **Operation mode**

  An operation mode can be selected for every channel m. The operation mode defines the *basic* operation and features of a channel.

  In synchronous channel operation, every channel in the channel group can operate in a different operation mode.

  Examples are “Capture Mode”, and “Interval Timer Mode”.

- **Channel output mode**

  The channel output mode defines the operation of TAUJnTTOUTm

  – of a single channel (independent output operation) or

  – of all channels in a channel group (synchronous output operation).

  Example: “Independent Channel Output Mode 1”

- **Channel operation function**

  The channel operation function defines the *complete* function and all features

  – of a single channel (independent channel operation) or

  – of all channels in a channel group (synchronous channel operation).

  It defines the operation mode, start and capture trigger, count clock, and the channel output mode.

  Examples are “Interval Timer Function”, “TINm Input Position Detection Function”, and “TINm Input Period Count Detection Function”

- **Upper / lower channel**

  Depending on the channel number m, a neighboring channel can be referred to as “upper” or “lower” channel:

  – Upper channel: Channel with a smaller channel number

  – Lower channel: Channel with a higher channel number

  Example:
  For channel 2, channel 1 is an upper channel and channel 3 is a lower channel.

## 15.3 Functional Description

The Timer Array Unit J is used to perform various count or timer operations and to output a signal which depends on the result of the operation. It contains one prescaler block for count clock generation and 4 channels, each equipped with a 32-bit counter TAUJnCNTm and a 32-bit data register TAUJnCDRm to hold the start or compare value of the counter.

It also contains several control and status registers.

**Independent and synchronous operation**

Every channel can operate in two operation modes, either independently or in combination with other channels (synchronously), i.e. multiple channels depend on each other with one master and one or more slave channels.

When a channel is operated independently, its operation mode and functions are not affected by those of other channels. When a channel is operated synchronously it is either a master or a slave. A master channel can have multiple slaves, and the state of one channel affects that of the other channels. For example, this means that one channel can control when another starts to count, is reset, etc.

The following describes the functional blocks:

**Prescaler block**

The prescaler block provides up to 4 clock signals (CK0 to CK3) that can be used as count clocks for all channels.

Count clocks CK0 to CK2 are derived from PCLK by a configurable prescaler division factor of $2^0$ to $2^{15}$. The fourth count clock CK3 can be adjusted more precisely by an additional division factor that is not a power of 2.

**Clock and count clock selection**

For every channel, the count clock selector selects which of the following is used as the clock source:

- One of the clocks CK0 to CK3 (selected by the clock selector)
- INTTAUJnIm from master channel
- Edge detected TAUJnTTINm input signal

**Controller**

The controller controls the main operations of the counter:

- Operation mode (selected by bits TAUJnCMORm.MD[4:0])
- Counter start enable (TAUJnTS.TSm) and counter stop (TAUJnTT.TTm)

  When counter start is enabled, status flag TAUJnTE.TEm is set.

**Trigger selector**

Depending on the selected operation mode, the counter starts automatically when it is enabled (TAUJnTE.TEm = 1), or it waits for an external start trigger signal. Any of the following signals can be used as the start trigger:

- Synchronous channel start trigger input TAUJnTSSTm
- TAUJnTTINm input valid edge
- INTTAUJnIm from the master channel

| | |
|---|---|
| **Simultaneous rewrite controller** | Simultaneous rewrite control is a special function that can be used in synchronous operation modes. The data registers of all channels in a channel group can be rewritten at any time. The simultaneous rewrite controller ensures that new data register values of all channels become effective at the same time. |
| **TAUJnTO Controller** | The output control of every channel enables the generation of various output signal forms such as PWM signals. |
| **Signals** | The TAUJ has various input and output signals.A full list can be found in the first section of this chapter under the keyword “I/O signals”. |

## 15.4 General Operating Procedure

The following lists the general operation procedure for the TAUJn:

After reset release, the operation of each channel is stopped. Clock supply is started and writing to each register is enabled. All circuits and registers of all channels are initialized. The control register of TAUJnTTOUTm is also initialized and outputs a low level.

1. Set the TAUJnTPS and TAUJnBRS registers to specify the clock frequency of CK0 to CK3.
2. Configure the desired TAUJn function:
   – Set the operation mode
   – Set the channel output mode
   – Set any other control bits
3. Enable the counter by setting the TAUJnTS.TSm bit to 1.
   The counter starts to count immediately, or when an appropriate trigger is detected, depending on the bit settings.
   The function is in operation.
4. If desired, and if possible for the configured function, stop the counter or perform a forced restart operation.
5. Stop the function by setting the TAUJnTT.TTm bit to 0.

**Note** A detailed description of the required control bits and the operation of the individual functions is given in the following chapters:

*15.14 “Independent Channel Interrupt Functions” on page 725*

*15.15 “Independent Channel Signal Measurement Fuctions” on page 739*

*15.16 “Other Independent Channel Functions” on page 770*

*15.17 “Synchronous PWM Signal Functions Triggered at Regular Intervals” on page 777*

## 15.5 Operation Modes

The TAUJ contains 7 operation modes. These determine the basic behavior of a channel, for example whether a timer counts up or down, whether the data register TAUJnCDRm acts as a compare register or stores the initial value of the counter, etc.

One operation mode can be set for each channel. It is specified using the TAUJnCMORm.MD[4:0] bits.

When choosing a function, these settings cannot be specified individually, but are grouped under the term operation mode. If a function uses multiple channels, the operation mode of each channel must be set correctly for the function to work correctly.

**Note** For more information about the operation modes required by each function, refer to the required function in the following chapters:

*15.14 “Independent Channel Interrupt Functions” on page 725*

*15.15 “Independent Channel Signal Measurement Fuctions” on page 739*

*15.16 “Other Independent Channel Functions” on page 770*

*15.17 “Synchronous PWM Signal Functions Triggered at Regular Intervals” on page 777*

## 15.6 Concepts of Synchronous Channel Operation

In synchronous channel operation, multiple channels depend on each other, or are affected by changes in another channel. Therefore, several rules apply for the use of synchronous channel functions. These rules are detailed in *15.6.1 "Rules"*.

Two special features for synchronous channel operation are detailed in the following subchapters:

- *15.6.2 "Simultaneous start and stop of synchronous channel counters" on page 705*
- *15.7 "Simultaneous Rewrite" on page 706*

### 15.6.1 Rules

**Number of masters and slaves**

- Only even channels (CH0, CH2) can be set as master channels. Any channel apart from CH0 can be set as a slave channel.
- Only channels lower than the master channel can be set as slave channels, and several slave channels can be set for one master channel.
  Example: If CH2 is a master channel, CH3 can be set as slave channel.
- If two master channels are used, slave channels cannot cross the master channels.
  Example: If CH0 and CH2 are master channels, CH1 can be set as slave channels for CH0, but CH3 cannot.

**Operation clock**

- The same operation clock must be set for the slave channel and the master channel. This is achieved using the TAUJnCMORm.CKS[1:0] bits of the slave and master channel.

The basic concepts of master/slave usage and operation clocks are illustrated in the following figure.

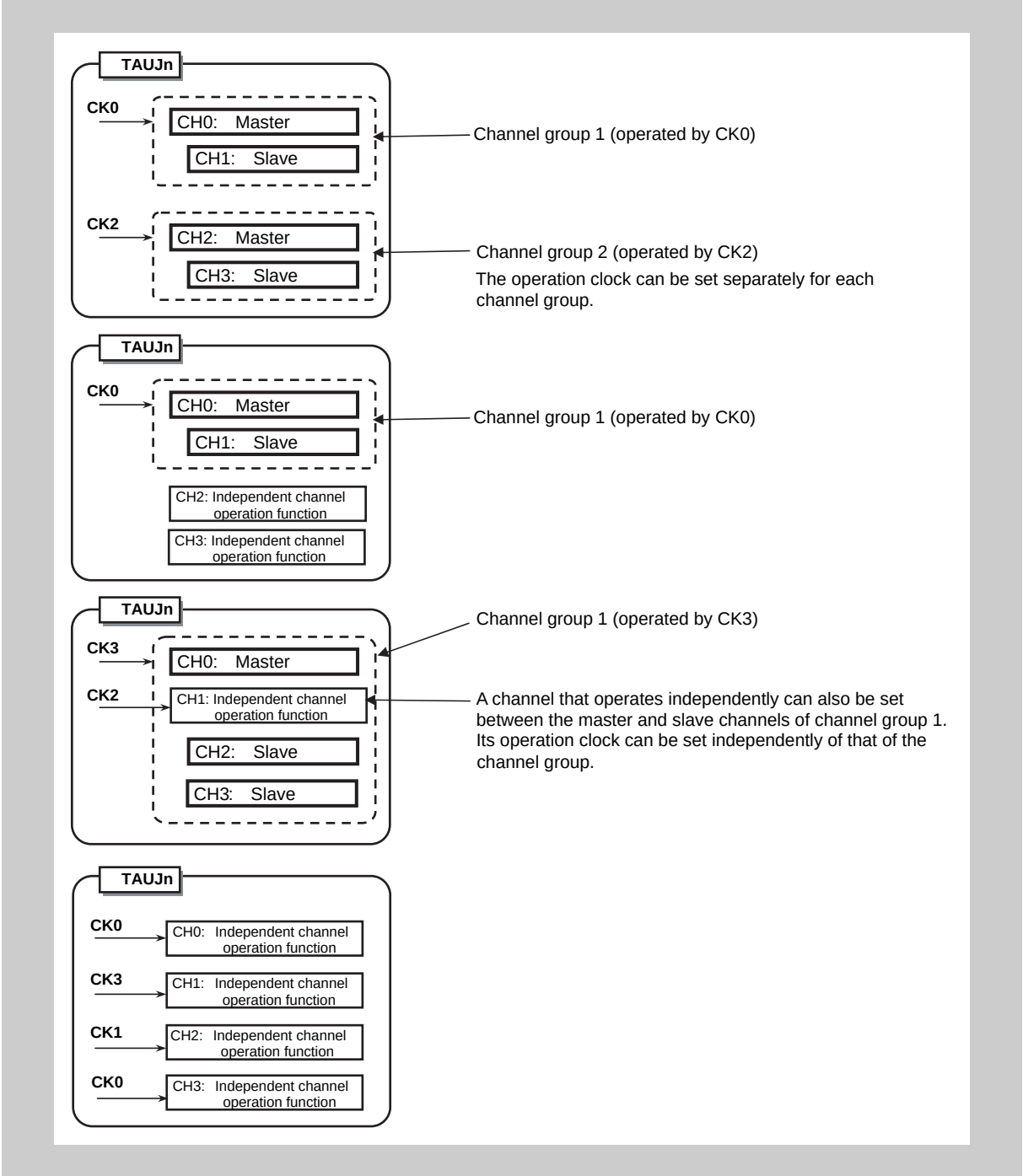



**Figure 15-3 Grouping of the channels and assignment of operation clocks**

**INTTAUJnIm, start trigger, and count clock**

- Master channels can transfer an interrupt request (INTTAUJnI), the start trigger, and the count clock to slave channels.
- Slave channels can use INTTAUJnI, the start trigger, and the count clock of the master channels but cannot transfer their INTTAUJnI, start trigger, or count clock to the lower channels.
- A master channel cannot use INTTAUJnI, the start trigger, or the count clock of the higher master channels.

### 15.6.2 Simultaneous start and stop of synchronous channel counters

Channels that are operated synchronously can be started and stopped simultaneously, both within a TAUJ unit, and between TAUJ units.

**(1) Simultaneous start and stop within a TAUJ unit**

- To simultaneously start synchronized channels, the TAUJnTS.TSm bits of the channels must be set at the same time.
- To simultaneously stop synchronized channels, the TAUJnTT.TTm bits of the channels must be set at the same time.

Writing to the TAUJnTS.TSm bits sets the corresponding TAUJnTE.TEm bits to 1, enabling counting. TAUJnTS.TSm = 1 only enables the corresponding counter to start; the exact time that it starts depends on the operation mode.

**(2) Simultaneous start between TAUJ units**

Counters in different TAUJ units can also be started simultaneously if the corresponding counters are enabled before receiving the simultaneous trigger signal. The simultaneous start trigger register is then sent to the TAUJnTSSTm input.

## 15.7 Simultaneous Rewrite

### 15.7.1 Introduction

Simultaneous rewrite describes the ability to change the compare/start value and the output logic of multiple channels at the same time.

The corresponding data and control registers (TAUJnCDRm and TAUJnTOLm) can nevertheless be written at any time. The new value does not affect the counter operation or the output signal until simultaneous rewrite is triggered.

Simultaneous rewrite can be triggered by the counter on the master channel reaching a certain value.

The following table shows the settings for simultaneous rewrite (TAUJnRDM.RDMm = 0).

**Table 15-7 Simultaneous rewrite settings**

| Method | Simultaneous rewrite triggered when | TAUJnRDE.RDEm |
|---|---|---|
| - | No simultaneous rewrite | 0 |
| A | The master channel (re)starts counting | 1 |

### 15.7.2 How to control simultaneous rewrite

The following figure shows the general procedure for simultaneous rewrite. The three main blocks (Initial settings, Start counter & count operation, and Simultaneous rewrite) are explained afterwards.

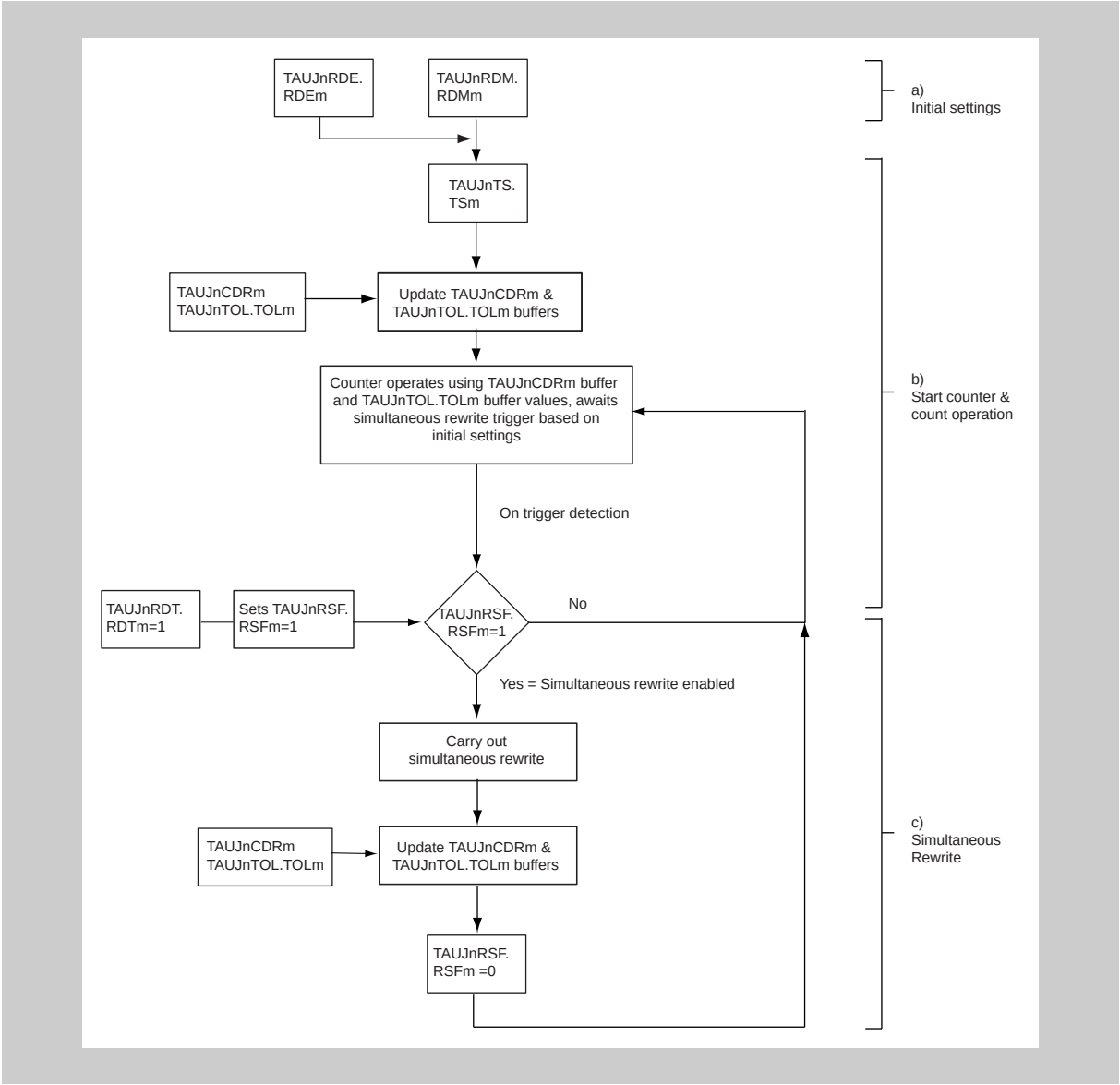



**Figure 15-4 General procedure for simultaneous rewrite**

#### (1) Initial settings

- To enable simultaneous rewrite in channel m, set TAUJnRDE.RDEm = 1
- To select simultaneous rewrite when the master channel starts counting, set TAUJnRDM.RDMm = 0

#### (2) Start counter and count operation

- To start all the TAUJnCNTm counters in the channel group, set the corresponding TAUJnTS.TSm bits to 1. TAUJnTOL.TOLm and the values in the data registers (TAUJnCDRm) are written to the corresponding TOLm buffer (TAUJnTOL.TOLm buf) and data buffer registers (TAUJnCDRm buf)

and the counters start.

- Setting the reload data trigger bit (TAUJnRDT.RDTm) to 1 sets the reload flag (TAUJnRSF.RSFm) to 1, enabling simultaneous rewrite. TAUJnRDT.RDTm then immediately returns to 0, but TAUJnRSF.RSFm remains at 1 until simultaneous rewrite has taken place.
- When the specified trigger for simultaneous rewrite is detected, the TAUJnRSF.RSFm bit is checked to see if simultaneous rewrite is enabled (TAUJnRSF.RSFm = 1). If it is, simultaneous rewrite is carried out. Otherwise the value of the TAUJnRSF.RSFm bit is re-evaluated the next time the trigger is detected.

**(3) Simultaneous rewrite**

- When the simultaneous rewrite trigger is detected and simultaneous rewrite is enabled (TAUJnRSF.RSFm = 1), the current values of the data registers are copied to their buffers. These values are then written to the corresponding counters and the values are applied the next time the counter starts or restarts.
- The TAUJnRSF.RSFm bit is set to 0, and the system awaits the next simultaneous rewrite trigger.

### 15.7.3 Other general rules of simultaneous rewrite

The following rules also apply:

- TAUJnRDE.RDEm and TAUJnRDM.RDMm cannot be changed while the counter is in operation (TAUJnTE.TEm = 1).
- TAUJnTOL.TOLm can only be rewritten during operation when in PWM output function. For all other output functions, TAUJnTOL.TOLm must be written before the counter starts. If it is rewritten in another function, TAUJnTTOUTm outputs an invalid wave.

### 15.7.4 Simultaneous rewrite procedure

Simultaneous rewrite is executed, when the master channel (re)starts counting. The simultaneous rewrite procedure is described in the following figure.

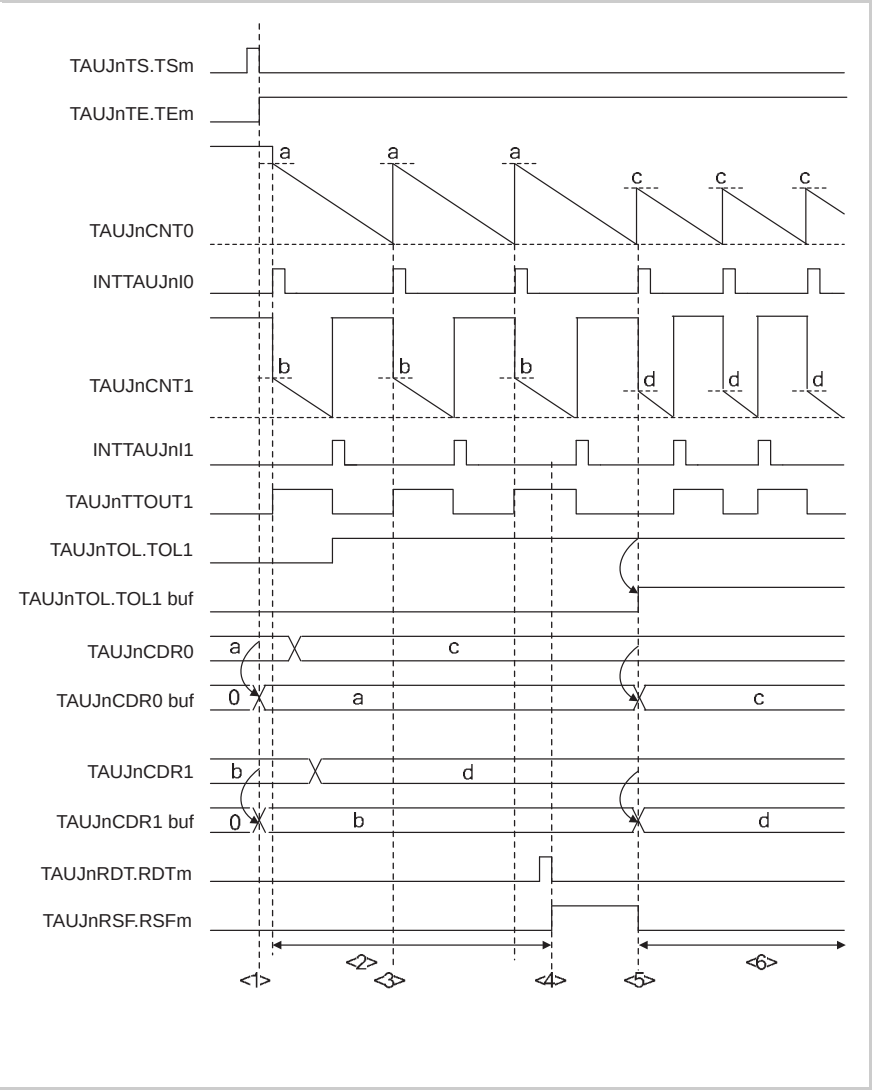



**Figure 15-5 Simultaneous rewrite when the master channel (re)starts counting**

Setup:

- CH0 is the master channel, counting down, CH1 represents an arbitrary slave channel, and simultaneous rewrite is applied when the master channel starts counting.

Description:

1. When the counter starts, the value of TAUJnCDRm is copied to the TAUJnCDRm buffer and the value of TAUJnTOL.TOLm is copied to the TAUJnTOL.TOLm buffer. The TAUJnCDRm buffer value is written to the counter.
2. The TAUJnCDRm and TAUJnTOL.TOLm registers can be written at any time, but the values do not affect the counter as the counter reads the buffer values.
3. CH0 restarts counting, but simultaneous rewrite does not occur because it is disabled (TAUJnRSF.RSFm = 0).
4. The reload data trigger bit (TAUJnRDT.RDTm) is set to 1 which sets the status flag (TAUJnRSF.RSFm = 1), enabling simultaneous rewrite.
5. Simultaneous rewrite is triggered by counter TAUJnCNT0 starting to count down. The TAUJnCDRm value is written to the TAUJnCDRm buffer and the TAUJnTOL.TOLm value is written to the TAUJnTOL.TOLm buffer.
   – The counter starts to count down from the value in the TAUJnCDRm buffer and the TAUJnRSF.RSFm bit is reset to 0.
   – The output logic specified by TAUJnTOL.TOLm becomes effective.
6. The counters count down and await the next simultaneous rewrite trigger. The values of TAUJnCDRm and TAUJnTOL.TOLm can be changed again.

## 15.8 Channel Output Modes

The output of the TAUJnTTOUTm pin can be controlled in two ways, the latter of which can be further split into individual modes:

- By software (Direct Channel Output Mode, TAUJnTOE.TOEm = 0)

  When controlled by software, the output register bit (TAUJnTO.TOm) can be written and the value of the bit is transferred to the output pin (TAUJnTTOUTm).

- By TAUJ signals (TAUJnTOE.TOEm = 1)

  When operated by TAUJ signals, the output level of TAUJnTTOUTm is set or reset or toggled by internal signals. The value of TAUJnTO.TOm is updated accordingly to reflect the value of TAUJnTTOUTm.

  – Independently (Independent Channel Output Mode, TAUJnTOM.TOMm = 0)

    When operated independently, the output of the TAUJnTTOUTm pin is only affected by settings of channel m. Therefore, independent channel operation must be selected (TAUJnTOM.TOMm = 0).

  – Synchronously (Synchronous Channel Output Mode, TAUJnTOM.TOMm = 1)

    When operated synchronously, the output of the TAUJnTTOUTm pin is affected by settings of channel m and those of other channels. Therefore, synchronous channel operation must be selected for all participating channels (TAUJnTOM.TOMm = 1).

The TAUJnTO.TOm bit can always be read to determine the current value of TAUJnTTOUTm, regardless of whether the pin is controlled by software, operated independently, or operated synchronously.

**Control bits**

The settings of the control bits required to select a specific channel output mode are listed in *Table 15-8 “Channel output modes” on page 712* .

The channel output modes are described in detail in

- *15.8.2 “Channel output modes controlled independently by TAUJn signals” on page 714* to
- *15.8.3 “Channel output modes controlled synchronously by TAUJn signals” on page 715* .

**Output logic**

Positive logic or inverted logic of the output is specified by control bit TAUJnTOL.TOLm.

The value of the TAUJnTOL.TOLm bit must be set before the counter is started. It can only be changed during operation in PWM output function. Otherwise, changes to TAUJnTOL.TOLm result in an invalid TAUJnTTOUTm signal.

Refer to *15.7 “Simultaneous Rewrite” on page 706* .

The channel output modes and the channel output control bits are listed in the following table (TAUJnTOC.TOCm = 0).

**Table 15-8 Channel output modes**

| Channel output mode | TAUJnTOE.TOEm | TAUJnTOM.TOMm |
|---|---|---|
| **By software** | | |
| Direct Channel Output Mode | 0 | x |
| **By timer signals, independently (Independent Channel Output Mode)** | | |
| Independent Channel Output Mode 1 | 1 | 0 |
| **By timer signals, synchronously (Synchronous Channel Output Mode)** | | |
| Synchronous Channel Output Mode 1 | 1 | 1 |

- The combinations not listed in this table are forbidden.
- The bit marked with an x can be set to any value.

**Note** The following bits cannot be changed during count operation (TAUJnTE.TE = 1):

- TAUJnTOM.TOMm
- TAUJnTOC.TOCm

### 15.8.1 General procedure for specifying a channel output mode

The following steps describe the general procedure for specifying a TAUJnTTOUTm channel output mode. The prerequisite is that timer output operation is disabled (TAUJnTOE.TOEm = 0).

1. Set TAUJnTO.TOm to specify the initial level of the TAUJnTTOUTm output.
2. Set the channel output mode using *Table 15-8 “Channel output modes” on page 712* and the output logic using the TAUJnTOL.TOLm bit.
3. Start the counter (TAUJnTS.TSm = 1).

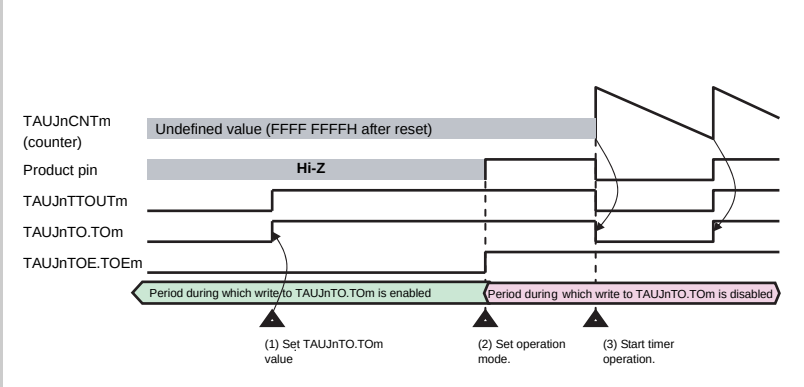



**Figure 15-6 General procedure for specifying a TAUJnTTOUTm channel output mode**

The following figure shows a general illustration of how the output changes when the counter is enabled:

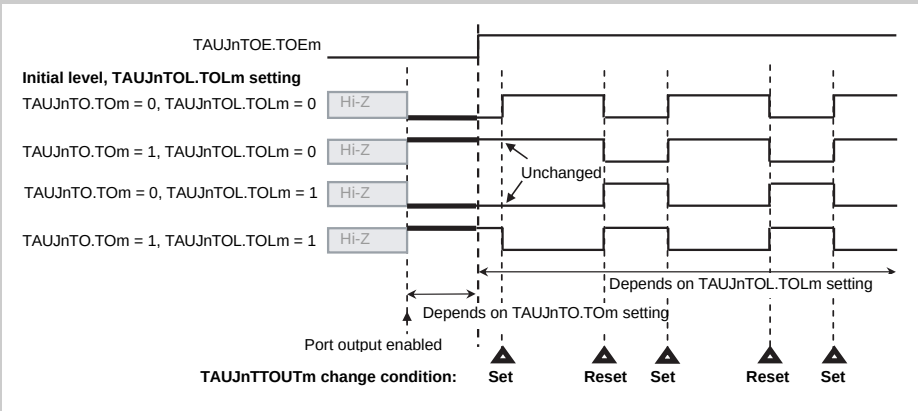



**Figure 15-7 General change of the TAUJnTTOUTm output**

- TAUJnTO.TOm sets the initial value of TAUJnTTOUTm and can be changed while TAUJnTOE.TOEm = 0.
- TAUJnTOL.TOLm specifies whether the set signal sets TAUJnTO.TOm to high (TAUJnTOL.TOLm = 0) or low (inverted logic, TAUJnTOL.TOLm = 1).

### 15.8.2 Channel output modes controlled independently by TAUJn signals

This chapter lists the channel output modes that are controlled independently by TAUJn signals. The control bits used to specify a mode are listed in *Table 15-8 “Channel output modes” on page 712* .

**(1) Independent Channel Output Mode 1**

**Set/reset conditions** In this output mode, TAUJnTTOUTm toggles when INTTAUJnIm is detected. The value of TAUJnTOL.TOLm is ignored.

**Prerequisites** None, other than those in *Table 15-8 “Channel output modes” on page 712* .

### 15.8.3 Channel output modes controlled synchronously by TAUJn signals

This chapter lists the channel output modes that are controlled synchronously by TAUJn signals. The control bits used to specify a mode are listed in *Table 15-8 “Channel output modes” on page 712* .

#### (1) Synchronous Channel Output Mode 1

**Set/reset conditions**

In this output mode, INTTAUJnIm of the master channel serves as the set signal and INTTAUJnIm of the slave channel as the reset signal. If INTTAUJnIm of the master channel and INTTAUJnIm of the slave channel are generated at the same time, INTTAUJnIm of the slave channel (reset signal) has priority over INTTAUJnIm (set signal) of the master channel, i.e. the master channel is ignored.

**Prerequisites**

None, other than those in *Table 15-8 “Channel output modes” on page 712* .

## 15.9 Start Timing of Operating Modes

This chapter describes when the counters of the different operating modes start after the TAUJnTS.TSm bit is set to 1.

In all modes, the value of the data register and whether or not an interrupt is issued depends on the individual mode and corresponding register settings.

### 15.9.1 Interval Timer Mode, Capture Mode

The counter starts at the start of the next count clock cycle after TAUJnTS.TSm is set to 1. The value of data register is also loaded at the point the counter starts.

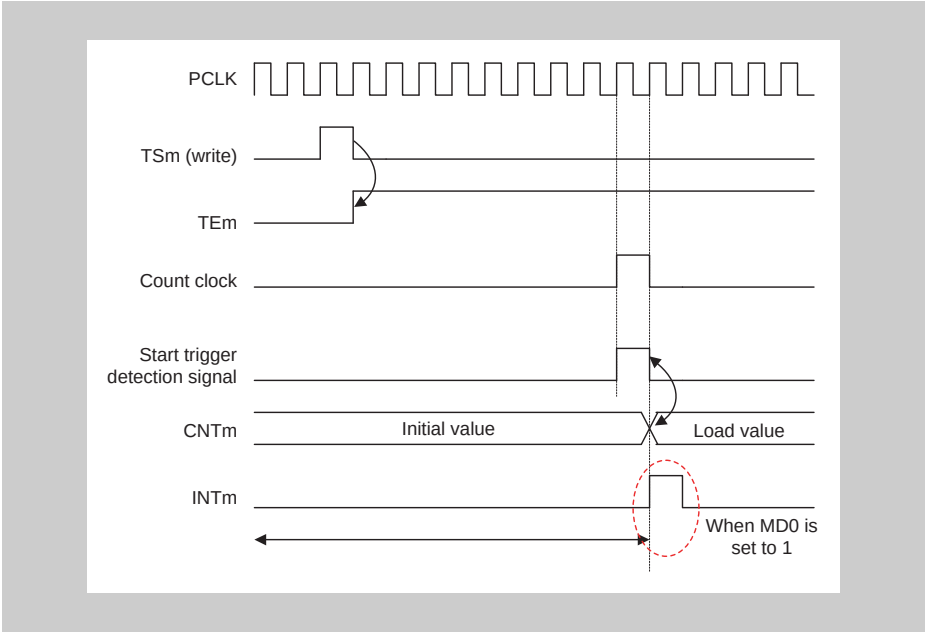



**Figure 15-8 Start timing of Interval Timer Mode, Capture Mode**

### 15.9.2 Other operating modes

In all other operating modes, the count clock cycles are ignored with regard to starting the counter. The counter is only triggered by detection of a valid TAUJnTTINm edge. The value of data register is also loaded at the point the counter starts. Nevertheless, the count clock cycles determine the frequency with which all operations take place.

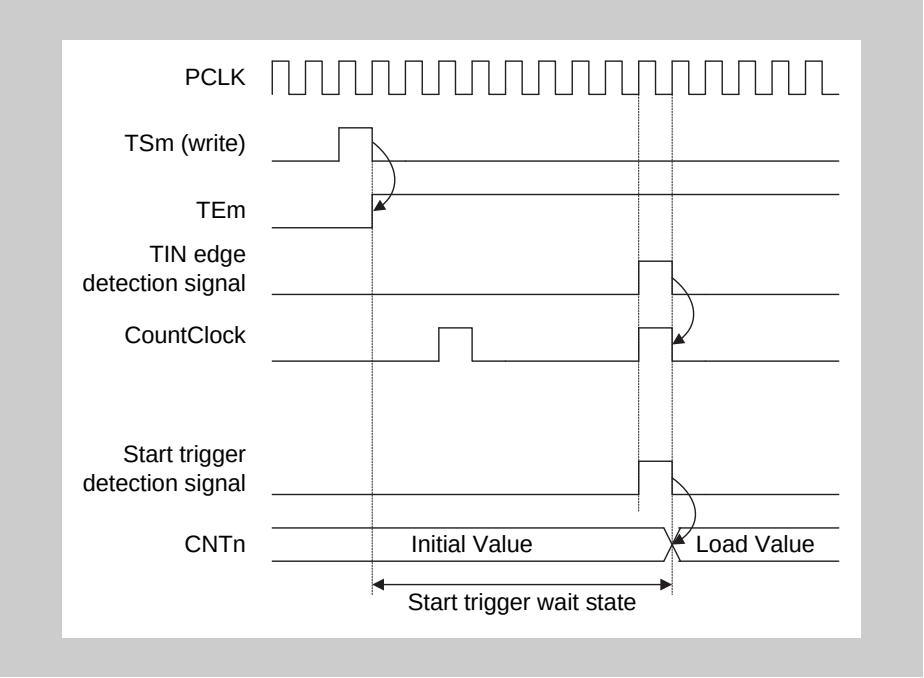



**Figure 15-9 Start timing of all other operating modes**

## 15.10 TAUJnTTOUTm Output and INTTAUJnIm Generation when Counter Starts or Restarts

When the counter starts, it is possible to specify whether an INTTAUJnIm is generated using the TAUJnCMORm.MD0 bit. The effect of the bit depends on the selected mode, as shown in the following table. The effects of INTTAUJnIm on TAUJnTTOUTm depend on the selected channel operation function.

**Table 15-9 Effect of CMORm.MD0 bit on generation of INTTAUJnIm when counter is triggered**

| Mode | TAUJnCMORm.MD0 bit | INTTAUJnIm generated when counter starts |
|---|---|---|
| Interval Timer Mode<br>Capture Mode<br>Count Capture Mode | 0 | No |
| | 1 | Yes |
| Capture & One Count Mode<br>Capture & Gate Count Mode | 0 | No |
| One Count Mode<br>Gate Count Mode | 0/1 | No, regardless of setting of TAUJnCMORm.MD0 bit. |

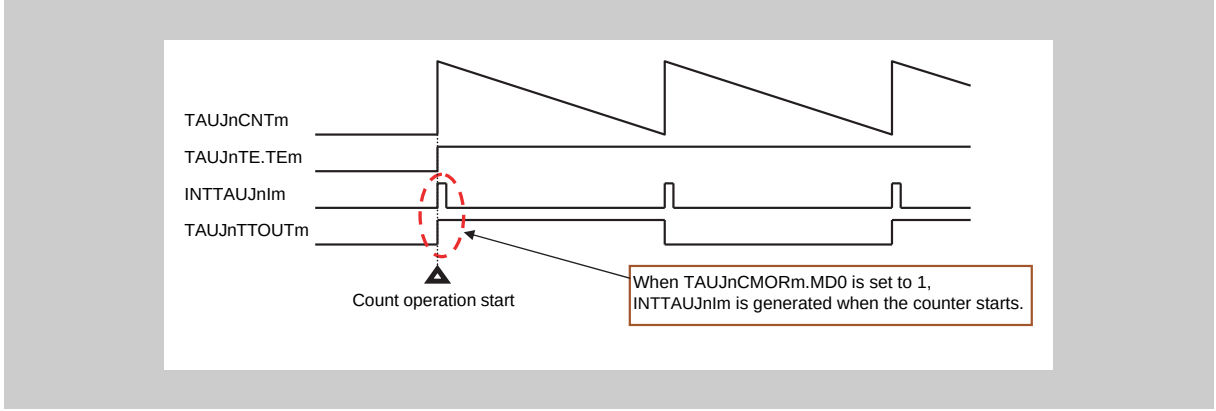



**Figure 15-10 INTTAUJnIm generated when counter starts**

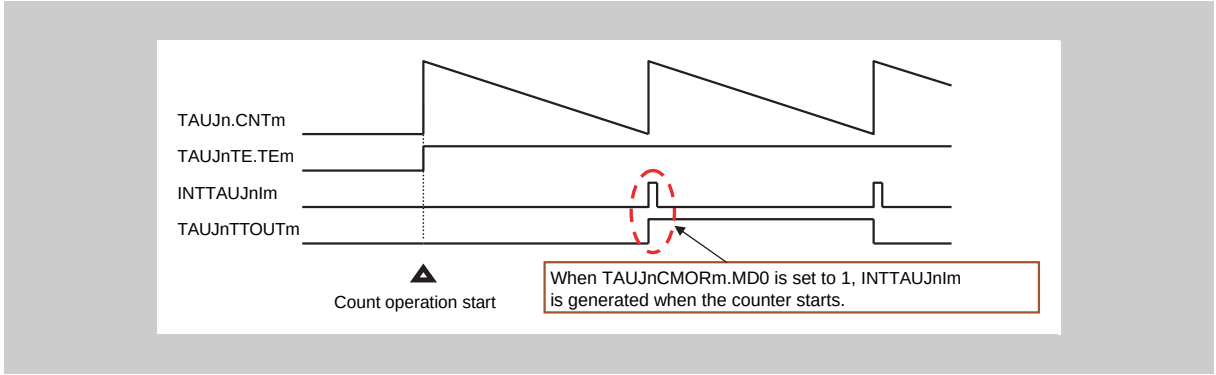



**Figure 15-11 INTTAUJnIm not generated when counter starts**

## 15.11 Interrupt Generation upon Overflow

Certain independent functions that count up, overflow without generating an interrupt when they reach FFFF $FFFF_H$. This section describes how it is possible to generate an interrupt, by combining a channel operating in one of these modes with a channel in a different operation mode which counts down.

The appropriate operation mode for the second channel depends on the operation mode of the first channel. Nevertheless, the principle is the same for all combinations:

- Find a operation mode for the second channel that counts down in such a manner, that it reaches 0000 $0000_H$ at the same time as the first channel overflows (TAUJnCNTm = FFFF $FFFF_H$).
- Set TAUJnCDRm of the second channel to FFFF $FFFF_H$
- The two channels must count at the same speed (i.e. they must have the same count clock)
- Both channels are triggered by the same TAUJnTTINm input
- The trigger detection settings (TAUJnCMORm.STS[2:0] and TAUJnCMURm.TIS[1:0]) must be identical for both channels
- This is only possible for TAUJ0 and TAUJ4 by using configurable "input selectors" (on system level)

Result: the down-counter of the second channel reaches 0000 $0000_H$ at exactly the same time as the up-counter of the first channel overflows (TAUJnCNTm = FFFF $FFFF_H$). Thus the second channel generates the desired interrupt.

The following sections list the operating modes that count down that are required to match specific operating modes that count up, as well as example timing diagrams.

### 15.11.1 Capture Mode

**Applies to**

- TAUJnTTINm Input Pulse Interval Measurement Function
- Real-Time Output Function Type 2
- Simultaneous Rewrite Trigger Generation Function Type 2

**Combine with** Interval Timer Mode

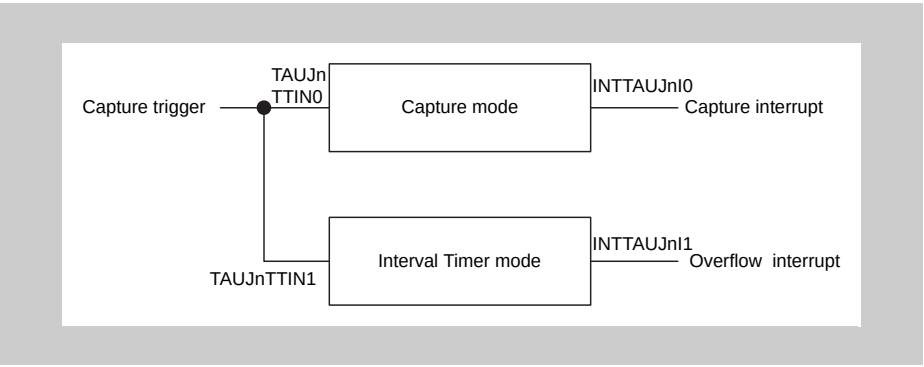



**Figure 15-12 Combination of Capture Mode and Interval Timer Mode**

**Timing diagram**

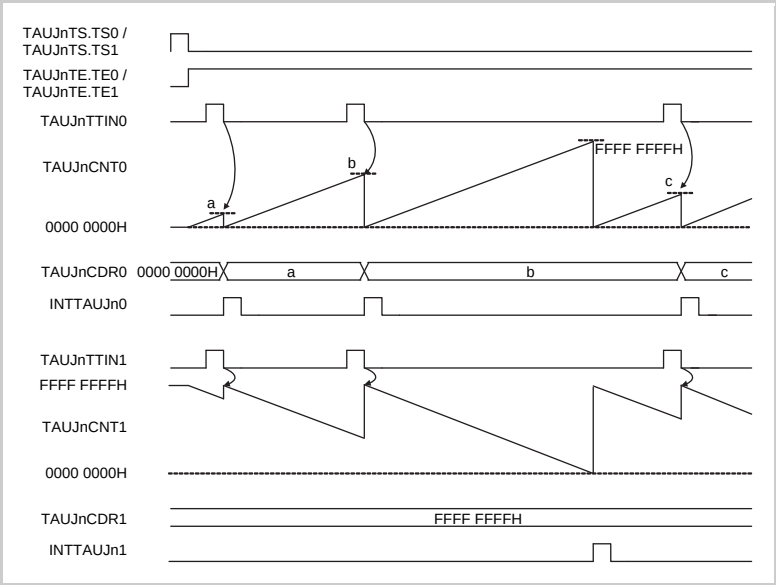



**Figure 15-13 Interrupt generation via combination of Capture Mode and Interval Timer Mode**

### 15.11.2 Capture and One Count Mode

**Applies to** • TAUJnTTINm Input Signal Width Measurement Function

**Combine with** One Count Mode

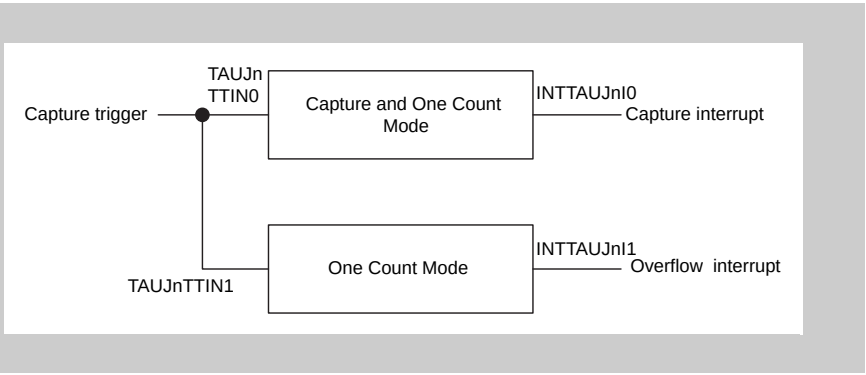



**Figure 15-14 Combination of Capture and One Count Mode and One Count Mode**

**Timing diagram**

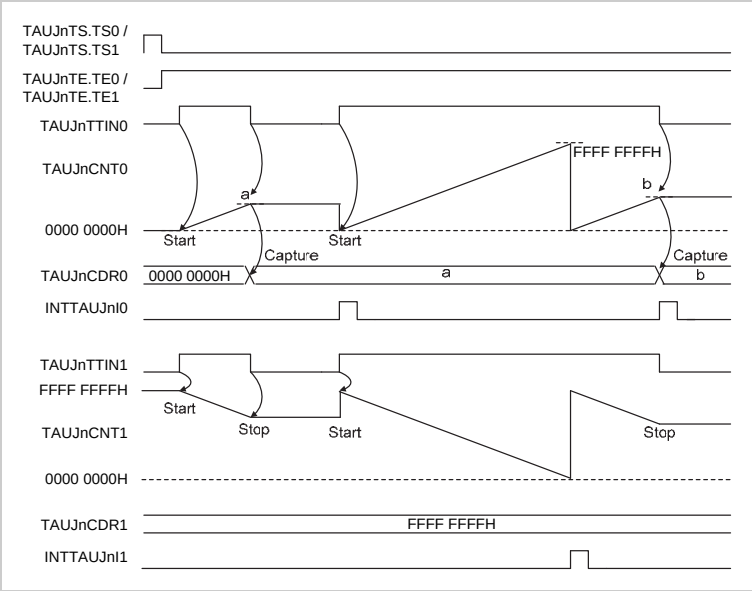



**Figure 15-15 Interrupt generation via combination of Capture and One Count Mode and One Count Mode**

### 15.11.3 Count Capture Mode

**Applies to** • TAUJnTTINm Input Position Detection Function

**Combine with** Interval Timer Mode

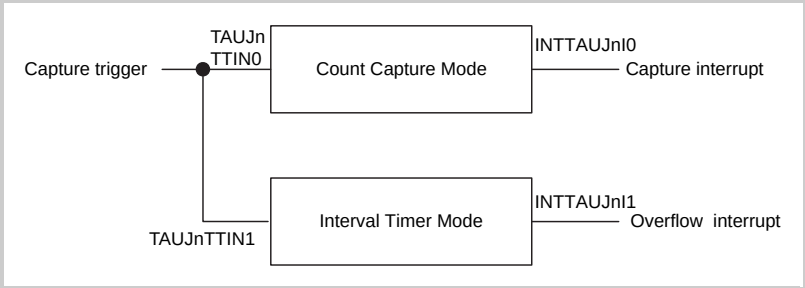



**Figure 15-16 Combination of Count Capture Mode and Interval Timer Mode**

**Timing diagram**

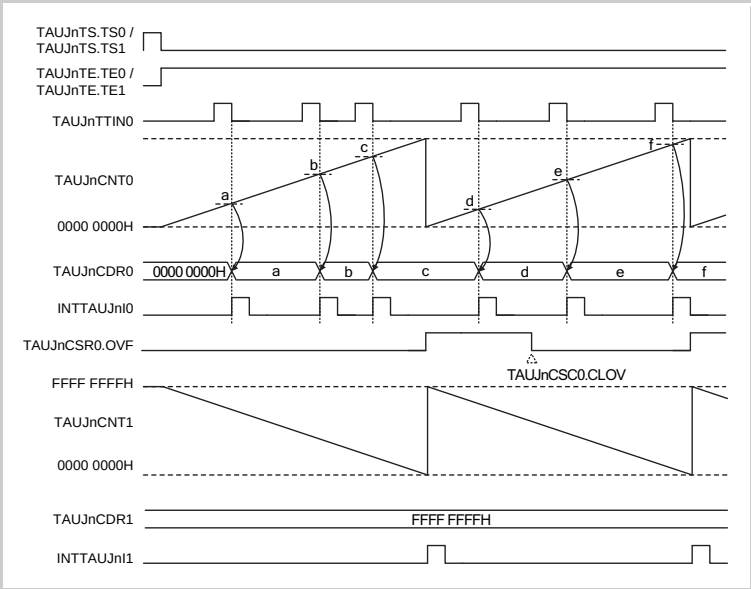



**Figure 15-17 Interrupt generation via combination of Count Capture Mode and Interval Timer Mode**

In the above timing diagram, TAUJnCSRm.OVF is set to 1 when TAUJnCNTm overflows. It is reset by a software command (TAUJnCSCm.CLOV = 1).

## 15.11.4 Capture and Gate Count Mode

**Applies to** • TAUJnTTINm Input Period Count Detection Function

**Combine with** Gate Count Mode

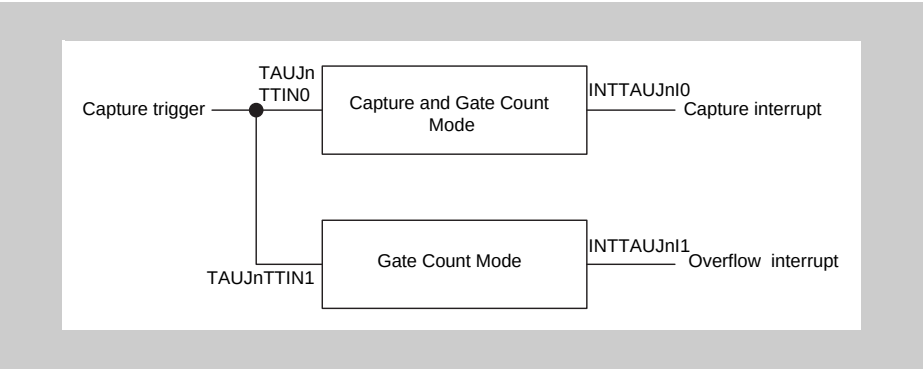



**Figure 15-18 Combination of Capture and Gate Count Mode and Gate Count Mode**

**Timing diagram**

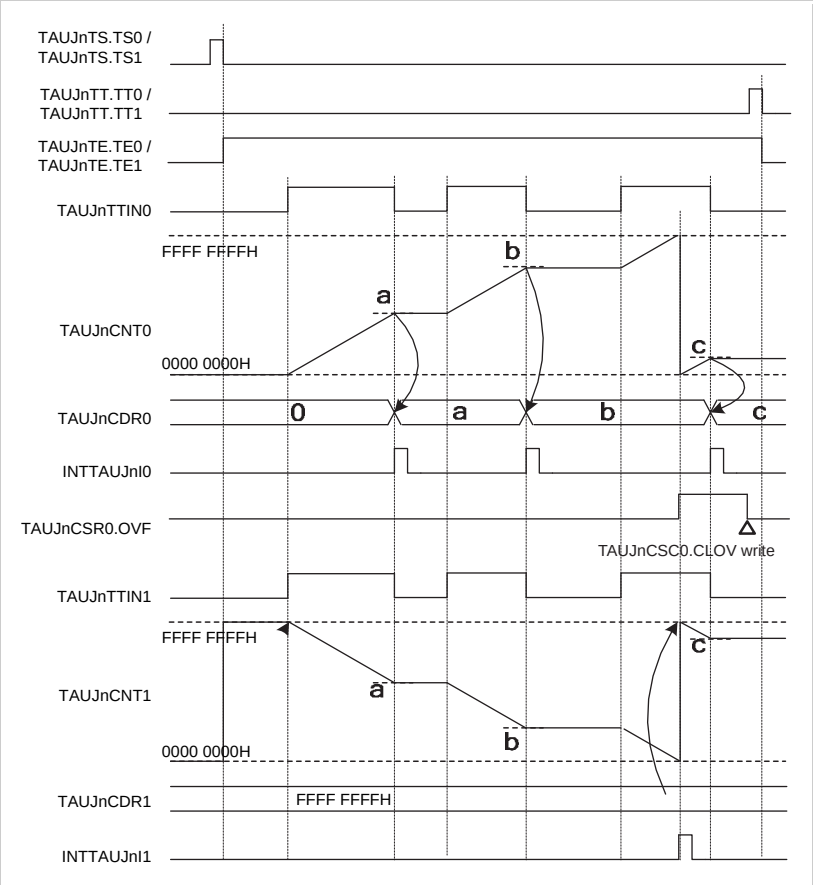



**Figure 15-19 Interrupt generation via combination of Capture and Gate Count Mode and Gate Count Mode**

In the above timing diagram, TAUJnCSRm.OVF is set to 1 when TAUJnCNTm overflows. It is reset by a software command (TAUJnCSCm.CLOV = 1).

## 15.12 TAUJnTTINm Edge Detection

Edge detection is based on the operation clock. This means that an edge can only be detected at the next rising edge of the operation clock. This can lead to a maximum delay of one operation clock cycle.

The following figure shows when edge detection takes place.

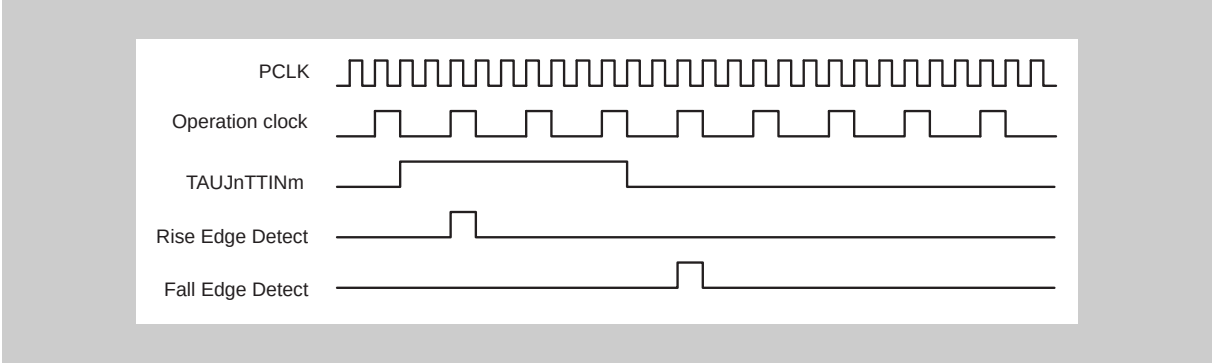



**Figure 15-20 Basic edge detection timing**

## 15.13 Independent Channel Operation Functions

The following sections list the independent channel operation functions provided by the Timer Array Unit J. For a general overview of independent channel operation, see *15.3 “Functional Description” on page 699* .

## 15.14 Independent Channel Interrupt Functions

This chapter describes functions that generate interrupts at regular intervals.

- *15.14.1 “Interval Timer Function”*
- *15.14.2 “TAUJnTTINm Input Interval Timer Function”*

### 15.14.1 Interval Timer Function

#### (1) Overview

**Summary** This function is used as a reference timer for generating timer interrupts (INTTAUJnIm) at regular intervals. When an interrupt is generated, the TAUJnTTOUTm signal toggles, resulting in a square wave.

**Prerequisites**
- The operation mode must be set to Interval Timer Mode, refer to *Table 15-10 "TAUJnCMORm settings for Interval Timer Function" on page 728*
- The channel output mode must be set to Independent Channel Output Mode 1, refer to *15.8 "Channel Output Modes" on page 711*

**Description** The counter is started by setting the channel trigger bit (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation. The current value of TAUJnCDRm is written to TAUJnCNTm and the counter starts to count down from this value.

When the counter reaches 0000 $0000_H$, INTTAUJnIm is generated and the TAUJnTTOUTm signal toggles. TAUJnCNTm then reloads the TAUJnCDRm value and subsequently continues operation.

The value of TAUJnCDRm can be rewritten at any time, and the changed value of TAUJnCDRm is applied the next time the counter starts to count down.

The counter can be stopped by setting TAUJnTT.TTm to 1, which in turn sets TAUJnTE.TEm to 0. TAUJnCNTm and TAUJnTTOUTm stop but retain their values. The counter can be reset by setting TAUJnTS.TSm to 1. The counter can also be forcibly restarted (without stopping it first) by setting TAUJnTS.TSm to 1 during operation.

**Conditions** If the TAUJnCMORm.MD0 bit is set to 0, the first interrupt after a start or restart is not generated, and therefore TAUJnTTOUTm does not toggle. This results in an inverted TAUJnTTOUTm signal compared to when TAUJnCMORm.MD0 is set to 1. For details refer to *15.10 "TAUJnTTOUTm Output and INTTAUJnIm Generation when Counter Starts or Restarts" on page 718* .

#### (2) Equations

INTTAUJnIm cycle = count clock cycle x (TAUJnCDRm + 1)

TAUJnTTOUTm square wave cycle = count clock cycle x (TAUJnCDRm + 1) x 2

### (3) Block diagram and general timing diagram

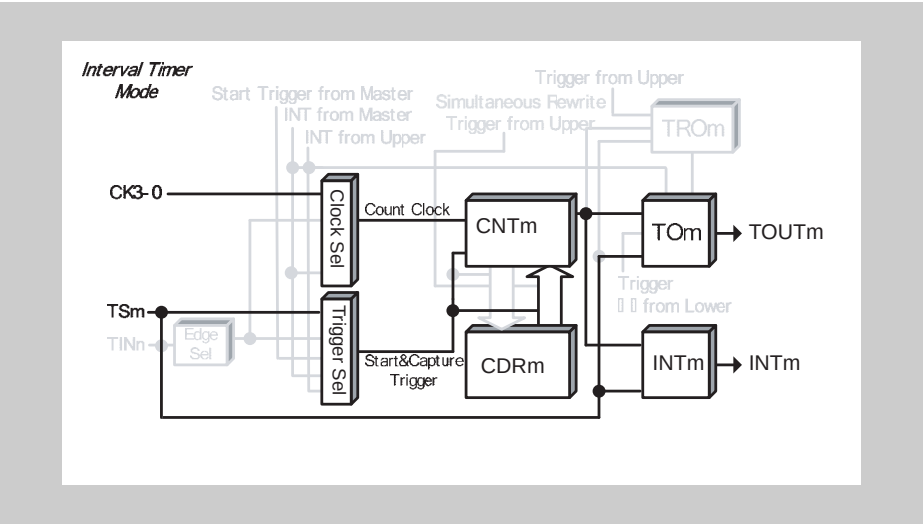



**Figure 15-21 Block diagram for Interval Timer Function**

The following settings apply to the general timing diagram:

- INTTAUJnIm not generated at operation start (TAUJnCMORm.MD0 = 0)

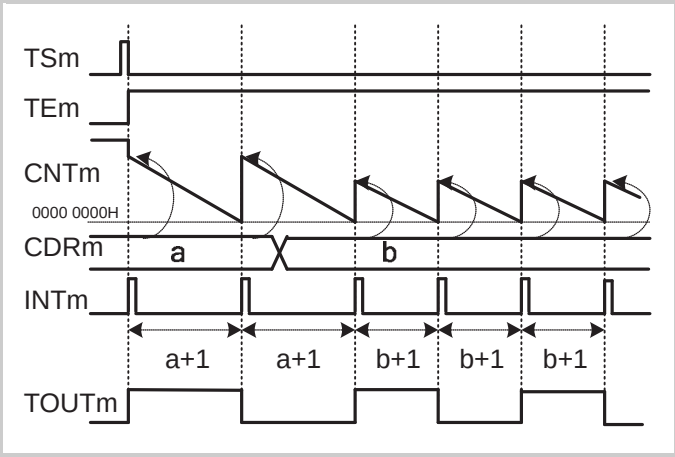



**Figure 15-22 General timing diagram for Interval Timer Function**

### (4) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

Table 15-10 TAUJnCMORm settings for Interval Timer Function

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUJnIm not generated and TAUJnTTOUTm does not toggle at operation start or restart<br>1: Generates INTTAUJnIm and toggles TAUJnTTOUTm at operation start or restart |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

Table 15-11 TAUJnCMURm settings for Interval Timer Function

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used, so set to 00 |

### (c) Channel output mode

**Table 15-12 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOE.TOEm | 1: Disables Direct Channel Output Mode |
| TOM.TOMm | 0: Independent channel output |
| TOC.TOCm | 0: Operation mode 1 (= Toggle mode if TAUJnTOM.TOMm = 0) |
| TOL.TOLm | 0: Positive logic |

**Note** The channel output mode can also be set to Direct Channel Output Mode by setting TAUJnTOE.TOEm = 0. TAUJnTTOUTm can then be controlled independently of the interrupts. For details refer to *15.8 “Channel Output Modes” on page 711* .

### (d) Simultaneous rewrite

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the Interval Timer Function. Therefore, these registers must be set to 0.

**Table 15-13 Simultaneous rewrite settings for Interval Timer Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (5) Operating procedure for Interval Timer Function

**Table 15-14 Operating procedure for Interval Timer Function**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-10 "TAUJnCMORm settings for Interval Timer Function" on page 728* and *Table 15-11 "TAUJnCMURm settings for Interval Timer Function" on page 728*<br><br>Set the value of the TAUJnCDRm register<br><br>Set the channel output mode by setting the control bits as described in *Table 15-12 "Control bit settings for Independent Channel Output Mode 1" on page 729* | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is set to 1 and the counter starts.<br>TAUJnCNTm loads the TAUJnCDRm value.<br>When TAUJnCMORm.MD0 = 1, INTTAUJnIm is generated and TAUJnTTOUTm toggles. |
| During operation | The TAUJnCDRm register value can be changed at any time.<br>The TAUJnCNTm register can be read at all times. | TAUJnCNTm counts down. When the counter reaches 0000 $0000_H$:<br>• TAUJnCNTm reloads the TAUJnCDRm value and continues count operation<br>• INTTAUJnIm is generated and TAUJnTTOUTm toggles. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm and TAUJnTTOUTm stop and retain their current values. |

Restart

**(6) Specific timing diagrams**

**(a) TAUJnCDRm = 0000 0000$_H$, count clock = PCLK/2**

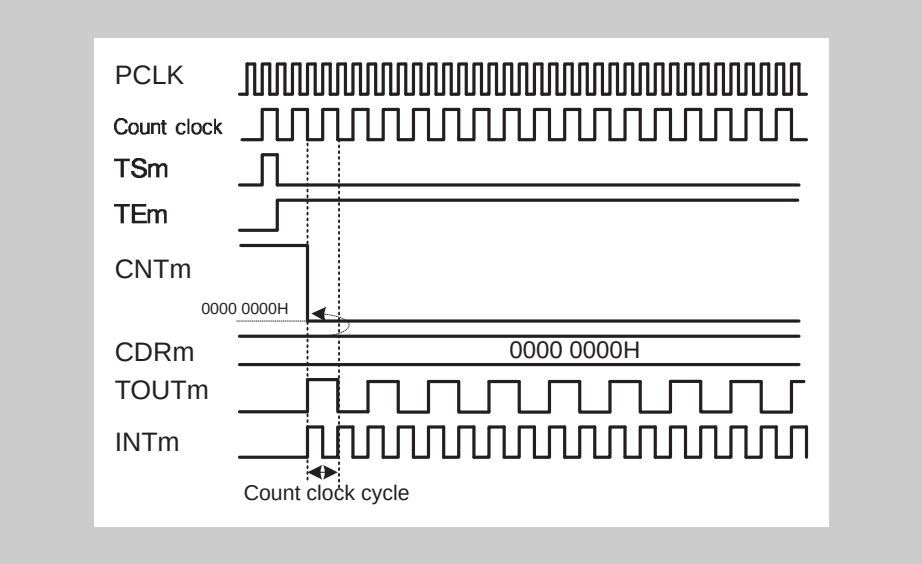



**Figure 15-23 TAUJnCDRm = 0000 0000$_H$, count clock = PCLK/2**

- If TAUJnCDRm = 0000 0000$_H$ and the count clock = PCLK/$2^1$, the TAUJnCDRm value is written to TAUJnCNTm every count clock, meaning that TAUJnCNTm is always 0000 0000$_H$.
- INTTAUJnIm is generated every count clock, resulting in TAUJnTTOUTm toggling every count clock.

**(b) TAUJnCDRm = 0000 0000$_H$, count clock = PCLK**

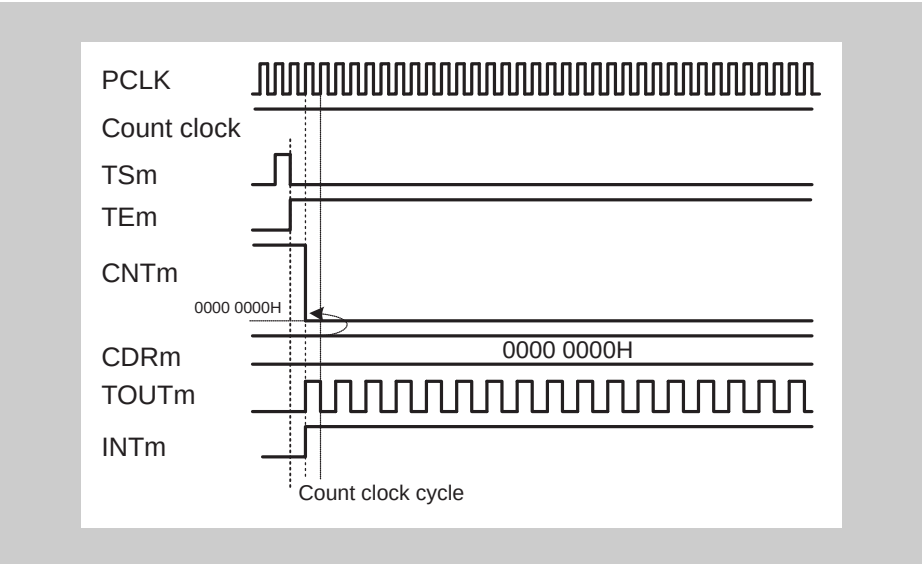



**Figure 15-24 TAUJnCDRm = 0000 0000$_H$, count clock = PCLK**

- If TAUJnCDRm = 0000 0000$_H$ and the count clock = PCLK, the TAUJnCDRm value is written to TAUJnCNTm every PCLK clock, meaning that TAUJnCNTm is always 0000 0000$_H$.
- INTTAUJnIm is generated continuously, resulting in TAUJnTTOUTm toggling every PCLK clock.

### (c) Operation stop and restart

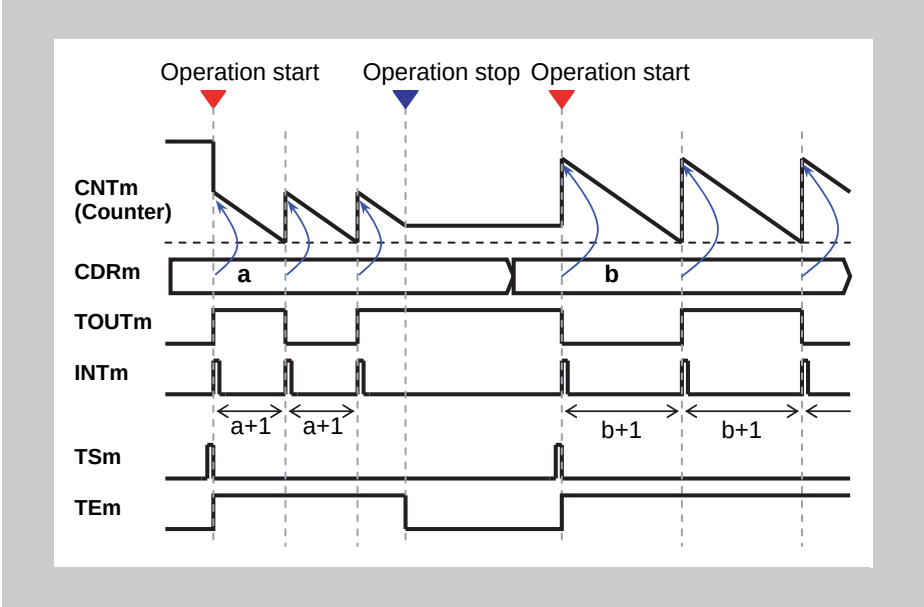



Figure 15-25 Operation stop and restart, TAUJnCMORm.MD0 = 1

- The counter can be stopped by setting TAUJnTT.TTm to 1, which in turn sets TAUJnTE.TEm to 0.
- TAUJnCNTm and TAUJnTTOUTm stop but retain their values.
- The counter can be restarted by setting TAUJnTS.TSm to 1.

### (d) Forced restart

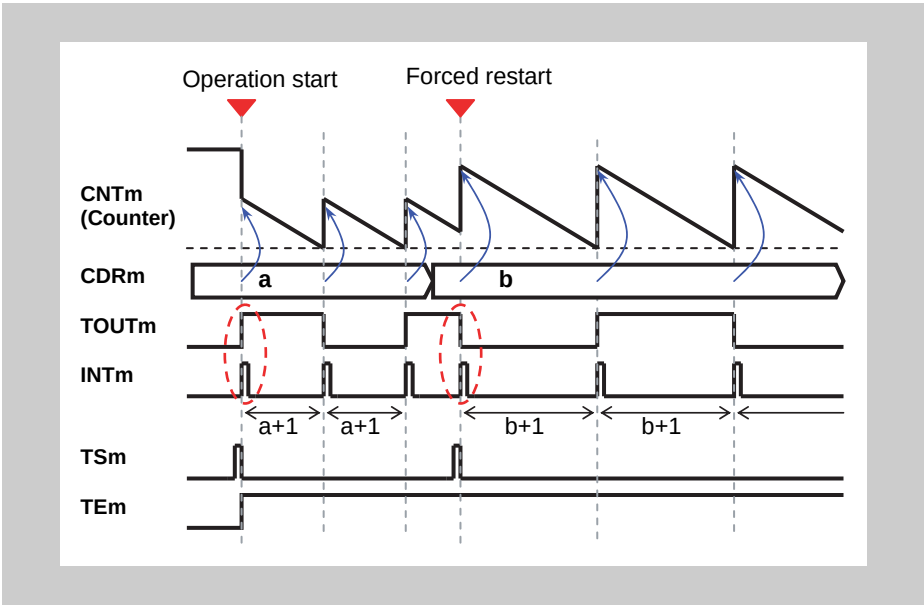



Figure 15-26 Forced restart operation, TAUJnCMORm.MD0 = 1

- The counter can be forcibly restarted (without stopping it first) by setting TAUJnTS.TSm to 1 during operation.
- If the TAUJnCMORm.MD0 bit is set to 1, the first interrupt after a start or restart is generated.

## 15.14.2 TAUJnTTINm Input Interval Timer Function

### (1) Overview

**Summary** This function is used as a reference timer for generating timer interrupts (INTTAUJnIm) at regular intervals or when a valid TAUJnTTINm input edge is detected. When an interrupt is generated, the TAUJnTTOUTm signal toggles, resulting in a square wave.

**Prerequisites**

- The operation mode must be set to Interval Timer Mode, refer to *Table 15-15 “TAUJnCMORm settings for TAUJnTTINm Input Interval Timer Function” on page 735*
- The channel output mode must be set to Independent Channel Output Mode 1, refer to *15.8 “Channel Output Modes” on page 711*

**Description** This function operates in an identical manner to the Interval Timer Function (see *15.14.1 “Interval Timer Function” on page 726* ), except that this function is restarted by a valid TAUJnTTINm input edge. The type of edge used as the trigger is specified using the TAUJnCMURm.TIS[1:0] bits. Either rising edge, falling edge, or rising and falling edge can be selected.

### (2) Equations

INTTAUJnIm cycle = count clock cycle x (TAUJnCDRm + 1)

TAUJnTTOUTm square wave cycle = count clock cycle x (TAUJnCDRm + 1) x 2

### (3) Block diagram and general timing diagram

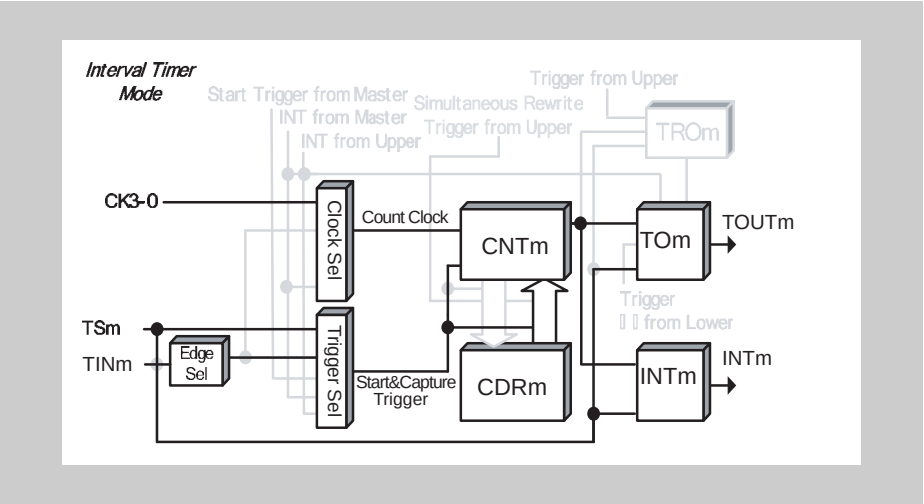



**Figure 15-27 Block diagram for TAUJnTTINm Input Interval Timer Function**

The following settings apply to the general timing diagram:

- INTTAUJnIm not generated at operation start (TAUJnCMORm.MD0 = 0)
- Rising edge detection (TAUJnCMURm.TIS[1:0] = $01_B$)

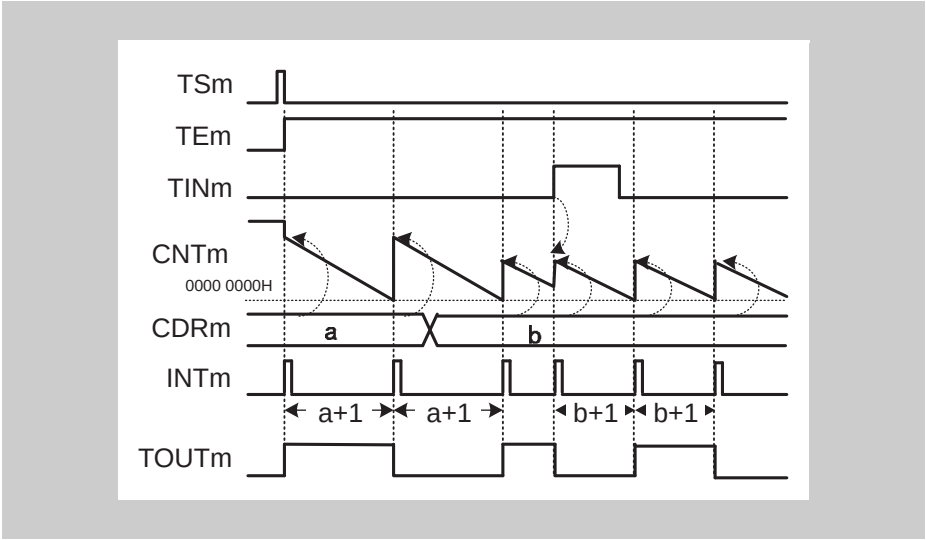



**Figure 15-28 General timing diagram for TAUJnTTINm Input Interval Timer Function**

### (4) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-15 TAUJnCMORm settings for TAUJnTTINm Input Interval Timer Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid TAUJnTTINm input edge signal is used as the external start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 0: INTTAUJnIm not generated and TAUJnTTOUTm does not toggle at operation start<br>1: Generates INTTAUJnIm and toggles TAUJnTTOUTm at operation start |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-16 TAUJnCMURm settings for TAUJnTTINm Input Interval Timer Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection |

### (c) Channel output mode

**Table 15-17 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOE.TOEm | 1: Disables Direct Channel Output Mode |
| TOM.TOMm | 0: Independent channel output |
| TOC.TOCm | 0: Operation mode 1 (= Toggle mode if TAUJnTOM.TOMm = 0) |
| TOL.TOLm | 0: Positive logic |

**Note** The channel output mode can also be set to Direct Channel Output Mode by setting TAUJnTOE.TOEm = 0. TAUJnTTOUTm can then be controlled independently of the interrupts. For details refer to *15.8 “Channel Output Modes” on page 711* .

### (d) Simultaneous rewrite

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the TAUJnTTINm Input Interval Timer Function. Therefore, these registers must be set to 0.

**Table 15-18 Simultaneous rewrite settings for TAUJnTTINm Input Interval Timer Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (5) Operating procedure for TAUJnTTINm Input Interval Timer Function

**Table 15-19 Operating procedure for TAUJnTTINm Input Interval Timer Function**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-15 "TAUJnCMORm settings for TAUJnTTINm Input Interval Timer Function" on page 735* and *Table 15-16 "TAUJnCMURm settings for TAUJnTTINm Input Interval Timer Function" on page 735*<br><br>Set the value of the TAUJnCDRm register<br><br>Set the channel output mode by setting the control bits as described in *Table 15-17 "Control bit settings for Independent Channel Output Mode 1" on page 736* | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is set to 1 and the counter starts.<br>TAUJnCNTm loads the TAUJnCDRm value.<br>When TAUJnCMORm.MD0 = 1, INTTAUJnIm is generated and TAUJnTTOUTm toggles. |
| During operation | The values of the TAUJnCMURm.TIS[1:0] and TAUJnTO.TOm bits and the TAUJnCDRm register can be changed at any time.<br>The TAUJnCNTm register can be read at all times.<br><br>Detection of TAUJnTTINm edge | TAUJnCNTm counts down. When the counter reaches 0000 $0000_H$:<br>• TAUJnCNTm reloads the TAUJnCDRm value and continues count operation<br>• INTTAUJnIm is generated and TAUJnTTOUTm toggles<br>When a TAUJnTTINm input valid edge is detected during count operation, TAUJnCNTm reloads the TAUJnCDRm value and continues count operation.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm and TAUJnTTOUTm stop and retain their current values. |

Restart

**(6) Specific timing diagrams**

The timing diagrams in *15.14.1 "Interval Timer Function" on page 726* also apply, except for this function the counter can also be restarted by a valid TAUJnTTINm input edge.

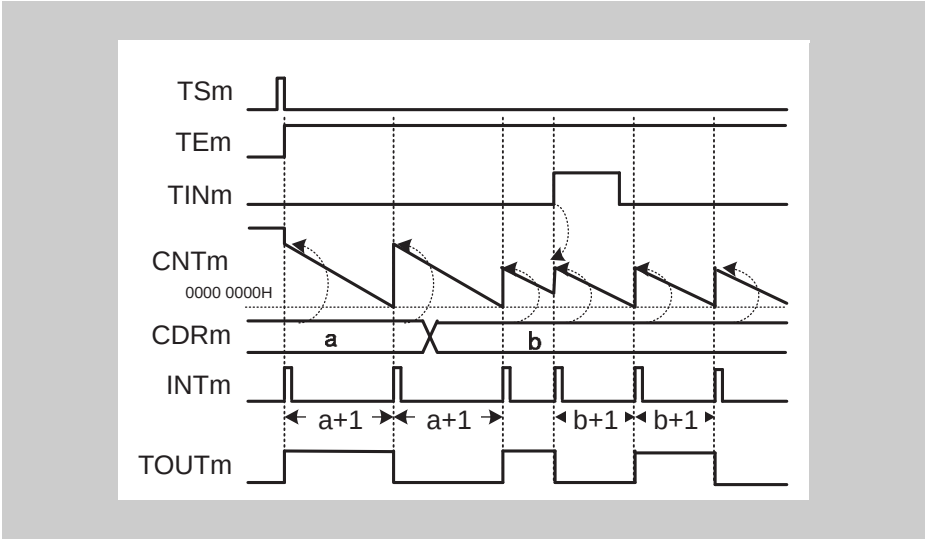



**Figure 15-29 Counter triggered by rising TAUJnTTINm input edge (TAUJnCMURm.TIS[1:0] = $01_B$), TAUJnCMORm.MD0 = 1**

- If a valid TAUJnTTINm input edge is detected, an interrupt is generated which causes TAUJnTTOUTm to toggle. In this example, the valid edge is a rising edge (TAUJnCMURm.TIS[1:0] = $01_B$).

## 15.15 Independent Channel Signal Measurement Fuctions

This chapter describes functions that measure the widths of an individual TAUJnTTINm pulse or the total width of successive TAUJnTTINm pulses. It also describes functions that measure the interval of the signal or that compare the width of a pulse with a reference value.

- *15.15.1 “TAUJnTTINm Input Pulse Interval Measurement Function”*
- *15.15.2 “TAUJnTTINm Input Signal Width Measurement Function”*
- *15.15.3 “Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)”*
- *15.15.4 “TAUJnTTINm Input Period Count Detection Function”*
- *15.15.5 “Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)”*

### 15.15.1 TAUJnTTINm Input Pulse Interval Measurement Function

#### (1) Overview

**Summary** This function captures the count value and uses this value and the overflow bit TAUJnCSRm.OVF to measure the interval of the TAUJnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Capture Mode, refer to *Table 15-21 "TAUJnCMORm settings for TAUJnTTINm Input Pulse Interval Measurement Function" on page 742*
- TAUJnTTOUTm is not used for this function

**Description** The counter is started by setting the channel trigger bit (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation. The counter TAUJnCNTm starts counting up from 0000 $0000_H$. When a valid TAUJnTTINm edge is detected, the value of TAUJnCNTm is captured, transferred to TAUJnCDRm, and an interrupt INTTAUJnIm is generated. The counter resets to 0000 $0000_H$ and subsequently continues operation.

If the counter reaches FFFF $FFFF_H$ before a valid TAUJnTTINm edge is detected, it overflows to 0000 $0000_H$. The counter is reset to 0000 $0000_H$ and subsequently continues operation. The values transferred to TAUJnCDRm and TAUJnCSRm.OVF respectively depend on the values of bits TAUJnCMORm.COS[1:0]:

**Table 15-20 Effects of an overflow**

| TAUJnCMORm.COS[1:0] | When overflow occurs | | When a valid TAUJnTTINm input is then detected | |
|---|---|---|---|---|
| | TAUJnCDRm | TAUJnCSRm.OVF | TAUJnCDRm and TAUJnCNTm | TAUJnCSRm.OVF |
| 00 | Unchanged | 0 | TAUJnCNTm written to TAUJnCDRm | 1 |
| 01 | | 1 | | |
| 10 | Set to FFFF $FFFF_H$ | 0 | TAUJnCNTm set to 0, TAUJnCDRm unchanged | 0 |
| 11 | | 1 | | |

If an overflow is set (TAUJnCSRm.OVF = 1), it can only be cleared by a CPU command that sets TAUJnCSCm.CLOV = 1.

The combination of the value of TAUJnCDRm and TAUJnCSRm.OVF can be used to deduce the interval of the TAUJnTTINm signal. However, if an overflow occurs multiple times before a valid TAUJnTTINm input is detected, the overflow bit TAUJnCSRm.OVF cannot indicate this.

The function can be stopped by setting TAUJnTT.TTm = 1, which in turn sets TAUJnTE.TEm = 0. TAUJnCNTm stops but retains its value. While the function is stopped, TAUJnTTINm input valid edge detection and TAUJnCNTm capture are not performed.

The function can be restarted by setting TAUJnTS.TSm = 1. The counter is reset to 0000 $0000_H$ and subsequently continues operation. The counter can also be forcibly restarted (without stopping it first) by setting TAUJnTS.TSm = 1 during operation.

**Conditions** If the TAUJnCMORm.MD0 bit is set to 0, the first interrupt after a start or restart is not generated. For details refer to *15.10 "TAUJnTTOUTm Output and INTTAUJnIm Generation when Counter Starts or Restarts" on page 718* .

**Note** When TAUJnCMORm.COS[1:0] = $11_B$, the value of TAUJnCNTm is *not* written to TAUJnCDRm when the first valid TAUJnTTINm input edge occurs after an overflow. However, an interrupt is generated.

### (2) Equations

TAUJnTTINm input pulse interval = count clock cycle x [(TAUJnCSRm.OVF x (FFFF $FFFF_H$ + 1)) + TAUJnCDRm capture value + 1]

### (3) Block diagram and general timing diagram

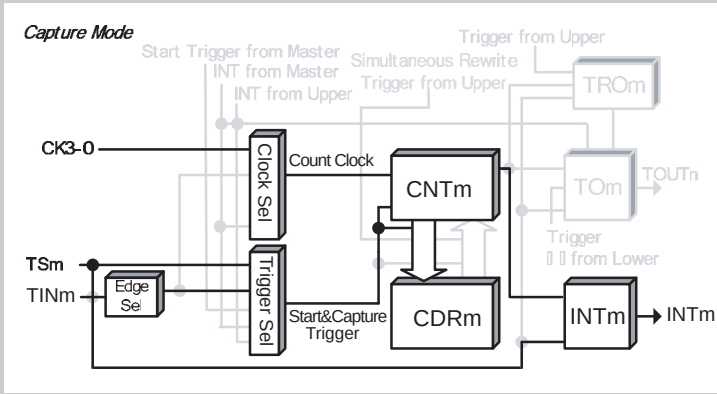



**Figure 15-30 Block diagram for TAUJnTTINm Input Pulse Interval Measurement Function**

The following settings apply to the general timing diagram:

- INTTAUJnIm not generated at operation start (TAUJnCMORm.MD0 = 0)
- Falling edge detection (TAUJnCMURm.TIS[1:0] = $00_B$)
- When a valid TAUJnTTINm input is detected after an overflow TAUJnCDRm is changed and TAUJnCSRm.OVF is set to 1 (TAUJnCMORm.COS[1:0] = $00_B$)

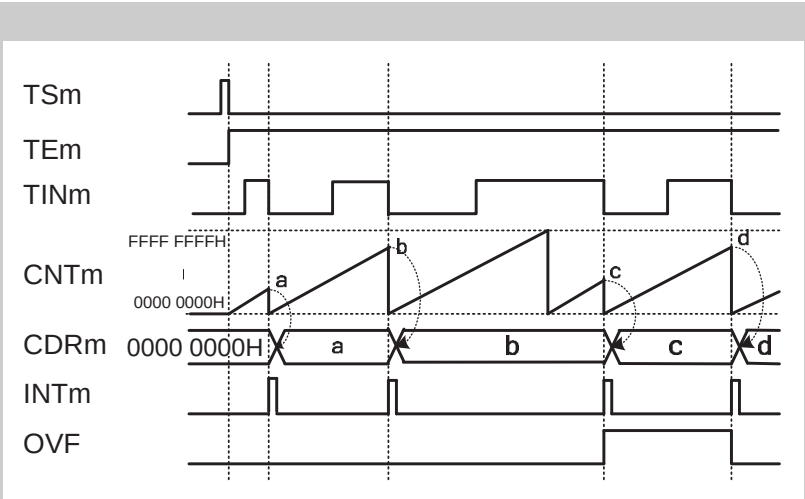



**Figure 15-31 General timing diagram for TAUJnTTINm Input Pulse Interval Measurement Function**

### (4) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-21 TAUJnCMORm settings for TAUJnTTINm Input Pulse Interval Measurement Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid edge of the TAUJnTTINm input signal is the external capture trigger |
| COS[1:0] | See *Table 15-20 "Effects of an overflow" on page 740* |
| MD[4:1] | 0010: Capture Mode |
| MD0 | 0: INTTAUJnIm not generated at operation start<br>1: Generates INTTAUJnIm at operation start |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-22 TAUJnCMURm settings for TAUJnTTINm Input Pulse Interval Measurement Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection (low width measurement)<br>11: Rising and falling edge detection (high width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the TAUJnTTINm Input Pulse Interval Measurement Function. Therefore, these registers must be set to 0.

**Table 15-23 Simultaneous rewrite settings for TAUJnTTINm Input Pulse Interval Measurement Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (5) Operating procedure for TAUJnTTINm Input Pulse Interval Measurement Function

**Table 15-24 Operating procedure for TAUJnTTINm Input Pulse Interval Measurement Function**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-21 "TAUJnCMORm settings for TAUJnTTINm Input Pulse Interval Measurement Function" on page 742* and *Table 15-22 "TAUJnCMURm settings for TAUJnTTINm Input Pulse Interval Measurement Function" on page 742*<br><br>Set the value of the TAUJnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is set to 1 and the counter starts.<br>TAUJnCNTm is cleared to 0000 $0000_H$. INTTAUJnIm is generated when TAUJnCMORm.MD0 is set to 1. |
| During operation | Detection of TAUJnTTINm edges.<br><br>The TAUJnCMURm.TIS[1:0] bits can be changed at any time.<br>The TAUJnCDRm and TAUJnCSRm registers can be read at any time. | TAUJnCNTm starts to count up from 0000 $0000_H$. When a TAUJnTTINm valid edge is detected:<br>• TAUJnCNTm transfers (captures) its value to TAUJnCDRm, and returns to 0000 $0000_H$<br>• INTTAUJnIm is then generated.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm stops and both it and TAUJnCSRm.OVF retain their current values. |

Restart

**(6) Specific timing diagrams: overflow behavior**

**(a) TAUJnCMORm.COS[1:0] = $00_B$**

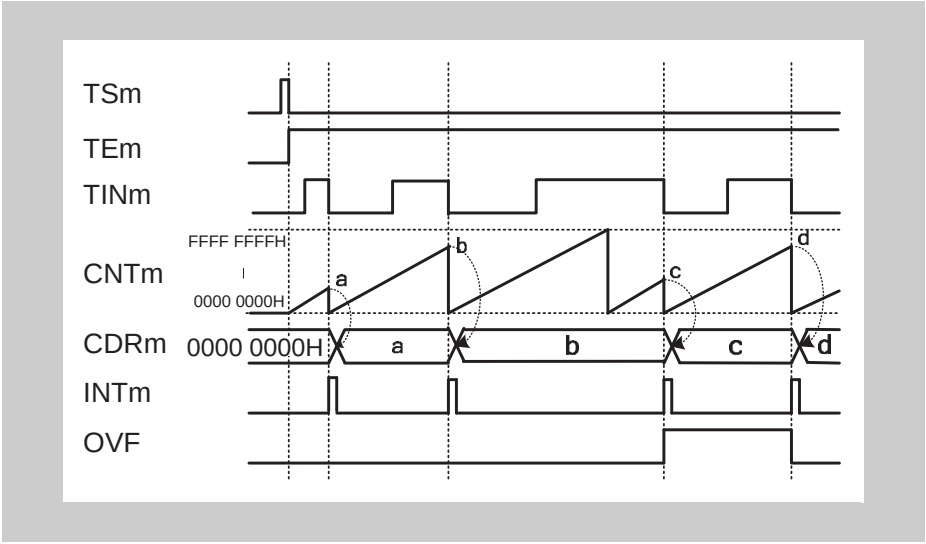



**Figure 15-32 TAUJnCMORm.COS[1:0] = $00_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, the value of TAUJnCDRm remains unchanged and TAUJnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUJnTTINm input edge, the value of TAUJnCNTm is written to TAUJnCDRm and TAUJnCSRm.OVF is set to 1.

**(b) TAUJnCMORm.COS[1:0] = $01_B$**

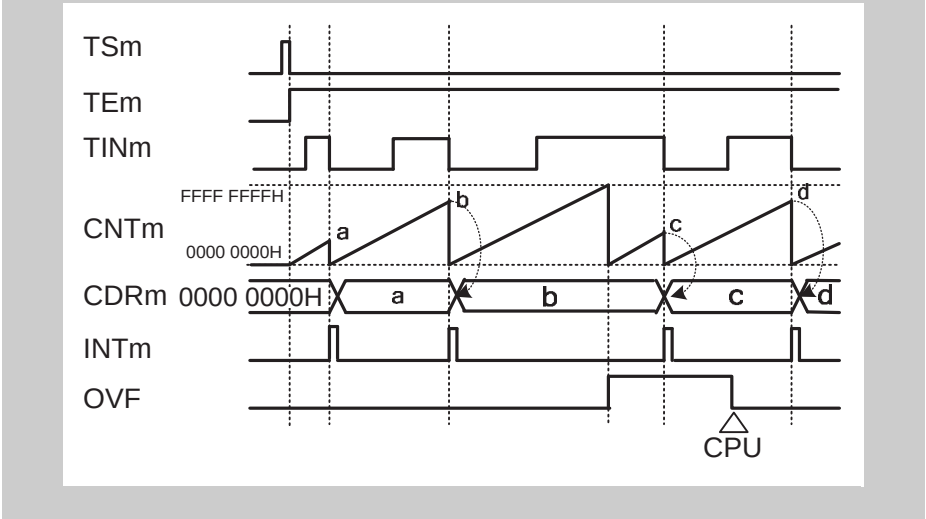



**Figure 15-33 TAUJnCMORm.COS[1:0] = $01_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, the value of TAUJnCDRm remains unchanged and TAUJnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUJnTTINm input edge, the value of TAUJnCNTm is written to TAUJnCDRm.
- TAUJnCSRm.OVF is only cleared by a CPU command.

**(c) TAUJnCMORm.COS[1:0] = $10_B$**

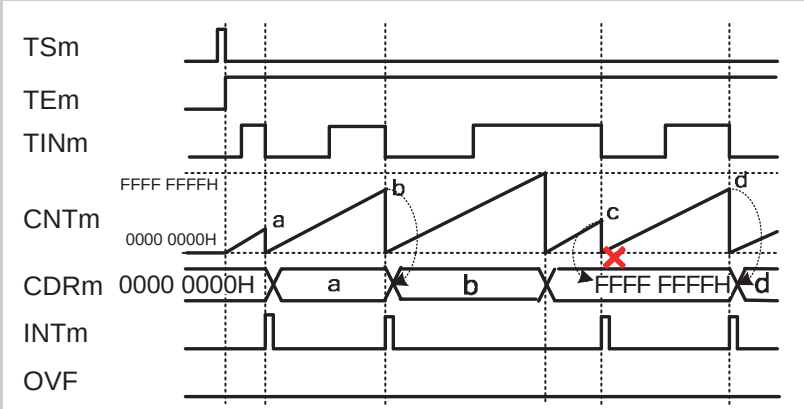



**Figure 15-34 TAUJnCMORm.COS[1:0] = $10_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, TAUJnCDRm is set to FFFF FFFF$_H$ and TAUJnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUJnTTINm input edge, TAUJnCNTm is reset to 0, but TAUJnCDRm and TAUJnCSRm.OVF remain unchanged.
- Thus, the next TAUJnTTINm input valid edge after the overflow is ignored.

**(d) TAUJnCMORm.COS[1:0] = $11_B$**

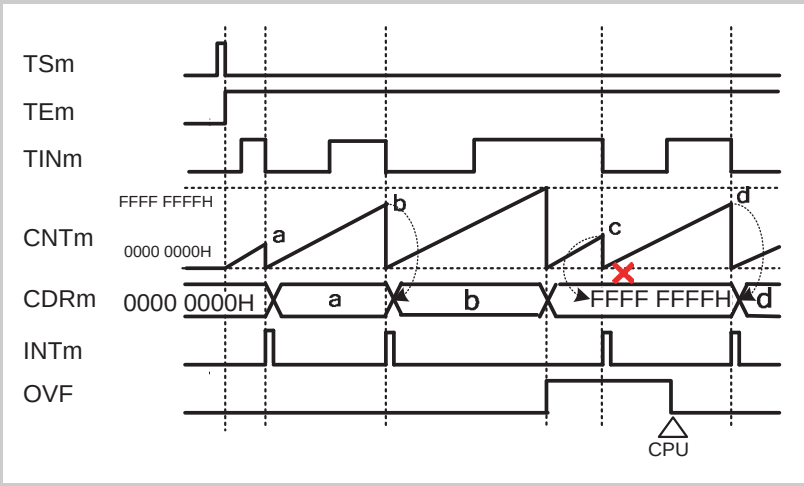



**Figure 15-35 TAUJnCMORm.COS[1:0] = $11_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$00_B$**

- When an overflow occurs, TAUJnCDRm is set to FFFF FFFF$_H$, and TAUJnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUJnTTINm input edge, TAUJnCNTm is reset to 0, but TAUJnCDRm and TAUJnCSRm.OVF remain unchanged.
- Thus, the next TAUJnTTINm input valid edge after the overflow is ignored.
- TAUJnCSRm.OVF is cleared by a CPU command.

## 15.15.2 TAUJnTTINm Input Signal Width Measurement Function

### (1) Overview

**Summary** This function measures the width of a TAUJnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Capture & One Count Mode, refer to *Table 15-26 "TAUJnCMORm settings for TAUJnTTINm Input Signal Width Measurement Function" on page 749*
- TAUJnTTOUTm is not used for this function
- TAUJnCMORm.MD0 must be set to 0

**Description** The counter is started by setting the channel trigger bit (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation. When a valid TAUJnTTINm start edge is detected, the counter TAUJnCNTm starts counting up from 0000 $0000_H$. When a valid TAUJnTTINm stop edge is detected, the value of TAUJnCNTm is captured, transferred to TAUJnCDRm, and an interrupt INTTAUJnIm is generated. The counter retains its value and awaits the next valid TAUJnTTINm input start edge.

If the counter reaches FFFF $FFFF_H$ before a valid TAUJnTTINm stop edge is detected, it overflows. The counter is reset to 0000 $0000_H$ and subsequently continues operation. The values transferred to TAUJnCDRm and TAUJnCSRm.OVF respectively depend on the values of bits TAUJnCMORm.COS[1:0]:

**Table 15-25 Effects of an overflow**

| TAUJnCMORm.COS[1:0] | When overflow occurs | | When a valid TAUJnTTINm input stop edge is detected | |
|---|---|---|---|---|
| | TAUJnCDRm | TAUJnCSRm.OVF | TAUJnCDRm and TAUJnCNTm | TAUJnCSRm.OVF |
| 00 | Unchanged | 0 | TAUJnCNTm written to TAUJnCDRm | 1 |
| 01 | | 1 | | |
| 10 | Set to FFFF $FFFF_H$ | 0 | TAUJnCNTm stops counting TAUJnCDRm unchanged | 0 |
| 11 | | 1 | | |

If an overflow is set (TAUJnCSRm.OVF = 1), it can only be cleared by a CPU command that sets TAUJnCSCm.CLOV = 1.

The combination of the value of TAUJnCDRm and TAUJnCSRm.OVF can be used to deduce the width of the TAUJnTTINm signal. However, if an overflow occurs multiple times before a valid TAUJnTTINm input is detected, the overflow bit TAUJnCSRm.OVF cannot indicate this.

This function cannot be forcibly restarted.

**Note** When TAUJnCMORm.COS[1:0] = $11_B$, the value of TAUJnCNTm is *not* written to TAUJnCDRm when the first valid TAUJnTTINm input edge occurs after an overflow. However, an interrupt is generated.

### (2) Equations

TAUJnTTINm input signal width = count clock cycle x [(TAUJnCSRm.OVF x (FFFF $FFFF_H$ + 1)) + TAUJnCDRm capture value + 1]

### (3) Block diagram and general timing diagram

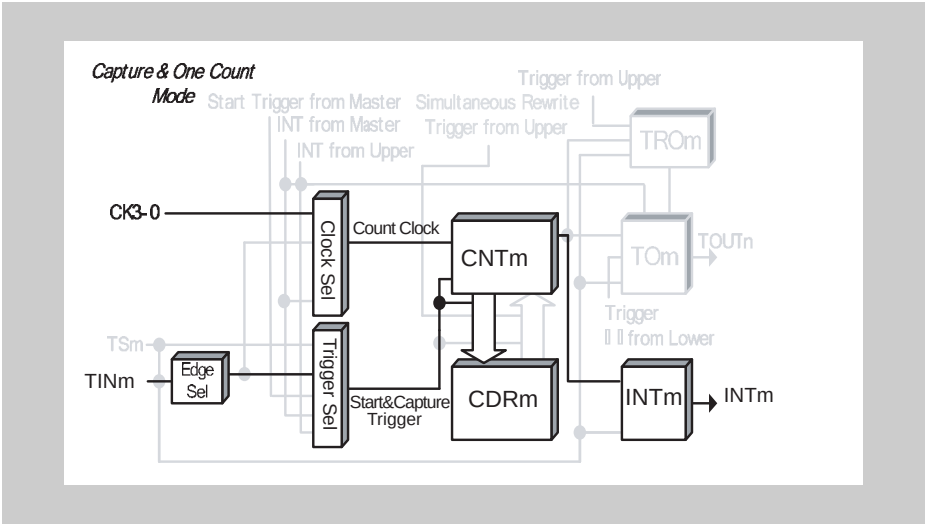



**Figure 15-36 Block diagram for TAUJnTTINm Input Signal Width Measurement Function**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUJnCMURm.TIS[1:0] = $11_B$)
- When a valid TAUJnTTINm input is detected after an overflow TAUJnCDRm is changed and TAUJnCSRm.OVF is set to 1 (TAUJnCMORm.COS[1:0] = $00_B$)

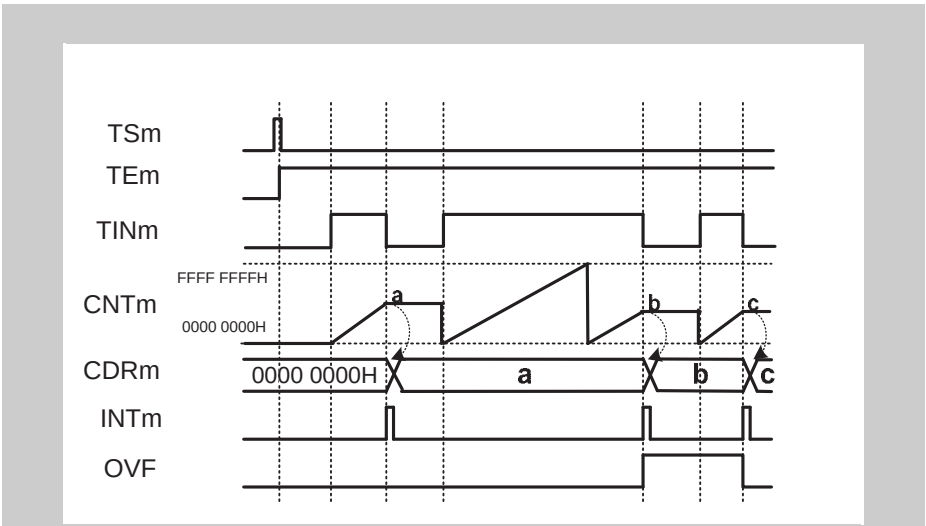



**Figure 15-37 General timing diagram for TAUJnTTINm Input Signal Width Measurement Function**

### (4) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-26 TAUJnCMORm settings for TAUJnTTINm Input Signal Width Measurement Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUJnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | See *Table 15-25 “Effects of an overflow” on page 747* |
| MD[4:1] | 0110: Capture & One Count Mode |
| MD0 | 0: INTTAUJnIm not generated at operation start |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-27 TAUJnCMURm settings for TAUJnTTINm Input Signal Width Measurement Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (low width measurement)<br>11: Rising and falling edge detection (high width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the TAUJnTTINm Input Signal Width Measurement Function. Therefore, these registers must be set to 0.

**Table 15-28 Simultaneous rewrite settings for TAUJnTTINm Input Signal Width Measurement Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (5) Operating procedure for TAUJnTTINm Input Signal Width Measurement Function

**Table 15-29 Operating procedure for TAUJnTTINm Input Signal Width Measurement Function**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-21 "TAUJnCMORm settings for TAUJnTTINm Input Pulse Interval Measurement Function" on page 742* and *Table 15-22 "TAUJnCMURm settings for TAUJnTTINm Input Pulse Interval Measurement Function" on page 742*<br><br>Set the value of the TAUJnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is set to 1 and TAUJnCNTm waits for detection of the TAUJnTTINm start edge.<br>When a TAUJnTTINm start is detected, TAUJnCNTm starts to count up. |
| During operation | Detection of TAUJnTTINm edges.<br><br>The TAUJnCMURm.TIS[1:0] bits can be changed at any time.<br>The TAUJnCDRm and TAUJnCSRm registers can be read at any time. | TAUJnCNTm starts to count up from 0000 $0000_H$. When a TAUJnTTINm valid edge is detected:<br>• TAUJnCNTm transfers (captures) its value to TAUJnCDRm, and retains its value<br>• INTTAUJnIm is then generated.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm stops and both it and TAUJnCSRm.OVF retain their current values. |

Restart

### (6) Specific timing diagrams: overflow behavior

#### (a) TAUJnCMORm.COS[1:0] = $00_B$

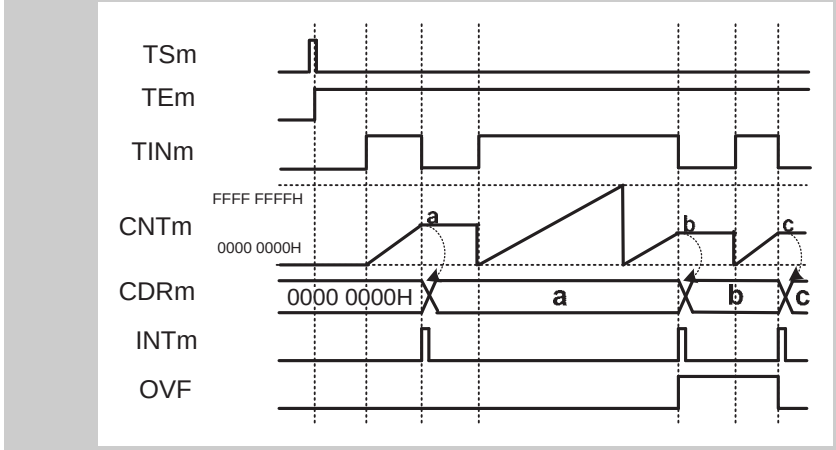



**Figure 15-38 TAUJnCMORm.COS[1:0] = $00_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, the value of TAUJnCDRm remains unchanged and TAUJnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUJnTTINm input edge, the value of TAUJnCNTm is written to TAUJnCDRm and TAUJnCSRm.OVF is set to 1.

#### (b) TAUJnCMORm.COS[1:0] = $01_B$

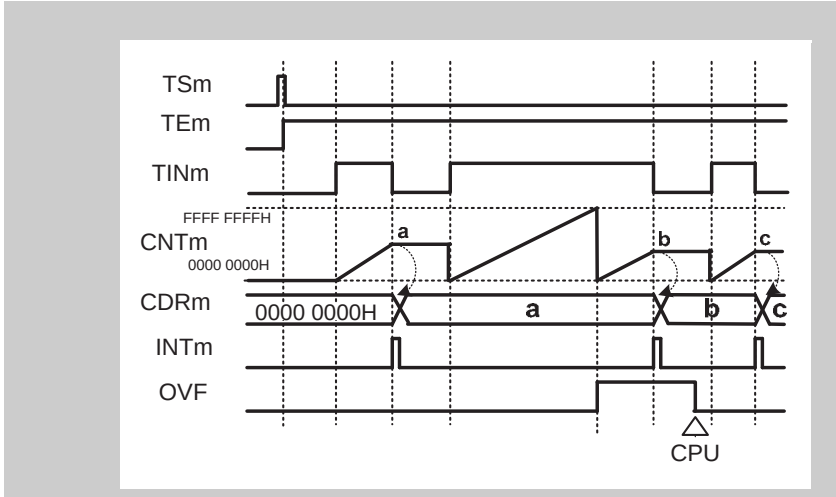



**Figure 15-39 TAUJnCMORm.COS[1:0] = $01_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, the value of TAUJnCDRm remains unchanged and TAUJnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUJnTTINm input edge, the value of TAUJnCNTm is written to TAUJnCDRm.
- TAUJnCSRm.OVF is only cleared by a CPU command.

**(c) TAUJnCMORm.COS[1:0] = $10_B$**

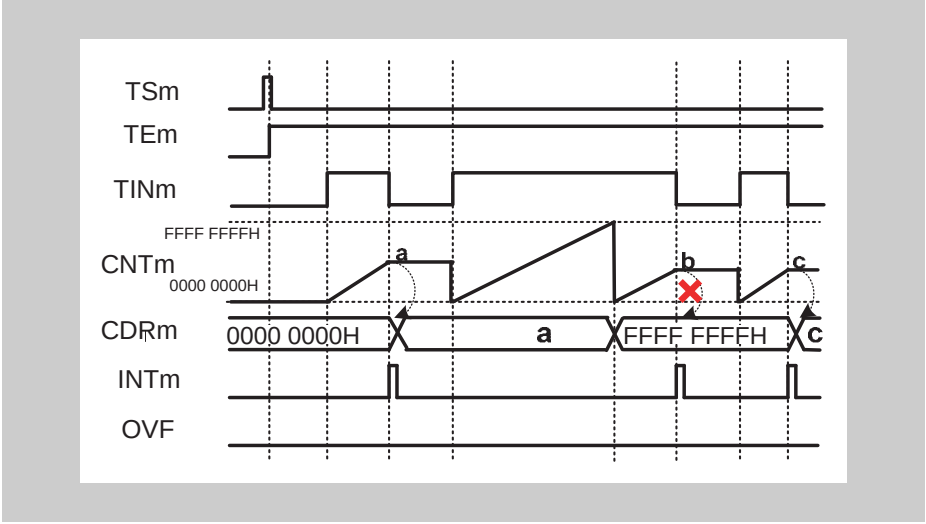



**Figure 15-40 TAUJnCMORm.COS[1:0] = $10_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, TAUJnCDRm is set to FFFF $FFFF_H$ and TAUJnCSRm.OVF remains = 0.
- Upon detection of the next valid TAUJnTTINm input edge, TAUJnCNTm is reset to 0, but TAUJnCDRm and TAUJnCSRm.OVF remain unchanged.
- Thus, the next TAUJnTTINm input valid edge after the overflow is ignored.

**(d) TAUJnCMORm.COS[1:0] = $11_B$**

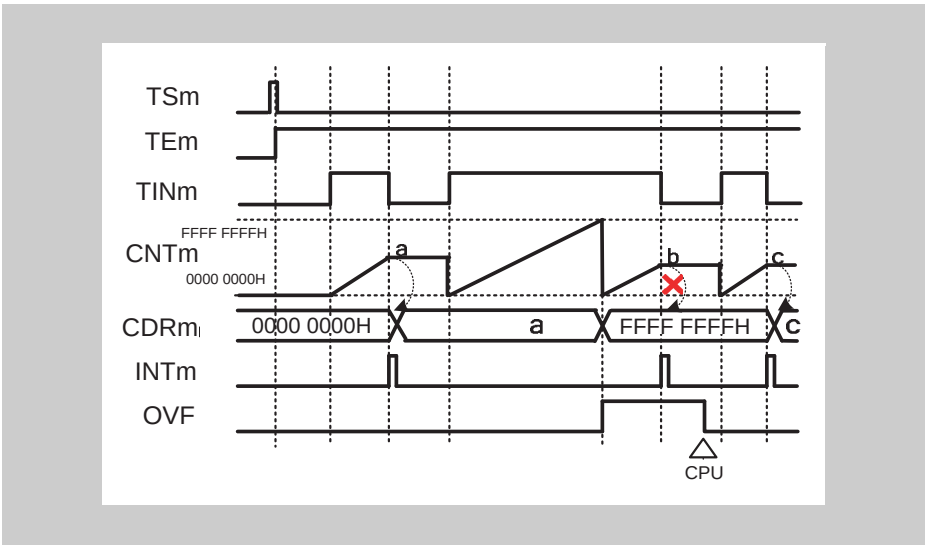



**Figure 15-41 TAUJnCMORm.COS[1:0] = $11_B$, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0]=$11_B$**

- When an overflow occurs, TAUJnCDRm is set to FFFF $FFFF_H$, and TAUJnCSRm.OVF is set to 1.
- Upon detection of the next valid TAUJnTTINm input edge, TAUJnCNTm is reset to 0, but TAUJnCDRm and TAUJnCSRm.OVF remain unchanged.
- Thus, the next TAUJnTTINm input valid edge after the overflow is ignored.
- TAUJnCSRm.OVF is cleared by a CPU command.

### 15.15.3 Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)

**(1) Overview**

**Summary** This function measures the width of an individual TAUJnTTINm input signal. An interrupt is generated if the TAUJnTTINm input width is longer than FFFF $FFFF_H$.

**Prerequisites**
- The operation mode must be set to One Count Mode, refer to *Table 15-30 "TAUJnCMORm settings for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)" on page 756*
- TAUJnTTOUTm is not used for this function
- The value of TAUJnCDRm must be set to FFFF $FFFF_H$.

**Description** The counter is enabled by setting the channel trigger bit (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation.

The counter starts when a valid TAUJnTTINm input start edge is detected. FFFF $FFFF_H$ is written to TAUJnCNTm and the counter starts to count down.

When a valid stop edge is detected, the counter stops and retains the current value.

When the next TAUJnTTINm input start edge is detected, TAUJnCNTm reloads FFFF $FFFF_H$ and starts to count down.

If the counter reaches 0000 $0000_H$ before a stop edge is detected, an interrupt is generated.

**Conditions** The valid start and stop edges are specified by the TAUJnCMURm.TIS[1:0] bits:
- If TAUJnCMURm.TIS[1:0] = $10_B$, the TAUJnTTINm input low width is measured. The start trigger is a falling edge and the stop trigger is a rising edge.
- If TAUJnCMURm.TIS[1:0] = $11_B$, the TAUJnTTINm input high width is measured. The start trigger is a rising edge and the stop trigger is a falling edge.

**Note** The counter cannot be restarted during operation.

### (2) Block diagram and general timing diagram

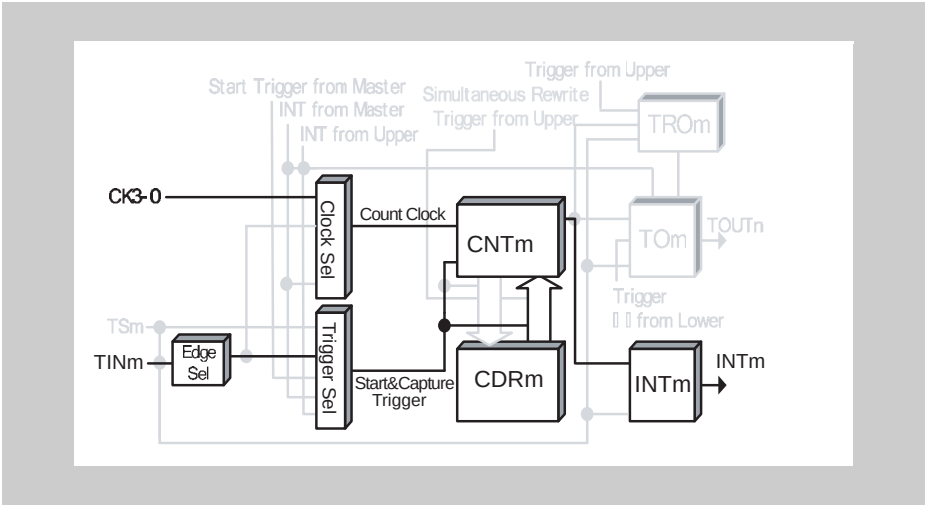



**Figure 15-42 Block diagram for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUJnCMURm.TIS[1:0] = $11_B$)

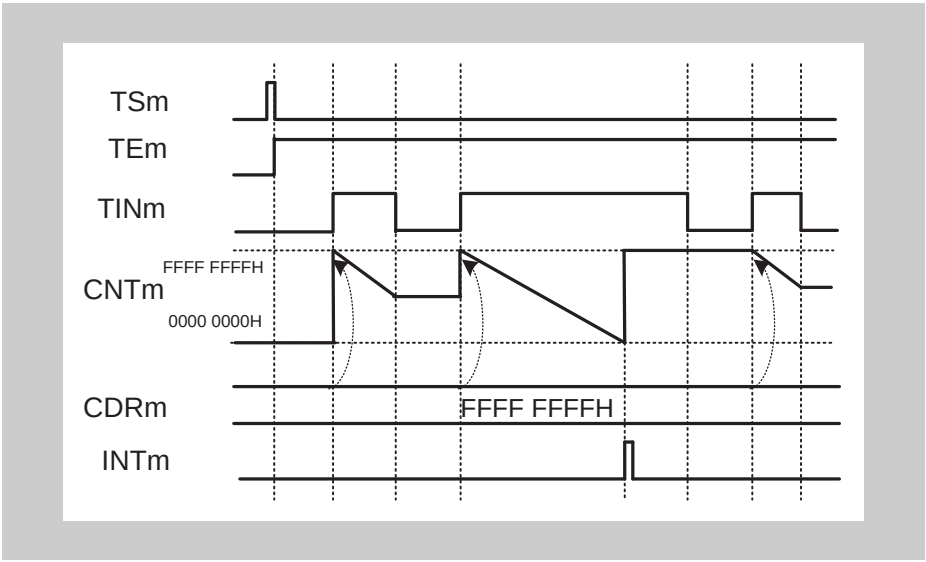



**Figure 15-43 General timing diagram for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)**

### (3) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-30 TAUJnCMORm settings for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUJnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 0: INTTAUJnIm not generated at operation start |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-31 TAUJnCMURm settings Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (Low width measurement)<br>11: Rising and falling edge detection (High width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement). Therefore, these registers must be set to 0.

**Table 15-32 Simultaneous rewrite settings for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (4) Operating procedure for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)

**Table 15-33 Operating procedure for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-30 "TAUJnCMORm settings for Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)" on page 756* and *Table 15-31 "TAUJnCMURm settings Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement)" on page 756*<br><br>Set the value of the TAUJnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUJnTTINm start edge | TAUJnTE.TEm is set to 1 and TAUJnCNTm waits for detection of the start edge.<br><br>When a start edge is detected, TAUJnCNTm loads the TAUJnCDRm value (FFFF $FFFF_H$). |
| During operation | The TAUJnCNTm register can be read at any time.<br><br>Detection of TAUJnTTINm edges. | TAUJnCNTm counts down. When the counter reaches 0000 $0000_H$:<br>• INTTAUJnIm is generated<br>• TAUJnCNTm stops counting at FFFF $FFFF_H$ and waits for a trigger.<br>When a reverse edge of TAUJnTTINm is detected during count operation:<br>• TAUJnCNTm stops counting and waits for a trigger.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm stops and retains its current value. |

Restart

### 15.15.4 TAUJnTTINm Input Period Count Detection Function

#### (1) Overview

**Summary** This function measures the cumulative width of a TAUJnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Capture & Gate Count Mode, refer to *Table 15-34 "TAUJnCMORm settings for TAUJnTTINm Input Period Count Detection Function" on page 761*
- TAUJnTTOUTm is not used for this function

**Description** The counter is enabled by setting the channel trigger bit (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation. The counter awaits a valid TAUJnTTINm input edge.

When a valid TAUJnTTINm input start edge is detected, the counter starts to count from 0000 $0000_H$.

When a valid TAUJnTTINm input stop edge is detected, the current TAUJnCNTm value is written to TAUJnCDRm and an interrupt (INTTAUJnIm) is generated. The counter stops and retains its value until the next valid TAUJnTTINm input start edge is detected.

If the counter reaches FFFF $FFFF_H$ the bit TAUJnCSRm.OVF is set to 1 and the counter restarts from 0000 $0000_H$. The value of TAUJnCSRm.OVF is reset by the CPU by setting TAUJnCSCm.CLOV = 1.

**Note** The input TAUJnTTINm is sampled at the frequency of the operation clock, specified by the TAUJnCMORm.CKS[1:0] bits. As a result, the output cycle of TAUJnTTOUTm has an error of ± 1 operation clock cycle.

**Conditions** The valid start and stop edges are specified by the TAUJnCMURm.TIS[1:0] bits:
- If TAUJnCMURm.TIS[1:0] = $10_B$, the TAUJnTTINm input low width is measured. The start trigger is a falling edge and the stop trigger is a rising edge.
- If TAUJnCMURm.TIS[1:0] = $11_B$, the TAUJnTTINm input high width is measured. The start trigger is a rising edge and the stop trigger is a falling edge.

#### (2) Equations

Cumulative TAUJnTTINm input width =
count clock cycle × ((FFFF $FFFF_H$ × TAUJnCSRm.OVF) + (TAUJnCDRm capture value + 1))

### (3) Block diagram and general timing diagram

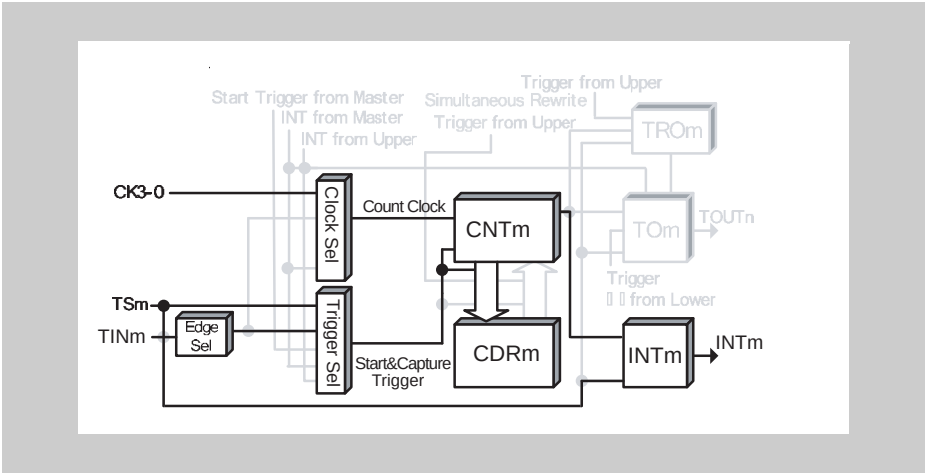



**Figure 15-44 Block diagram for TAUJnTTINm Input Period Count Detection Function**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUJnCMURm.TIS[1:0] = $11_B$)

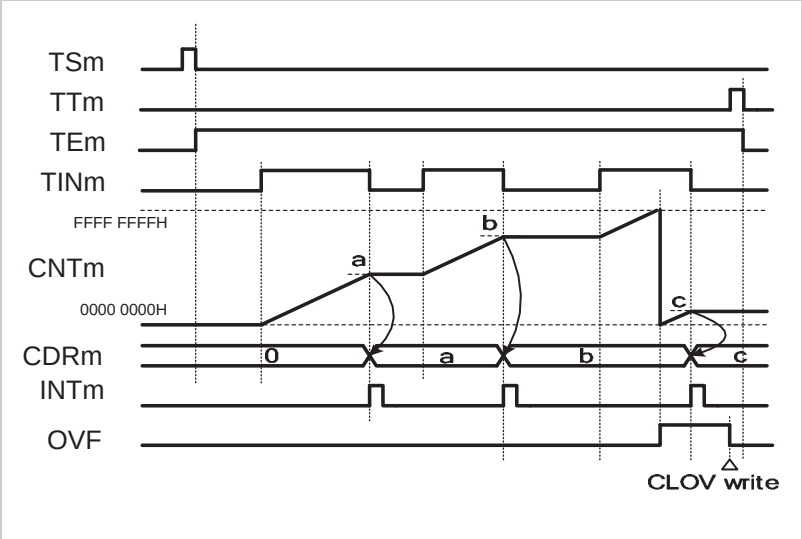



**Figure 15-45 General timing diagram for TAUJnTTINm Input Period Count Detection Function**

### (4) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-34 TAUJnCMORm settings for TAUJnTTINm Input Period Count Detection Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUJnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | 01: Overflow (TAUJnCSRm.OVF) set upon counter overflow and cleared by a CPU instruction |
| MD[4:1] | 1101: Capture & Gate Count Mode |
| MD0 | 0: INTTAUJnIm not generated at operation start |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-35 TAUJnCMURm settings for TAUJnTTINm Input Period Count Detection Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (Low width measurement)<br>11: Rising and falling edge detection (High width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the TAUJnTTINm Input Period Count Detection Function. Therefore, these registers must be set to 0.

**Table 15-36 Simultaneous rewrite settings for TAUJnTTINm Input Period Count Detection Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (5) Operating procedure for TAUJnTTINm Input Period Count Detection Function

**Table 15-37 Operating procedure for TAUJnTTINm Input Period Count Detection Function**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-34 "TAUJnCMORm settings for TAUJnTTINm Input Period Count Detection Function" on page 761* and *Table 15-35 "TAUJnCMURm settings for TAUJnTTINm Input Period Count Detection Function" on page 761*<br><br>Set the value of the TAUJnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUJnTTINm start edge | TAUJnTE.TEm is set to 1 and TAUJnCNTm waits for detection of the TAUJnTTINm start edge.<br><br>When a start edge is detected, TAUJnCNTm is cleared to 0000 $0000_H$ and TAUJnCNTm starts to count up. |
| During operation | Detection of TAUJnTTINm edges.<br><br>The TAUJnCDRm, TAUJnCNTm, and TAUJnCSRm registers can be read at any time.<br>The TAUJnCSC.CLOV bit can be set to 1. | When a TAUJnTTINm start edge (rising edge for high width measurement, falling edge for low width measurement) is detected, TAUJnCNTm starts to count up from the stop value.<br>When TAUJnCNTm detects a capture edge (falling edge for high width measurement, rising edge for low width measurement), it transfers the value to TAUJnCDRm and INTTAUJnIm is generated.<br>Counting stops at the "value transferred to TAUJnCDRm + 1" value and TAUJnCNTm waits for detection of the TAUJnTTINm start edge.<br>If the TAUJnCNTm reaches FFFF $FFFF_H$, the counter overflows and TAUJnCSR.OVF is set to 1.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm stops and it and TAUJnCSRm.OVF retain their current values. |

Restart

### (6) Specific timing diagrams

#### (a) Operation stop and restart

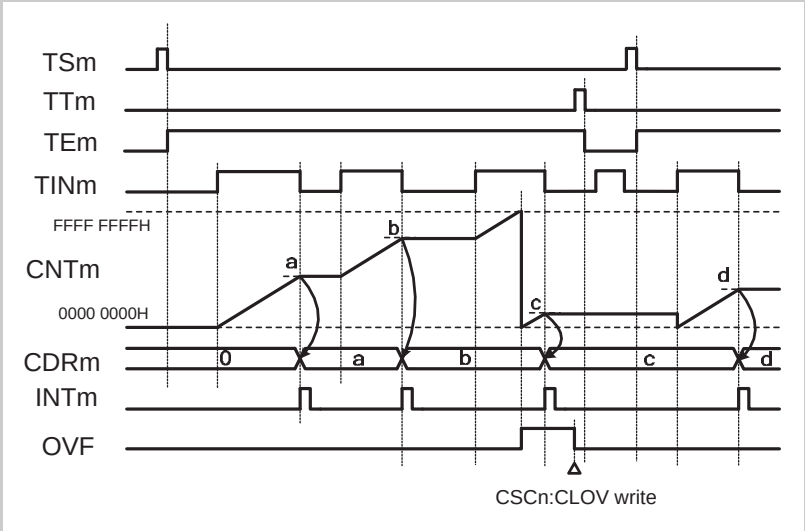



**Figure 15-46 Operation stop and restart, TAUJnCMURm.TIS[1:0] = $11_B$**

- The counter can be stopped by setting TAUJnTT.TTm to 1, which in turn sets TAUJnTE.TEm to 0.
- TAUJnCNTm stops and the current value is retained.
- If the counter is stopped, valid TAUJnTTINm input edges are ignored.
- The counter can be restarted by setting TAUJnTS.TSm to 1. TAUJnCNTm restarts to count from 0000 $0000_H$.

### 15.15.5 Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)

#### (1) Overview

**Summary** This function measures the cumulative width of a TAUJnTTINm input signal. An interrupt is generated if the cumulative TAUJnTTINm input width is longer than FFFF FFFF$_H$.

**Prerequisites**

- The operation mode must be set to Gate Count Mode, refer to *Table 15-38 "TAUJnCMORm settings for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)" on page 767*
- TAUJnTTOUTm is not used for this function
- The value of TAUJnCDRm must be set to FFFF FFFF$_H$.

**Description** The counter is enabled by setting the channel trigger bit (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation.

The counter starts when a valid TAUJnTTINm input start edge is detected. FFFF FFFF$_H$ is written to TAUJnCNTm and the counter starts to count down.

When a valid stop edge is detected, the counter stops and retains the current value. The counter awaits the next TAUJnTTINm input start edge and then continues to count down from the current value.

When the counter reaches 0000 0000$_H$ an interrupt is generated. FFFF FFFF$_H$ is written to TAUJnCNTm and the counter continues to count down until a TAUJnTTINm input stop edge is detected.

**Conditions** The valid start and stop edges are specified by the TAUJnCMURm.TIS[1:0] bits:

- If TAUJnCMURm.TIS[1:0] = 10$_B$, the TAUJnTTINm input low width is measured. The start trigger is a falling edge and the stop trigger is a rising edge.
- If TAUJnCMURm.TIS[1:0] = 11$_B$, the TAUJnTTINm input high width is measured. The start trigger is a rising edge and the stop trigger is a falling edge.

**Note** The counter cannot be restarted during operation.

### (2) Block diagram and general timing diagram

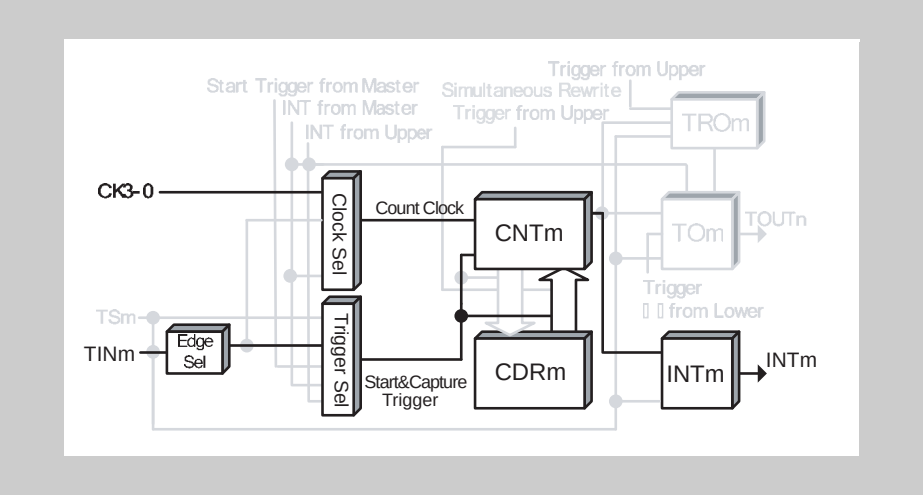



**Figure 15-47 Block diagram for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)**

The following settings apply to the general timing diagram:

- Rising and falling edge detection = high width measurement (TAUJnCMURm.TIS[1:0] = $11_B$)

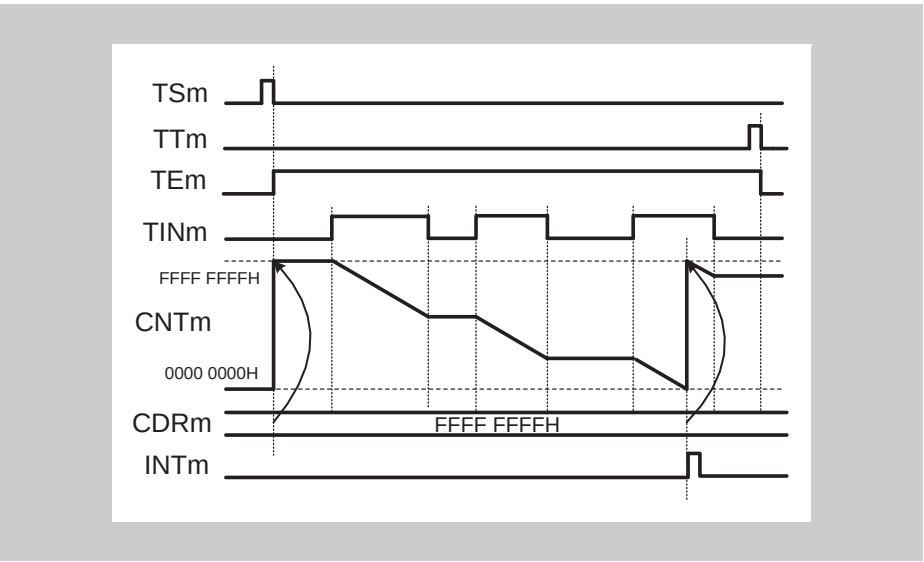



**Figure 15-48 General timing diagram for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)**

### (3) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-38 TAUJnCMORm settings for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 010: Valid edge of the TAUJnTTINm input signal is the external start trigger and the reverse edge is the stop trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 1100: Gate Count Mode |
| MD0 | 0: INTTAUJnIm not generated at operation start |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-39 TAUJnCMURm settings for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 10: Rising and falling edge detection (Low width measurement)<br>11: Rising and falling edge detection (High width measurement) |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection). Therefore, these registers must be set to 0.

**Table 15-40 Simultaneous rewrite settings for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (4) Operating procedure for Overflow Interrupt Output Function (during TAUJnTTINm input period count detection)

**Table 15-41 Operating procedure for Overflow Interrupt Output Function (during TAUJnTTINm input period count detection)**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-38 "TAUJnCMORm settings for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)" on page 767* and *Table 15-39 "TAUJnCMURm settings for Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection)" on page 767*<br><br>Set the value of the TAUJnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0.<br><br>Detection of TAUJnTTINm start edge | TAUJnTE.TEm is set to 1 and TAUJnCNTm waits for detection of the start edge.<br><br>When a start edge is detected, TAUJnCNTm loads the TAUJnCDRm value (FFFF $FFFF_H$). |
| During operation | The TAUJnCNTm register can be read at all times. | TAUJnCNTm counts down. When the counter reaches 0000 $0000_H$:<br>• INTTAUJnIm is generated<br>• TAUJnCNTm loads the TAUJnCDRm value (FFFF $FFFF_H$) and continues to count down.<br>When a reverse edge of TAUJnTTINm is detected during count operation:<br>• TAUJnCNTm counts down from the stop value.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm stops and retains its current value. |

Restart

## 15.16 Other Independent Channel Functions

This chapter describes a function that generates an interrupt when a certain number of TAUJnTTINm pulses has occurred, a function that divides the frequency of TAUJnTTINm, and a function that measures the duration between the function start and a TAUJnTTINm input signal:

- *15.16.1 “TAUJnTTINm Input Position Detection Function”*

### 15.16.1 TAUJnTTINm Input Position Detection Function

#### (1) Overview

**Summary** This function measures the duration between the function start and a TAUJnTTINm input signal.

**Prerequisites**
- The operation mode must be set to Count Capture Mode, refer to *Table 15-42 "TAUJnCMORm settings for TAUJnTTINm Input Position Detection Function" on page 773*
- TAUJnTTOUTm is not used for this function

**Description** The counter is enabled by setting the channel trigger bit (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation. The counter starts to count from 0000 $0000_H$. When a valid TAUJnTTINm input stop edge is detected, the current TAUJnCNTm value is written to TAUJnCDRm and an interrupt (INTTAUJnIm) is generated. The counter continues to count from the current value until the next valid TAUJnTTINm input edge is detected.

When the counter reaches FFFF $FFFF_H$, the bit TAUJnCSRm.OVF is set to 1 and the counter restarts from 0000 $0000_H$. The value of TAUJnCSRm.OVF is reset by the CPU by TAUJnCSCm.CLOV = 1.

**Note** The input TAUJnTTINm is sampled at the frequency of the operation clock, specified by the TAUJnCMORm.CKS[1:0] bits. As a result, the output cycle of TAUJnTTOUTm has an error of ± 1 operation clock cycle.

**Conditions** If the TAUJnCMORm.MD0 bit is set to 0, the first interrupt after a start or restart is not generated. For details refer to *15.10 "TAUJnTTOUTm Output and INTTAUJnIm Generation when Counter Starts or Restarts" on page 718* .

#### (2) Equations

Function duration at a TAUJnTTINm input pulse =

count clock cycle × [(FFFF $FFFF_H$+1 × TAUJnCSRm.OVF) + (TAUJnCDRm capture value + 1)]

### (3) Block diagram and general timing diagram

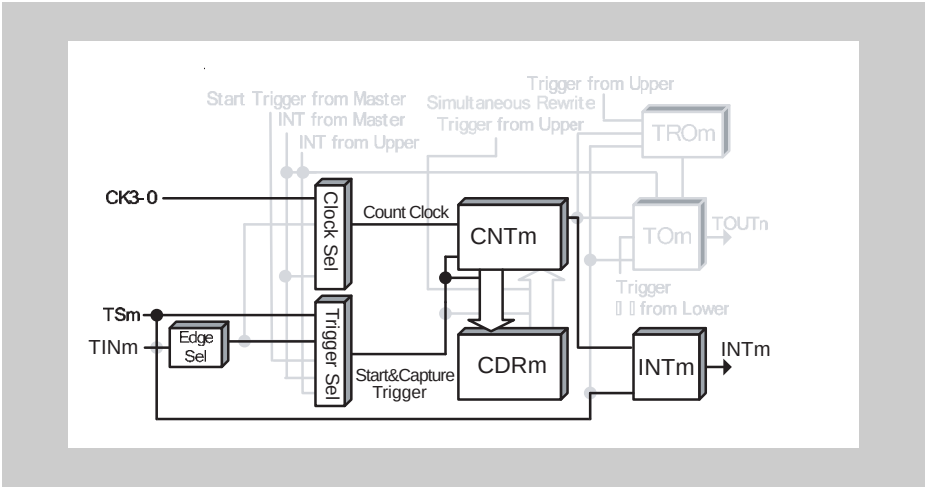



**Figure 15-49 Block diagram for TAUJnTTINm Input Position Detection Function**

The following settings apply to the general timing diagram:

- INTTAUJnIm not generated at operation start (TAUJnCMORm.MD0 = 0)
- Falling edge detection (TAUJnCMURm.TIS[1:0] = $00_B$)

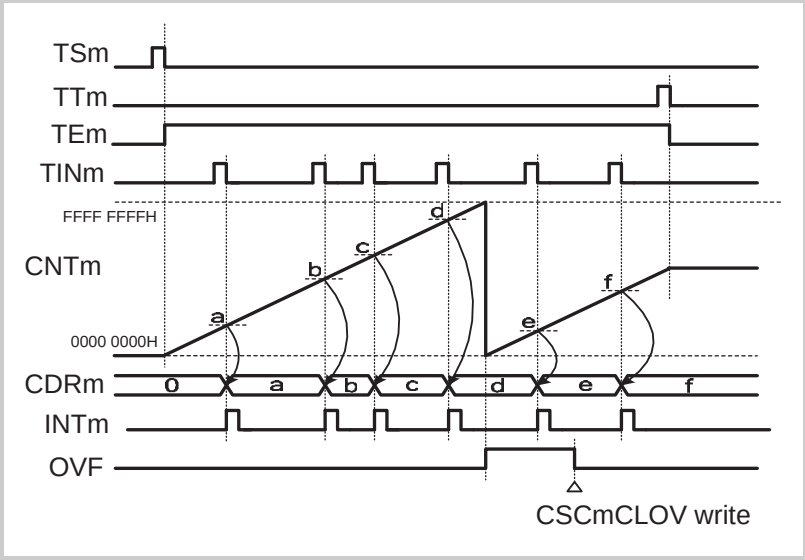



**Figure 15-50 General timing diagram for TAUJnTTINm Input Position Detection Function**

### (4) Register settings

#### (a) TAUJnCMORm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-42 TAUJnCMORm settings for TAUJnTTINm Input Position Detection Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Operation clock = CK0<br>01: Operation clock = CK1<br>10: Operation clock = CK2<br>11: Operation clock = CK3 |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Not used, so set to 0 |
| STS[2:0] | 001: Valid TAUJnTTINm input edge signal is used as the external capture trigger |
| COS[1:0] | 01: Overflow (TAUJnCSRm.OVF) set upon counter overflow and cleared by a CPU instruction |
| MD[4:1] | 1011: Count Capture Mode |
| MD0 | 0: INTTAUJnIm not generated at operation start<br>1: Generates INTTAUJnIm at operation start |

#### (b) TAUJnCMURm

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-43 TAUJnCMURm settings for TAUJnTTINm Input Position Detection Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Falling edge detection<br>01: Rising edge detection<br>10: Rising and falling edge detection |

**(c) Channel output mode**

The channel output mode is not used by this function. However, it can be used in Direct Channel Output Mode.

**(d) Simultaneous rewrite**

The simultaneous rewrite registers (TAUJnRDE and TAUJnRDM) cannot be used with the TAUJnTTINm Input Position Detection Function. Therefore, these registers must be set to 0.

**Table 15-44 Simultaneous rewrite settings for TAUJnTTINm Input Position Detection Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 0: Disables simultaneous rewrite |
| RDM.RDMm | 0: When simultaneous rewrite is disabled (TAUJnRDE.RDEm = 0), set these bits to 0 |

### (5) Operating procedure for TAUJnTTINm Input Position Detection Function

**Table 15-45 Operating procedure for TAUJnTTINm Input Position Detection Function**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Set the TAUJnCMORm register and TAUJnCMURm registers as described in *Table 15-42 "TAUJnCMORm settings for TAUJnTTINm Input Position Detection Function" on page 773* and *Table 15-43 "TAUJnCMURm settings for TAUJnTTINm Input Position Detection Function" on page 773*<br><br>Set the value of the TAUJnCDRm register | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm to 1.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is set to 1 and the counter starts.<br>INTTAUJnIm is generated when TAUJnCMORm.MD0 is set to 1. |
| During operation | The TAUJnCMURm.TIS[1:0] bits can be changed at any time.<br>The TAUJnCDRm and TAUJnCSRm registers can be read at any time. | TAUJnCNTm starts to count up from 0000 $0000_H$.<br>When a TAUJnTTINm valid edge is detected:<br>• TAUJnCNTm transfers (captures) its value to TAUJnCDRm<br>• TAUJnTTINm is output.<br>• The counter value is not cleared to 0000 $0000_H$ and TAUJnCNTm continues count operation.<br>Afterwards, this procedure is repeated. |
| Stop operation | Set TAUJnTT.TTm to 1.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm stops and both it and TAUJnCSRm.OVF retain their current values. |

Restart

### (6) Specific timing diagrams

#### (a) Operation stop and restart

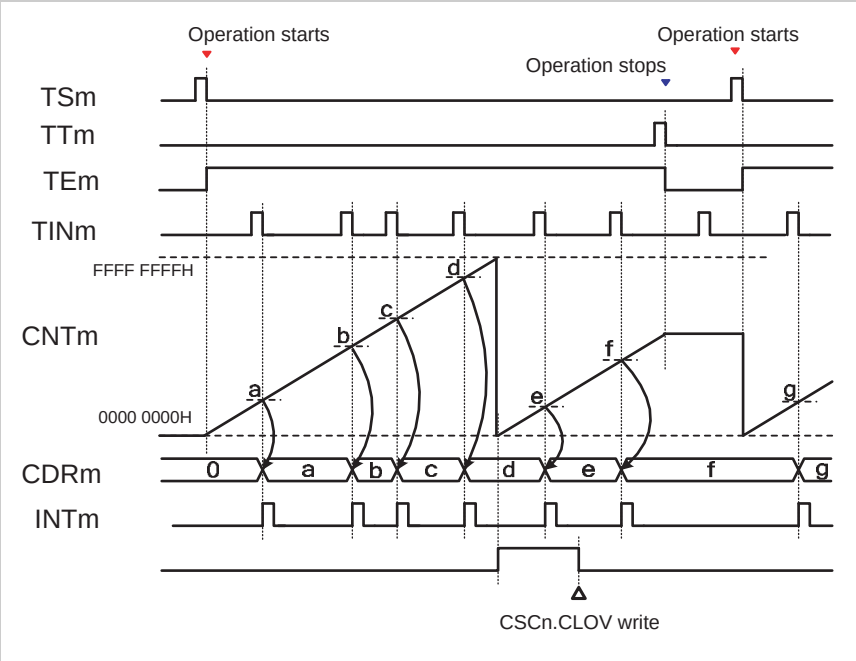



**Figure 15-51 Operation stop and restart, TAUJnCMORm.MD0 = 0, TAUJnCMURm.TIS[1:0] = 00**

- The counter can be stopped by setting TAUJnTT.TTm to 1, which in turn sets TAUJnTE.TEm to 0.
- TAUJnCNTm stops and the current value is retained.
- If the counter is stopped, valid TAUJnTTINm input edges are ignored.
- The counter can be restarted by setting TAUJnTS.TSm to 1. TAUJnCNTm restarts to count from 0000 $0000_H$.

## 15.17 Synchronous PWM Signal Functions Triggered at Regular Intervals

This chapter describes a function that generates PWM signals at regular intervals. For a general overview of synchronous channel operation, see *15.3 “Functional Description” on page 699* .

- *15.17.1 “PWM Output Function”*

### 15.17.1 PWM Output Function

#### (1) Overview

**Summary** This function generates multiple PWM outputs by using a master and multiple slave channels. It enables the pulse cycle (frequency) and the pulse width (duration) of the TAUJnTTOUTm to be set. The pulse cycle is set in the master channel. The pulse width is set in the slave channel.

**Prerequisites**

- Two channels
- The operation mode of the master channel must be set to Interval Timer Mode, refer to *Table 15-46 “TAUJnCMORm settings for the master channel of the PWM Output Function” on page 781*
- The operation mode of the slave channel(s) must be set to One Count Mode, refer to *Table 15-49 “TAUJnCMORm settings for the slave channel of the PWM Output Function” on page 783*
- TAUJnTTOUTm is not used for the master channel of this function
- The channel output mode of the slave channel(s) must be set to Synchronous Channel Output Mode 1 ( *15.8 “Channel Output Modes” on page 711* )

**Description** The counters are started by setting the channel trigger bits (TAUJnTS.TSm) to 1. This in turn sets TAUJnTE.TEm = 1, enabling count operation. The current value of TAUJnCDRm is written to TAUJnCNTm and the counters start to count down from these values. INTTAUJnIm is generated on the master channel and TAUJnTTOUTm (slave) toggles.

- Master channel:

  When the counter of the master channel reaches 0000 $0000_H$, pulse cycle time has elapsed and INTTAUJnIm is generated. The counter reloads the TAUJnCDRm value and counts down.

- Slave channel(s)

  The INTTAUJnIm of the master channel triggers the counter of the slave channel(s). The current value of TAUJnCDRm (slave) is written to TAUJnCNTm (slave) and the counter starts to count down from this value. The TAUJnTTOUTm signal is set.

  When the counter reaches 0000 $0000_H$, i.e. duty time has elapsed, INTTAUJnIm is generated and the TAUJnTTOUTm signal is reset. The counter returns to FFFF $FFFF_H$ and awaits the next INTTAUJnIm of the master channel, and thus the start of the next pulse cycle.

The counter can be stopped by setting TAUJnTT.TTm to 1 for the master and slave channel(s), which in turn sets TAUJnTE.TEm to 0. TAUJnCNTm and TAUJnTTOUTm of master and slave channel(s) stop but retain their values. The counters can be restarted by setting TAUJnTS.TSm to 1.

**Note** If a forced restart is executed during operation, TAUJnTTOUTm is not output as a PWM signal.

**Conditions** Simultaneous rewrite can be used with this function. Please refer to *15.7 “Simultaneous Rewrite” on page 706*

### (2) Equations

Pulse cycle = (TAUJnCDRm (master) + 1) x count clock cycle

Duty cycle [%] = (TAUJnCDRm (slave) / (TAUJnCDRm (master) + 1)) x 100

– Duty cycle = 0 %

  TAUJnCDRm (slave) = 0000 $0000_H$

– Duty cycle = 100 %

  TAUJnCDRm (slave) ≥ TAUJnCDRm (master) + 1

### (3) Block diagram and general timing diagram

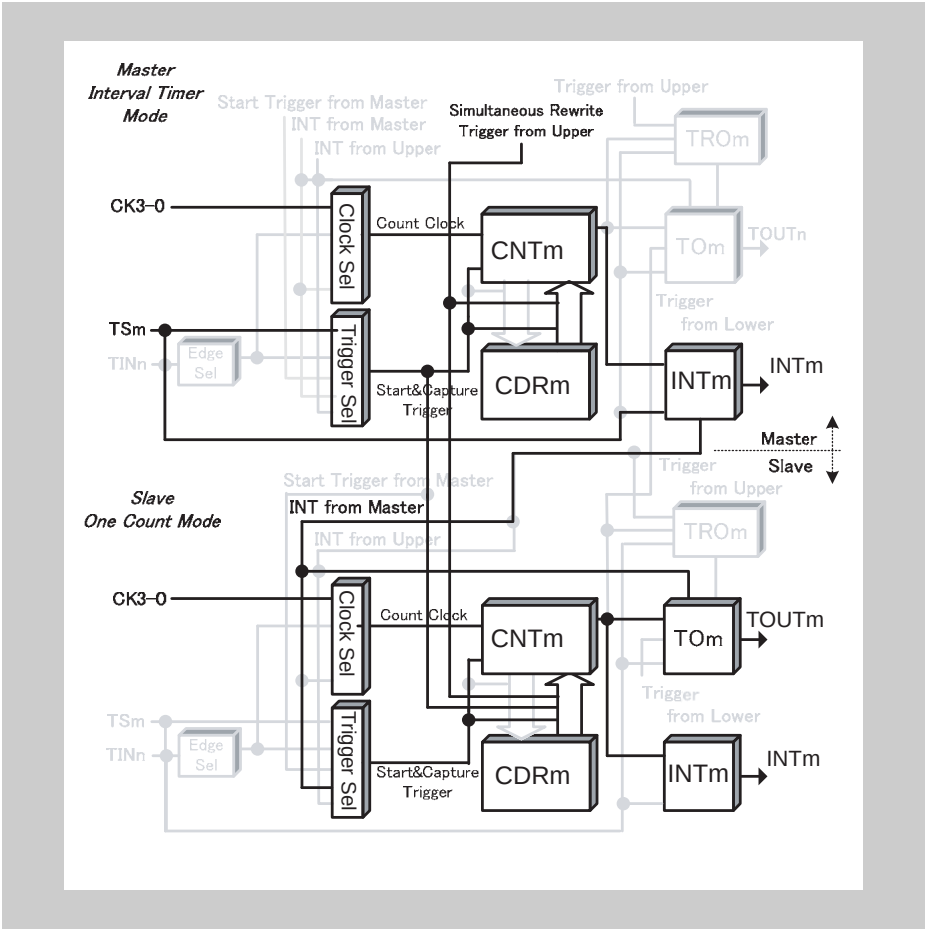



**Figure 15-52 Block diagram for PWM Output Function**

The following settings apply to the general timing diagram:

- Slave channel: Positive logic (TAUJnTOL.TOLm = 0)

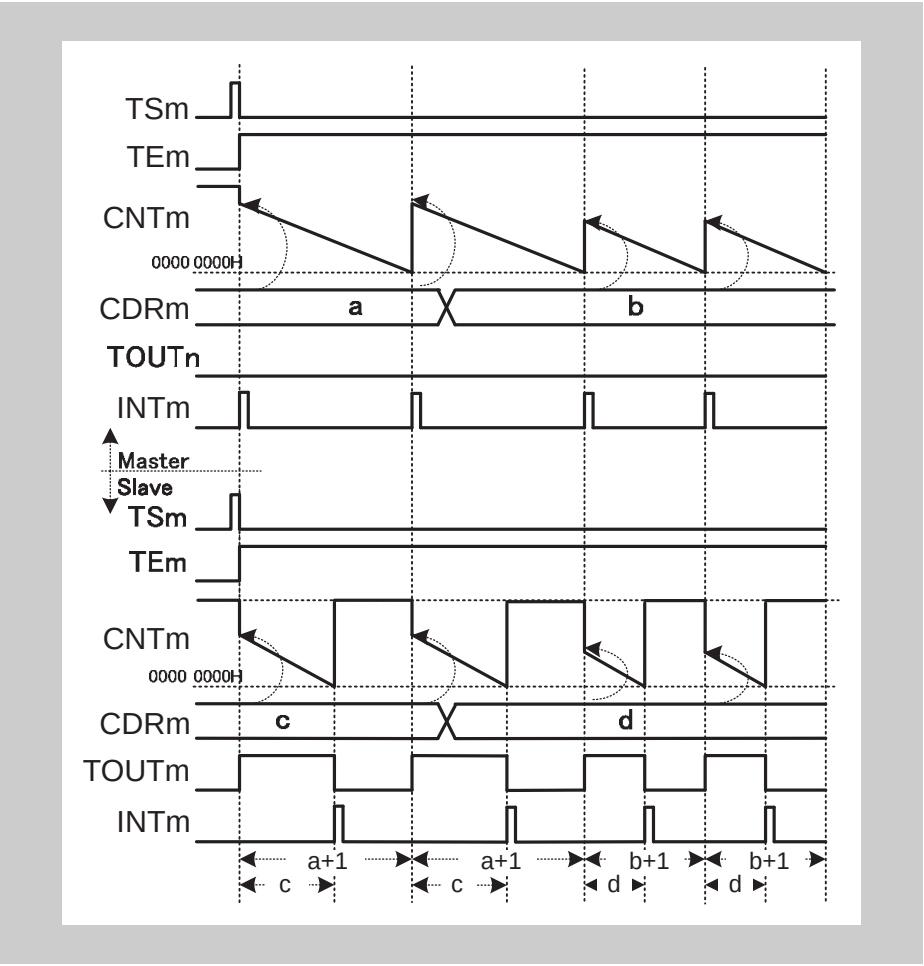



**Figure 15-53 General timing diagram for PWM Output Function**

**Note** The interval between the slave channel starting to count and an interrupt being generated is the value of corresponding TAUJnCDRm, whereas for the master channel the interval is the corresponding TAUJnCDRm + 1.

### (4) Register settings for the master channel

#### (a) TAUJnCMORm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-46 TAUJnCMORm settings for the master channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 1: Channel is master channel |
| STS[2:0] | 000: Counter triggered by software trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0000: Interval Timer Mode |
| MD0 | 1: Generates INTTAUJnIm at operation start |

#### (b) TAUJnCMURm for the master channel

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-47 TAUJnCMURm settings for the master channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

**(c) Channel output mode for the master channel**

The channel output mode is not used by this function. However, it can be used by other functions or in Direct Channel Output Mode.

**(d) Simultaneous rewrite for the master channel**

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 15-48 Simultaneous rewrite settings for the master channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |

### (5) Register settings for the slave channel(s)

#### (a) TAUJnCMORm for the slave channel(s)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:1] | | | | MD0 |

**Table 15-49 TAUJnCMORm settings for the slave channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| CKS[1:0] | 00: Prescaler output CK0<br>01: Prescaler output CK1<br>10: Prescaler output CK2<br>11: Prescaler output CK3<br>The value of the CKS[1:0] bit of the master and slave channel(s) must be identical. |
| CCS[1:0] | 00: Operation clock is used as the count clock |
| MAS | 0: Channel is a slave channel |
| STS[2:0] | 100: INTTAUJnIm of the master channel is the start trigger |
| COS[1:0] | 00: Not used, so set to 00 |
| MD[4:1] | 0100: One Count Mode |
| MD0 | 1: Generates INTTAUJnIm and toggles TAUJnTTOUTm at operation start |

#### (b) TAUJnCMURm for the slave channel(s)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | | | | | | | | | | | | | | TIS[1:0] | |

**Table 15-50 TAUJnCMURm settings for the slave channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| TIS[1:0] | 00: Not used so set to 00 |

### (c) Channel output mode for the slave channel(s)

**Table 15-51 Control bit settings for Independent Channel Output Mode 1**

| Bit name | Setting |
|---|---|
| TOEm | 1: Disables Direct Channel Output Mode |
| TOMm | 1: Synchronous channel operation |
| TOCm | 0: Operation mode 1 |
| TOLm | 0: Positive logic<br>1: Inverted logic |

### (d) Simultaneous rewrite for the slave channel(s)

The simultaneous rewrite settings of the master and slave channel must be identical.

**Table 15-52 Simultaneous rewrite settings for the slave channel of the PWM Output Function**

| Bit name | Setting |
|---|---|
| RDE.RDEm | 1: Enables simultaneous rewrite |
| RDM.RDMm | 0: The simultaneous rewrite trigger signal is generated when the master channel starts counting |

### (6) Operating procedure for PWM Output Function

**Table 15-53 Operating procedure for PWM Output Function**

| | Operation | Status of TAUJn |
|---|---|---|
| Initial channel setting | Master channel: set the TAUJnCMORm and TAUJnCMURm registers and the channel output mode as described in *4 "Register settings for the master channel" on page 781*<br><br>Slave channel: set the TAUJnCMORm and TAUJnCMURm registers and the channel output mode as described in *5 "Register settings for the slave channel(s)" on page 783*<br><br>Set the values of the TAUJnCDRm registers of all channels | Channel operation is stopped.<br>(A clock is supplied, and a small amount of power is consumed.) |
| Start operation | Set TAUJnTS.TSm of the master and slave channels to 1 simultaneously.<br>TAUJnTS.TSm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm (master and slave channels) is set to 1 and the counters of the master and slave channels start.<br>INTTAUJnIm is generated on the master channel and TAUJnTTOUTm (slave) is set. |
| During operation | TAUJnCDRm can be changed at any time.<br>TAUJnCNTm and TAUJnRSF.RSFm can be read at any time.<br><br>TAUJnRDT.RDTm can be changed during operation. | TAUJnCNTm of the master channel loads TAUJnCDRm and counts down. When the counter reaches 0000 $0000_H$:<br>• INTTAUJnIm (master) is generated<br>• TAUJnCNTm (master) reloads the TAUJnCDRm value and continues count operation<br>• TAUJnCNTm (slave) reloads the TAUJnCDRm value and counts down<br>• TAUJnTTOUTm (slave) is set<br>When TAUJnCNTm (slave) reaches 0000 $0000_H$:<br>• INTTAUJnIm (slave) is generated<br>• TAUJnTTOUTm (slave) is reset |
| Stop operation | Set TAUJnTT.TTm of the master and slave channels to 1 simultaneously.<br>TAUJnTT.TTm is a trigger bit, so it is automatically cleared to 0. | TAUJnTE.TEm is cleared to 0 and the counter stops.<br>TAUJnCNTm and TAUJnTTOUTm stop and retain their current values.<br><br>When TAUJnTOE.TOEm is 0, TAUJnTTOUTm output is initialized to the value set by TAUJnTO.TOm. |

Restart

### (7) Specific timing diagrams

#### (a) Duty cycle = 0 %

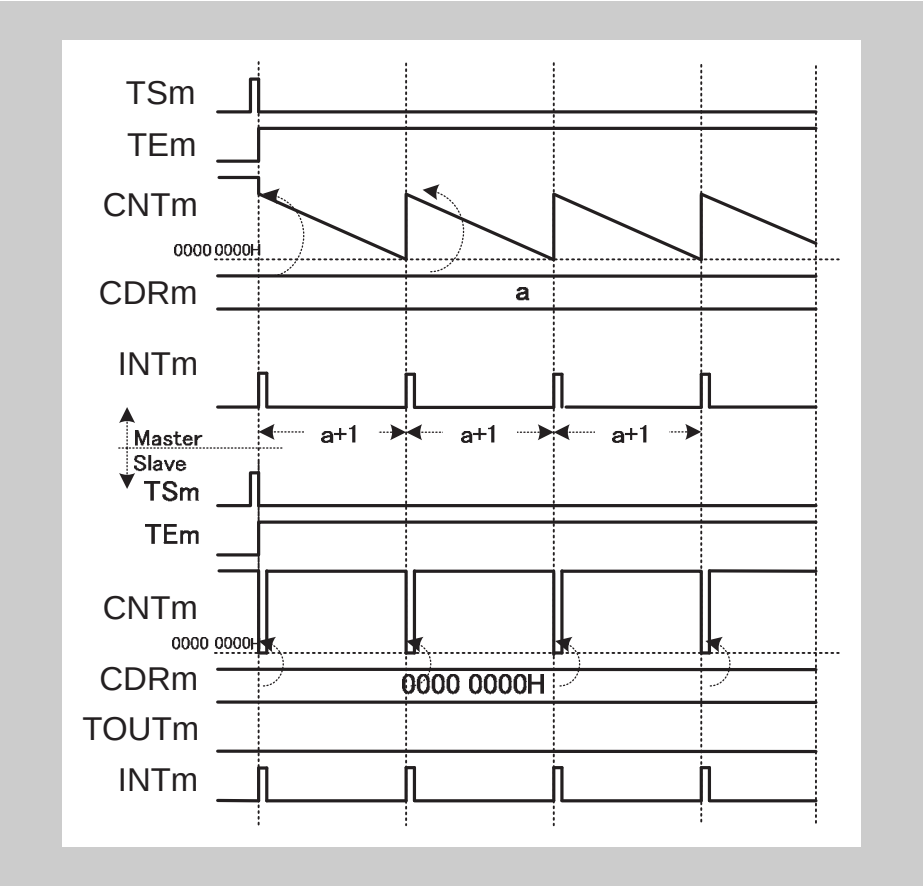



**Figure 15-54 TAUJnCDRm (slave) = 0000 $0000_H$, positive logic (TAUJnTOL.TOLm (slave) = 0)**

- Every time the master channel generates an interrupt (INTTAUJnIm), 0000 $0000_H$ is written to TAUJnCNTm (slave). Therefore, TAUJnCNTm (slave) cannot start to count and TAUJnTTOUTm remains at not active state.
- TAUJnCNTm (slave) generates an interrupt every time the value of TAUJnCDRm is reloaded. The slave and the master channel generate interrupts in the same cycle.

**(b) Duty cycle = 100 %**

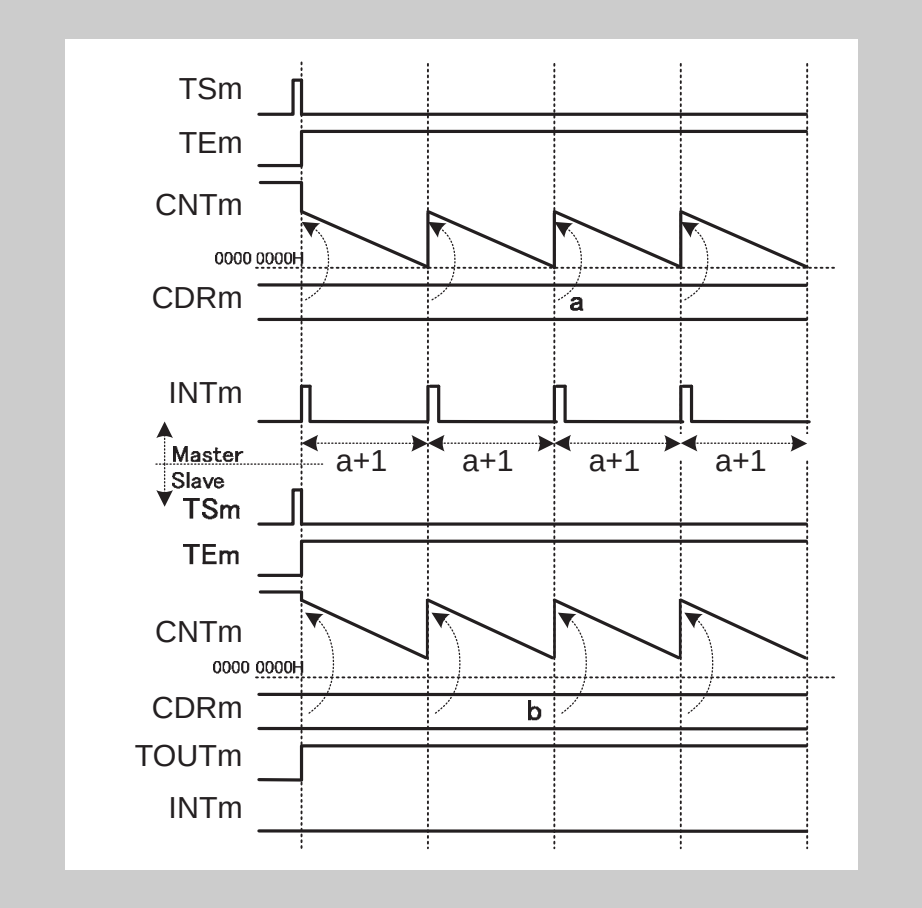



**Figure 15-55 TAUJnCDRm (slave) ≥ TAUJnCDRm (master) + 1, positive logic (TAUJnTOL.TOLm (slave) = 0)**

- If the value TAUJnCDRm (slave) is higher than the value TAUJnCDRm (master), the counter of the slave channel cannot reach $0000\ 0000_H$ and cannot generate interrupts. The TAUJnTTOUTm remains at active state.

### (c) Stop and restart operation

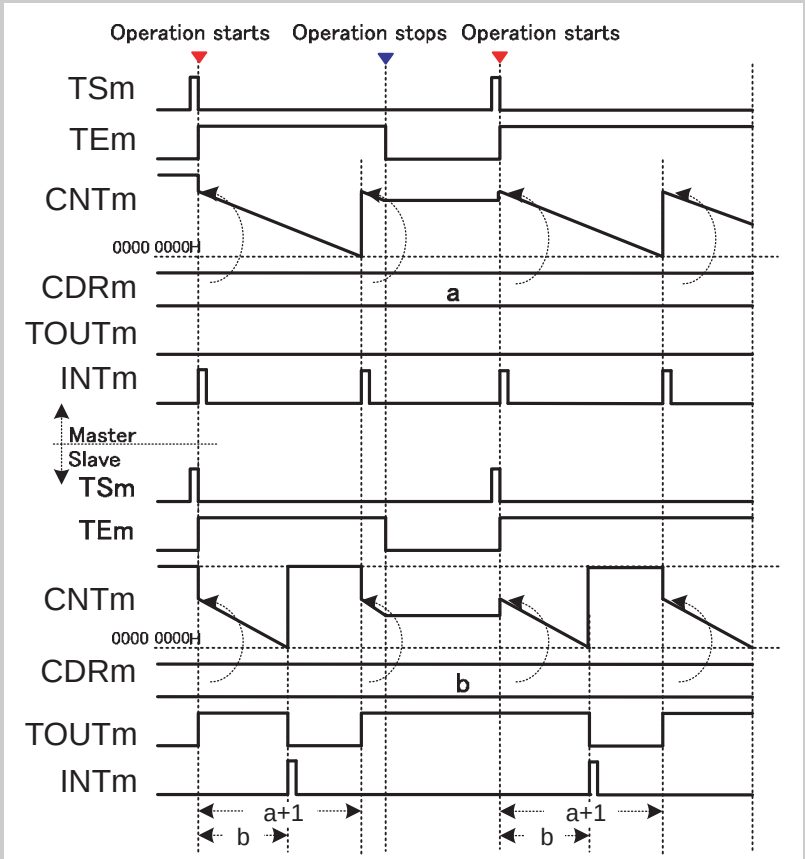



**Figure 15-56 Stop and restart operation, positive logic (TAUJnTOL.TOLm (slave) = 0)**

- The counter can be stopped by setting TAUJnTT.TTm of the master and slave channel(s) to 1, which in turn sets TAUJnTE.TEm to 0.
- TAUJnCNTm and TAUJnTTOUTm of all channels stop and the current values are retained. No interrupts are generated.
- The counter can be restarted by setting TAUJnTS.TSm of master and slave channel(s) to 1. TAUJnCNTm of master and slave channel reload the current values of TAUJnCDRm and start to count down from these values.

## 15.18 Registers

This section contains a description of all the registers of the 32-bit Timer Array Unit J.

### 15.18.1 TAUJn registers overview

The TAUJn is controlled and operated by the registers in the following table. Where there is one register per channel, this is indicated by an “m”, where m runs from 0 to 3.

**Table 15-54 TAUJn registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| TAUJn prescaler registers | | |
| TAUJn prescaler clock select register | TAUJnTPS | <TAUJn_base> + $90_H$ |
| TAUJn prescaler baud rate setting register | TAUJnBRS | <TAUJn_base> + $94_H$ |
| TAUJn control registers | | |
| TAUJn channel data register m | TAUJnCDRm | <TAUJn_base> + m x $4_H$ |
| TAUJn channel counter register m | TAUJnCNTm | <TAUJn_base> + $10_H$ + m x $4_H$ |
| TAUJn channel mode OS register m | TAUJnCMORm | <TAUJn_base> + $80_H$ + m x $4_H$ |
| TAUJn channel mode user register m | TAUJnCMURm | <TAUJn_base> + $20_H$ + m x $4_H$ |
| TAUJn channel status register m | TAUJnCSRm | <TAUJn_base> + $30_H$ + m x $4_H$ |
| TAUJn channel status clear trigger register m | TAUJnCSCm | <TAUJn_base> + $40_H$ + m x $4_H$ |
| TAUJn channel start trigger register | TAUJnTS | <TAUJn_base> + $54_H$ |
| TAUJn channel enable status register | TAUJnTE | <TAUJn_base> + $50_H$ |
| TAUJn channel stop trigger register | TAUJnTT | <TAUJn_base> + $58_H$ |
| TAUJn output registers | | |
| TAUJn channel output enable register | TAUJnTOE | <TAUJn_base> + $60_H$ |
| TAUJn channel output register | TAUJnTO | <TAUJn_base> + $5C_H$ |
| TAUJn channel output mode register | TAUJnTOM | <TAUJn_base> + $98_H$ |
| TAUJn channel output configuration register | TAUJnTOC | <TAUJn_base> + $9C_H$ |
| TAUJn channel output active level register | TAUJnTOL | <TAUJn_base> + $64_H$ |
| TAUJn reload data registers | | |
| TAUJn channel reload data enable register | TAUJnRDE | <TAUJn_base> + $A0_H$ |
| TAUJn channel reload data mode register | TAUJnRDM | <TAUJn_base> + $A4_H$ |
| TAUJn channel reload data trigger register | TAUJnRDT | <TAUJn_base> + $68_H$ |
| TAUJn channel reload status register | TAUJnRSF | <TAUJn_base> + $6C_H$ |
| **TAUJn emulation register** | | |
| TAUJn emulation register | TAUJnEMU | <TAUJn_base> + $A8_H$ |

**<TAUJn_base>** The <TAUJn_base> addresses of the registers are defined in the first section of this chapter under the keyword “Register addresses”.

## 15.18.2 TAUJn prescaler registers details

### (1) TAUJnTPS - TAUJn prescaler clock select register

This register specifies the PCLK prescalers for clocks CK0, CK1, CK2, and CK3_PRE for all channels. CK3 is generated by dividing CK3_PRE by the factor specified in TAUJnBRS.

**Access** This register can be read/written in 16-bit units.

**Address** <TAUJn_base> + $90_H$

**Initial Value** $FFFF_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PRS3[3:0] | | | | PRS2[3:0] | | | | PRS1[3:0] | | | | PRS0[3:0] | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-55 TAUJnTPS register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 12 | PRS3[3:0] | Specifies the CK3_PRE clock.<br>Clock CK3_PRE is the input clock of the BRG unit. The BRG unit supplies the CK3 operation clock for all channels.<br>**PRS3[3:0] / CK3_PRE clock**<br>$0000_B$ / PCLK/$2^0$<br>$0001_B$ / PCLK/$2^1$<br>$0010_B$ / PCLK/$2^2$<br>$0011_B$ / PCLK/$2^3$<br>... / ...<br>$1110_B$ / PCLK/$2^{14}$<br>$1111_B$ / PCLK/$2^{15}$<br>These bits can only be rewritten when all counters using CK3 are stopped (TAUJnTE.TEm = 0). |
| 11 to 8 | PRS2[3:0] | Specifies the CK2 clock.<br>**PRS2[3:0] / CK2 clock**<br>$0000_B$ / PCLK/$2^0$<br>$0001_B$ / PCLK/$2^1$<br>$0010_B$ / PCLK/$2^2$<br>$0011_B$ / PCLK/$2^3$<br>... / ...<br>$1110_B$ / PCLK/$2^{14}$<br>$1111_B$ / PCLK/$2^{15}$<br>These bits can only be rewritten when all counters using CK2 are stopped (TAUJnTE.TEm = 0). |

**Table 15-55 TAUJnTPS register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>7 to 4</td><td>PRS1[3:0]</td><td>Specifies the CK1 clock.
<table>
<tr><th>PRS1[3:0]</th><th>CK1 clock</th></tr>
<tr><td>$0000_B$</td><td>PCLK/$2^0$</td></tr>
<tr><td>$0001_B$</td><td>PCLK/$2^1$</td></tr>
<tr><td>$0010_B$</td><td>PCLK/$2^2$</td></tr>
<tr><td>$0011_B$</td><td>PCLK/$2^3$</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>$1110_B$</td><td>PCLK/$2^{14}$</td></tr>
<tr><td>$1111_B$</td><td>PCLK/$2^{15}$</td></tr>
</table>
These bits can only be rewritten when all counters using CK1 are stopped (TAUJnTE.TEm = 0).</td></tr>
<tr><td>3 to 0</td><td>PRS0[3:0]</td><td>Specifies the CK0 clock.
<table>
<tr><th>PRS0[3:0]</th><th>CK0 clock</th></tr>
<tr><td>$0000_B$</td><td>PCLK/$2^0$</td></tr>
<tr><td>$0001_B$</td><td>PCLK/$2^1$</td></tr>
<tr><td>$0010_B$</td><td>PCLK/$2^2$</td></tr>
<tr><td>$0011_B$</td><td>PCLK/$2^3$</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>$1110_B$</td><td>PCLK/$2^{14}$</td></tr>
<tr><td>$1111_B$</td><td>PCLK/$2^{15}$</td></tr>
</table>
These bits can only be rewritten when all counters using CK0 are stopped (TAUJnTE.TEm = 0).</td></tr>
</table>

**Note** The TAUJn clock input PCLK is specified in the first section of this chapter under the keyword “Clock supply”.

### (2) TAUJnBRS - TAUJn prescaler baud rate setting register

This register specifies the division factor of prescaler clock CK3.

CK3 is generated by dividing CK3_PRE by the factor specified in this register plus one. The PCLK prescaler for CK3_PRE is specified in TAUJnTPS.PRS3[3:0].

**Access** This register can be read/written in 8-bit units.

**Address** <TAUJn_base> + $94_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BRS[07:00] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-56 TAUJnBRS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | BRS[07:00] | Specifies the CK3_PRE clock division factor for generating CK3: |

| BRS[07:00] | CK3 clock |
|---|---|
| $0000\ 0000_B$ | CK3_PRE / 1 |
| $0000\ 0001_B$ | CK3_PRE / 2 |
| $0000\ 0010_B$ | CK3_PRE / 3 |
| $0000\ 0011_B$ | CK3_PRE / 4 |
| ... | ... |
| $1111\ 1110_B$ | CK3_PRE / 255 |
| $1111\ 1111_B$ | CK3_PRE / 256 |

### 15.18.3 TAUJn control registers details

#### (1) TAUJnCDRm - TAUJn channel data register

This register functions either as a compare register or as a capture register, depending on the operation mode specified in TAUJnCMORm.MD[4:0].

**Access** This register can be read/written in 32-bit units.

- In capture mode, only reading is possible. Write operation is ignored.
- In compare mode, reading and writing is possible.

**Address** <TAUJn_base> + $0_H$ + m x $4_H$

**Initial Value** 0000 $0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CDR[31:16] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CDR[15:0] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-57 TAUJnCDRm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 0 | CDR[31:0] | Data register for the capture/compare value. |

### (2) TAUJnCNTm - TAUJn channel counter register

This register is the channel m counter register.

**Access** This register can be read in 32-bit units.

**Address** <TAUJn_base> + $10_H$ + m × $4_H$

**Initial Value** 0000 $0000_H$ or FFFF $FFFF_H$ The initial value depends on the operation mode, see *Table 15-59 "TAUJnCNTm read values after the counter is re-enabled" on page 794*

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNT[31:16] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNT[15:0] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 15-58 TAUJnCNTm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 to 0 | CNT[31:0] | 32-bit counter value. |

The read value depends on the counter, the operation mode change, and the values of the TAUJnTS.TSm and TAUJnTT.TTm bits.

The *initial* counter read value depends on the operation mode and how the counter was stopped:

- by a reset
- by a counter stop trigger (TAUJnTT.TTm = 1)

The following table lists the initial counter read values after the counter has stopped (TAUJnTE.TEm= 0 ) and re-enabled (TAUJnTS.TSm = 1).

The table also contains the counter read value one count after the counter is enabled (TAUJnTS.TSm = 1) for modes where the counter waits for a start trigger.

**Table 15-59 TAUJnCNTm read values after the counter is re-enabled**

| Mode name | Count method (up/down) | TAUJnCNTm value | | |
|---|---|---|---|---|
| | | After reset | After stop trigger | After one count |
| Interval Timer mode | Count down | FFFF $FFFF_H$ | Stop value | - |
| Capture mode | Count up | 0000 $0000_H$ | Stop value | - |
| One Count mode | Count down | FFFF $FFFF_H$ | Stop value | FFFF $FFFF_H$ |
| Capture & One Count mode | Count up | 0000 $0000_H$ | Stop value | Captured value + 1 (TAUJnCDRm) |
| Count Capture Mode | Count up | 0000 $0000_H$ | Stop value | - |
| Gate Count Mode | Count down | FFFF $FFFF_H$ | Stop value | Stop value |
| Capture & Gate Count Mode | Count up | 0000 $0000_H$ | Stop value | Stop value |

**Note** If the operation mode is changed while the counter is stopped, the initial counter value after counter restart is undefined. The operation mode is changed by register TAUJnCMORm.MD[4:0].

### (3) TAUJnCMORm - TAUJn channel mode OS register

This register controls channel m operation. It specifies, for example, the operation clock, count clock, and master/slave function.

**Access** This register can be read/written in 16-bit units. It can only be written when the counter is stopped (TAUJnTE.TEm = 0).

**Address** <TAUJn_base> + $80_H$ + m × $4_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CKS[1:0] | | CCS[1:0] | | MAS | STS[2:0] | | | COS[1:0] | | - | MD[4:0] | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W |

**Table 15-60 TAUJnCMORm register contents (1/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 15,14 | CKS[1:0] | Selects the operation clock.<br>The operation clock is used for the TAUJnTTINm input edge detection circuit. It can also be used as the count clock depending on bits TAUJnCMORm.CCS[1:0].<br>**CKS1 / CKS0 / Selected operation clock**<br>0 / 0 / CK0<br>0 / 1 / CK1<br>1 / 0 / CK2<br>1 / 1 / CK3 |
| 13,12 | CCS[1:0] | Selects the count clock for TAUJnCNTm counter:<br>**CCS1 / CCS0 / Selected count clock**<br>0 / 0 / Operation clock as specified by TAUJnCMORm.CKS[1:0].<br>0 / 1 / Valid edge of TAUJnTTINm input signal<br>1 / 0 / Setting prohibited<br>1 / 1 / Setting prohibited |
| 11 | MAS | Specifies the channel as master or slave channel during synchronous channel operation:<br>0: Slave<br>1: Master<br>This bit is only valid for even channels (CHm_even). For odd channels (CHm_odd), it is fixed to 0. |

**Table 15-60 TAUJnCMORm register contents (2/3)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>10 to 8</td><td>STS[2:0]</td><td>Selects the external start trigger:
<table>
<tr><th>STS2</th><th>STS1</th><th>STS0</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Software trigger.</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Valid edge of the TAUJnTTINm input signal. TAUJnCMURm.TIS[1:0] specifies the valid edge.</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Valid edge of the TAUJnTTINm input signal is the start trigger and the reverse edge is the stop (capture) trigger.</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Setting prohibited</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>INTTAUJnI of the master channel.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td rowspan="3">Setting prohibited</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>7, 6</td><td>COS[1:0]</td><td>Specifies when the capture register TAUJnCDRm and the overflow flag TAUJnCSRm.OVF of channel m are updated.<br>These bits are only valid if channel m is in capture mode.
<table>
<tr><th>COS1</th><th>COS0</th><th>Capture register</th><th>TAUJnCSRm.OVF</th></tr>
<tr><td>0</td><td>0</td><td rowspan="2">Updated upon detection of a TAUJnTTINm input valid edge.</td><td>Updated (cleared or set) upon detection of a TAUJnTTINm input valid edge:<br>• If a counter overflow has occurred since the last valid edge detection, TAUJnCSRm.OVF is set.<br>• If no counter overflow has occurred since the last valid edge detection, TAUJnCSRm.OVF is cleared.</td></tr>
<tr><td>0</td><td>1</td><td>Set upon counter overflow and cleared by a CPU instruction.</td></tr>
<tr><td>1</td><td>0</td><td rowspan="2">Updated upon detection of a TAUJnTTINm input valid edge and upon counter overflow:<br>• TAUJnTTINm input valid edge: Counter value is written to TAUJnCDRm.<br>• Overflow: FFFF FFFF$_H$ is written to TAUJnCDRm. The next TAUJnTTINm input valid edge detection is ignored.</td><td>Not set.</td></tr>
<tr><td>1</td><td>1</td><td>Set upon counter overflow and cleared by a CPU instruction.</td></tr>
</table></td></tr>
</table>

**Table 15-60 TAUJnCMORm register contents (3/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 4 to 0 | MD[4:0] | Specifies the operation mode. |

| MD4 | MD3 | MD2 | MD1 | MD0 | Description |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1/0 | Interval Timer mode |
| 0 | 0 | 0 | 1 | 1/0 | Setting prohibited |
| 0 | 0 | 1 | 0 | 1/0 | Capture mode |
| 0 | 0 | 1 | 1 | 1/0 | Setting prohibited |
| 0 | 1 | 0 | 0 | 1/0 | One Count mode |
| 0 | 1 | 0 | 1 | 1/0 | Setting prohibited |
| 0 | 1 | 1 | 0 | 0 | Capture & One Count mode |
| 0 | 1 | 1 | 1 | 1/0 | Setting prohibited |
| 1 | 0 | 0 | 0 | | |
| 1 | 0 | 0 | 1 | | |
| 1 | 0 | 1 | 0 | | |
| 1 | 0 | 1 | 1 | 1/0 | Count Capture mode |
| 1 | 1 | 0 | 0 | 0 | Gate Count mode |
| 1 | 1 | 0 | 1 | 0 | Capture & Gate Count mode |

| Mode | Role of the MD0 bit |
|---|---|
| Interval Timer mode<br>Capture mode<br>Count Capture mode | Specifies whether an INTTAUJnIm is generated when the counter is triggered:<br>0: No INTTAUJnIm generated<br>1: INTTAUJnIm generated |
| One Count mode<br>Gate Count mode | Enables/disables start trigger detection during counting:<br>0: Disabled<br>1: Enabled<br>INTTAUJnIm and TAUJnTTOUTm are not output when the counter is triggered. |
| Capture & One Count mode<br>Capture & Gate Count mode | This bit must be set to 0:<br>0: No INTTAUJnIm generated when the counter is triggered. Start trigger is disabled during counting.<br>1: - |

### (4) TAUJnCMURm - TAUJn channel mode user register

This register specifies the type of valid edge detection used for the TAUJnTTINm input.

**Access** This register can be read/written in 8-bit units. It can only be written when the counter is enabled (TAUJnTE.TEm = 1).

**Address** <TAUJn_base> + $20_H$ + m × $4_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | TIS[1:0] | |
| R | R | R | R | R | R | R/W | R/W |

**Table 15-61 TAUJnCMURm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1, 0 | TIS[1:0] | Specifies the valid edge of the TAUJnTTINm input:<br>• To detect rising and falling edges when TAUJnCMORm.STS[2:0] is not set to $010_B$, set TAUJnCMURm.TIS[1:0] = $10_B$.<br>• Edge detection for TAUJnTTINm input signals is performed based on the operation clock selected by TAUJnCMORm.CKS[1:0]. |

| TIS1 | TIS0 | Description |
|---|---|---|
| 0 | 0 | Falling edge |
| 0 | 1 | Rising edge |
| 1 | 0 | Rising and falling edges (low-width measurement selection).<br>Start trigger: falling edge<br>Stop trigger (capture): rising edge |
| 1 | 1 | Rising and falling edges (high-width measurement selection).<br>Start trigger: rising edge<br>Stop trigger (capture): falling edge<br>This setting is only valid if the start trigger selection is set to TAUJnCMORm.STS[2:0] = $010_B$ |

### (5) TAUJnCSRm - TAUJn channel status register

This register indicates the overflow status of channel m's counter.

**Access** This register can be read in 8-bit units.

**Address** <TAUJn_base> + $30_H$ + m × $4_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | OVF |
| R | R | R | R | R | R | R | R |

**Table 15-62 TAUJnCSRm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | OVF | Indicates the counter overflow status:<br>0: No overflow occurred<br>1: Overflow occurred<br>This bit is only used in the following modes:<br>• Capture mode<br>• Capture & One Count mode<br>• Count Capture mode<br>• Capture & Gate Count mode<br><br>The function of this bit depends on the setting of control bits TAUJnCMORm.COS[1:0].<br>OVF is not be set when TAUJnCMORm.COS[1:0] = $10_B$. |

### (6) TAUJnCSCm - TAUJn channel status clear register

This register is a trigger register for clearing the overflow flag TAUJnCSRm.OVF of a channel m.

**Access** This register can be written in 8-bit units. It is always read as $0000_H$.

**Address** <TAUJn_base> + $40_H$ + m × $4_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | 0 | CLOV |
| R | R | R | R | R | R | R | W |

**Table 15-63 TAUJnCSCm register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | CLOV | 0: No function<br>1: Clears the overflow flag TAUJnCSRm.OVF |

### (7) TAUJnTS - TAUJn channel start trigger register

This register enables the counter for each channel.

**Access** This register can be written in 8-bit units. It is always read as $0000_H$.

**Address** <TAUJn_base> + $54_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TS03 | TS02 | TS01 | TS00 |
| W | W | W | W | W | W | W | W |

**Table 15-64 TAUJnTS register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TSm | Enables the counter for channel m:<br>0: No function<br>1: Enables the counter and sets TAUJnTE.TEm = 1. When the counter is enabled, this bit immediately returns to 0.<br>TAUJnTE.TEm = 1 only *enables* counter. Whether the counter *starts* depends on the selected operation mode. |

### (8) TAUJnTE - TAUJn channel enable status register

This register indicates whether counter is enabled or disabled.

**Access** This register can be read in 8-bit units.

**Address** <TAUJn_base> + $50_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TE03 | TE02 | TE01 | TE00 |
| R | R | R | R | R | R | R | R |

**Table 15-65 TAUJnTE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TEm | Indicates whether counter for channel m is enabled or disabled:<br>0: Counter disabled<br>1: Counter enabled<br>Setting TAUJnTS.TSm to 1 or trigger input detection TAUJnTSSTm = 1 sets this bit to 1.<br>Setting TAUJnTT.TTm to 1 resets this bit to 0. |

### (9) TAUJnTT - TAUJn channel stop trigger register

This register stops the counter for each channel.

**Access** This register can be written in 8-bit units. It is always read as $0000_H$.

**Address** <TAUJn_base> + $58_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TT03 | TT02 | TT01 | TT00 |
| W | W | W | W | W | W | W | W |

**Table 15-66 TAUJnTT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TTm | Stops the counter of channel m:<br>0: No function<br>1: Stops the counter and sets TAUJnTE.TEm = 0. When the counter has stopped, this bit immediately returns to 0.<br>TAUJnCNTm stops counting and TAUJnCNTm, TAUJnTO.TOm, and TAUJnTTOUTm all retain the values they had before the counter was stopped. |

## 15.18.4 TAUJn output registers details

### (1) TAUJnTOE - TAUJn channel output enable register

This register enables and disables Direct Channel Output Mode.

**Access** This register can be read/written in 8-bit units.

**Address** <TAUJn_base> + $60_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TOE03 | TOE02 | TOE01 | TOE00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-67 TAUJnTOE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TOEm | Enables/disables Direct Channel Output Mode:<br>0: Enables Direct Channel Output Mode (TAUJnTTOUT m output is controlled by the application software)<br>1: Disables Direct Channel Output Mode (TAUJnTTOUT m output is controlled by the timer) |

### (2) TAUJnTO - TAUJn channel output register

This register specifies and reads the level of TAUJnTTOUTm.

**Access** This register can be read/written in 8-bit units.

**Address** <TAUJn_base> + $5C_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TO03 | TO02 | TO01 | TO00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-68 TAUJnTO register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TOm | Specifies/reads the level of TAUJnTTOUTm:<br>0: Low<br>1: High<br>Only TOm bits for which Independent Channel Output function is enabled (TAUJnTOEm = 0) can be written. |

### (3) TAUJnTOM - TAUJn channel output mode register

This register specifies the output mode of each channel.

**Access** This register can be read/written in 8-bit units. It can only be written when the counter is stopped (TAUJnTE.TEm = 0).

**Address** <TAUJn_base> + $98_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TOM 03 | TOM 02 | TOM 01 | TOM 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-69 TAUJnTOM register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TOMm | Specifies the timer controlled channel output mode, if Direct Channel Output Mode is disabled (TAUJnTOE.TOEm = 1):<br>0: Independent Channel Output Mode<br>1: Synchronous Channel Output Mode<br>The output mode depends on several channel output control bits, as can be seen in *Table 15-8 “Channel output modes” on page 712* |

### (4) TAUJnTOC - TAUJn channel output configuration register

This register specifies the output mode of each channel in combination with TAUJnTOMm.

**Access** This register can be read/written in 8-bit units. It can only be written when the counter is stopped (TAUJnTE.TEm = 0).

**Address** <TAUJn_base> + $9C_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TOC 03 | TOC 02 | TOC 01 | TOC 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-70 TAUJnTOC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TOCm | Specifies the output mode:<br>0: Operation mode 1 (= Toggle mode)<br>1: No function<br>This bit must be set to 0 for all output modes except Direct Channel Output Mode. |

### (5) TAUJnTOL - TAUJn channel output level register

This register specifies the output logic of the channel output bit (TAUJnTO.TOm).

**Access** This register can be read/written in 8-bit units.

**Address** <TAUJn_base> + $64_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | TOL 03 | TOL 02 | TOL 01 | TOL 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-71 TAUJnTOL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | TOLm | Specifies the output logic of the channel m output bit (TAUJnTO.TOm):<br>0: Positive logic (active high)<br>1: Inverted logic (active low)<br>These bits apply in all channel output modes except Direct Channel Output Mode. |

## 15.18.5 TAUJn simultaneous rewrite register details

### (1) TAUJnRDE - TAUJn channel reload data enable register

This register enables and disables simultaneous rewrite of the data register TAUJnCDRm. It also enables simultaneous rewrite of the data register TAUJnTOLm for the PWM output function and the triangle PWM output function.

**Access** This register can be read/written in 8-bit units. It can only be written when TAUJnTE.TEm = 0.

**Address** <TAUJn_base> + $A0_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | RDE 03 | RDE 02 | RDE 01 | RDE 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-72 TAUJnRDE register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | RDEm | Enables/disables simultaneous rewrite of the data register of channel m:<br>0: Disables simultaneous rewrite<br>1: Enabled simultaneous rewrite |

### (2) TAUJnRDM - TAUJn channel reload data mode register

This register selects when the signal that controls simultaneous rewrite is loaded.

**Access** This register can be read/written in 8-bit units. It can only be written when TAUJnTE.TEm = 0.

**Address** <TAUJn_base> + $A4_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | RDM 03 | RDM 02 | RDM 01 | RDM 00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-73 TAUJnRDM register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | RDMm | Specifies when the signal that triggers simultaneous rewrite is generated:<br>0: When the master channel counter starts counting<br>1: No function<br>These bits only apply when TAUJnRDE.RDEm = 1. |

### (3) TAUJnRDT - TAUJn channel reload data trigger register

This register triggers the simultaneous rewrite pending state.

**Access** This register can be written in 8-bit units. It is always read as $0000_H$.

**Address** <TAUJn_base> + $68_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | RDT 03 | RDT 02 | RDT 01 | RDT 00 |
| W | W | W | W | W | W | W | W |

**Table 15-74 TAUJnRDT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | RDTm | Triggers the simultaneous rewrite pending state:<br>0: No function<br>1: Simultaneous rewrite pending state is triggered. The simultaneous rewrite pending flag (TAUJnRSFm) is set to 1. The system waits for the simultaneous rewrite trigger. TAUJnRDT.RDTm immediately returns to 0. |

### (4) TAUJnRSF - TAUJn channel reload status register

This flag register indicates that simultaneous rewrite is possible.

**Access** This register can be read in 8-bit units.

**Address** <TAUJn_base> + $6C_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | RSF 03 | RSF 02 | RSF 01 | RSF 00 |
| R | R | R | R | R | R | R | R |

**Table 15-75 TAUJnRSF register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 to 0 | RSFm | Indicates the simultaneous rewrite status:<br>0: Simultaneous rewrite disabled<br>1: Simultaneous rewrite enabled |

### 15.18.6 TAUJn emulation register

#### (1) TAUJnEMU - TAUJ emulation register

This register controls whether the TAUJn can be stopped during emulation, for instance upon a breakpoint hit.

**Access** This register can be read/written in 8-bit units.

**Address** <TAUJn_base> + $A8_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TAUJn SVSDIS | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 15-76 TAUJnEMU register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | TAUJn SVSDIS | Emulation control<br>0: TAUJn can be stopped during emulation<br>1: TAUJn continuous operating during emulation |

# Chapter 16 Asynchronous Serial Interface E (URTE)

This chapter contains a generic description of the Asynchronous Serial Interface E.

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 16.1 V850E2/Fx4-G URTEn Features

**Instances**

This microcontroller has the following number of instances of the Asynchroneous Serial Interface E URTEn.

**Table 16-1 Instances of URTE**

| Asynchroneous Serial Interface E | V850E2/FF4-G | V850E2/FG4-G |
|---|---|---|
| Instance | 2 | 3 |
| Name | URTE10, URTE11 | URTE2, URTE10, URTE11 |

**Instances index n**

Throughout this chapter, the instance of an Asynchroneous Serial Interface E is identified by the index "n" (n = 2, 10, 11), for example, URTEnCTL0 for the URTEn control register 0.

**Register addresses**

All URTEn register addresses are given as address offsets to the individual base address <URTEn_base>.
The <URTEn_base> address of each URTEn are listed in the following table:

**Table 16-2 Register base addresses <URTEn_base>**

| URTEn instance | <URTEn_base> address |
|---|---|
| URTE2 | FF5E 0000$_H$ |
| URTE10 | FF66 0000$_H$ |
| URTE11 | FF67 0000$_H$ |

**Clock supply** All Asynchroneous Serial Interface E provide one clock input.

**Table 16-3 URTEn clock supply**

| URTEn instance | URTEn clock | Connected to |
|---|---|---|
| URTE2 | PCLK | Clock Controller CKSCLK_034 |
| URTE10 | PCLK | Clock Controller CKSCLK_011 |
| URTE11 | PCLK | Clock Controller CKSCLK_011 |

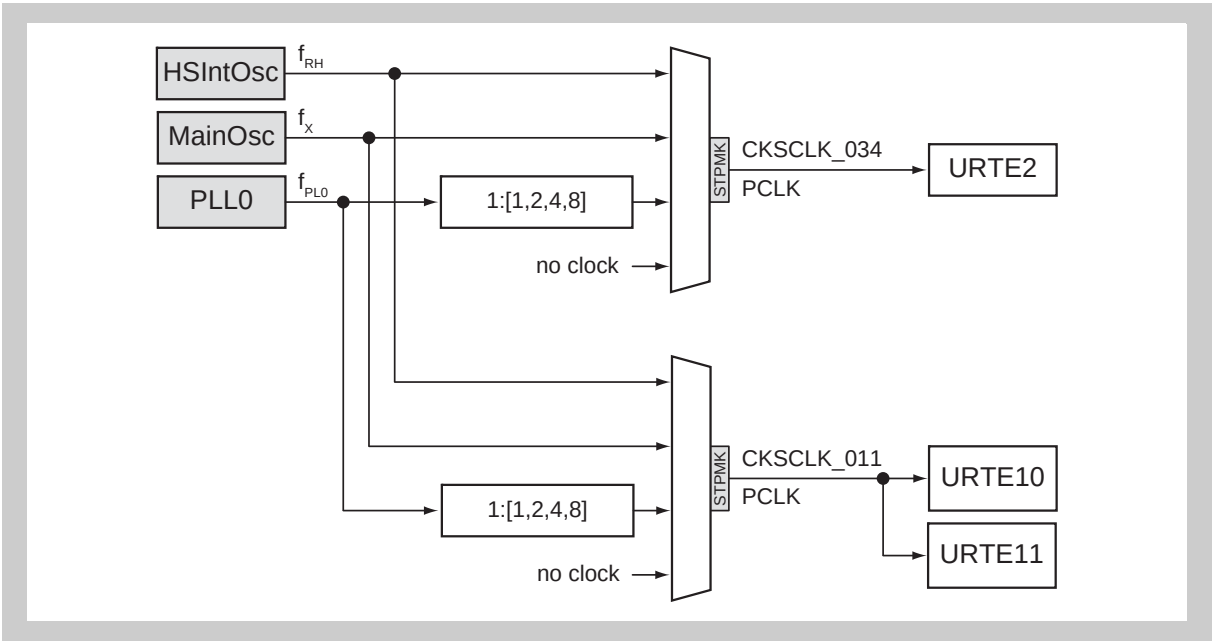



**Figure 16-1 URTE clock supply**

**Interrupts** The Asynchroneous Serial Interface E can generate following interrupts requests, which are input to their respective LIN Master Controllers LMAn:

**Table 16-4 URTEn interrupts requests**

| URTEn signals | Function | Connected to |
|---|---|---|
| INTUAEnTIT | Transmission interrupt | LMAn INTUAEnIT |
| INTUAEnTIR | Reception interrupt | LMAn INTUAEnIR |
| INTUAEnTIS | Status interrupt | LMAn INTUAEnIS |

**URTE H/W reset**

The Asynchroneous Serial Interfaces E and their registers are initialized by the following reset signal:

**Table 16-5 URTEn reset signal**

| URTEn | Reset signal |
|---|---|
| URTEn | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

**I/O signals**

The I/O signals of the Asynchroneous Serial Interface E are listed in the table below.

**Table 16-6 URTEn I/O signals**

| URTEn signals | Function | Connected to |
|---|---|---|
| URTEnTTXD | Transmit data output | Port URTEnTX |
| URTEnTRXD | Receive data input | Port URTEnRX[a] |

a) These input signals are passed through a noise filter, refer to the section *"Port Filters"* in the chapter *"Port Functions"*.

**Baudrate measurement**

Following URTEn receive data signals can be internally connected to a capture input of the Timer Array Units B.

**Table 16-7 URTEn timer connections**

| URTEn signals | Function | Connected to |
|---|---|---|
| **URTE2:** | | |
| Port URTE2RX | Receive data input | TAUB0 TAUB0TTIN2 |
| **URTE10:** | | |
| Port URTE10RX | Receive data input | TAUB0 TAUB0TTIN0 |
| **URTE11:** | | |
| Port URTE11RX | Receive data input | TAUB0 TAUB0TTIN1 |

Refer to section *"TAUB Input Selections"* in the *"Timer Array Unit B (TAUB)"* for details.

## 16.2 Functional Overview

- Full-duplex communication:
  - Internal URTEn receive data register n (URTEnRX)
  - Internal URTEn transmit data register n (URTEnTX)
- 2-pin configuration:
  - URTEnTTXD: Transmit data output pin
  - URTEnTRXD: Receive data input pin
- Reception error and status output function
  - Parity error
  - Framing error
  - Overrun error
  - Data consistency error
  - BF receive error
- Interrupt sources: 3
  - Transmission interrupt INTUAEnTIT
  - Reception interrupt INTUAEnTIR
  - Status interrupt INTUAEnTIS
- Character length: 7, 8 bits
- Parity function: odd, even, 0, none
- Transmission stop bit: 1, 2 bits
- MSB-/LSB-first transfer selectable
- Transmit/receive data inverted input/output possible
- 13 to 20 bits selectable for the BF (Break Field) in the LIN (Local Interconnect Network) communication format
  - Recognition of 11 bits or more possible for BF reception in LIN communication format
  - BF reception flag provided
- BF reception can be detected during data communication
- Bus monitor function to keep data consistency of the transmit data

## 16.3 Configuration

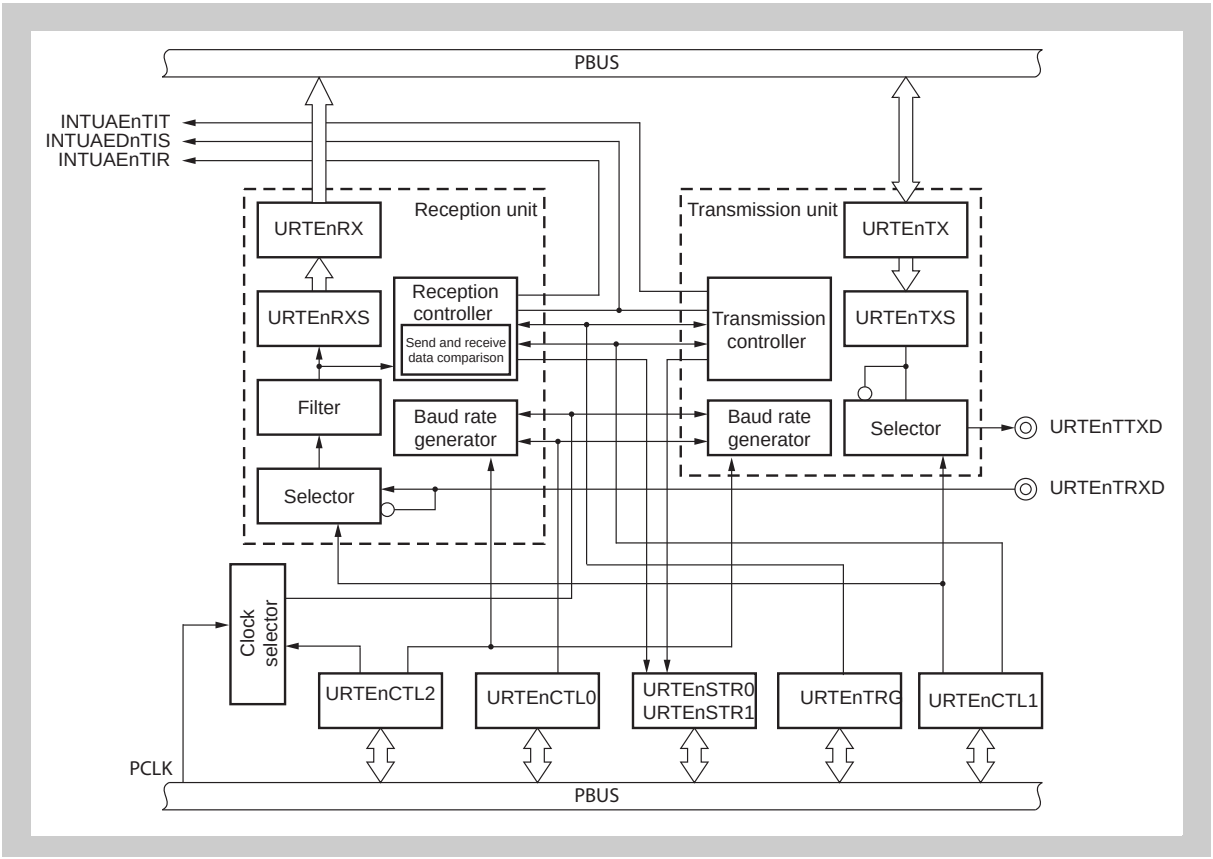



**Figure 16-2 Block diagram of Asynchronous Serial Interface URTEn**

## 16.4 URTE Registers

The URTE is controlled and operated by means of the following registers:

**Table 16-8 URTEn registers**

| Register function | Name | Address |
|---|---|---|
| Control register 0 | URTEnCTL0 | <URTEn_base> + $00_H$ |
| Control register 1 | URTEnCTL1 | <URTEn_base> + $20_H$ |
| Control register 2 | URTEnCTL2 | <URTEn_base> + $24_H$ |
| Trigger register | URTEnTRG | <URTEn_base> + $04_H$ |
| Status register 0 | URTEnSTR0 | <URTEn_base> + $08_H$ |
| Status register 1 | URTEnSTR1 | <URTEn_base> + $0C_H$ |
| Status clear register | URTEnSTC | <URTEn_base> + $10_H$ |
| Receive data register | URTEnRX | <URTEn_base> + $14_H$ |
| Transmit data register | URTEnTX | <URTEn_base> + $18_H$ |
| Emulation register | URTEnEMU | <URTEn_base> + $34_H$ |

**<URTEn_base>** The base addresses <URTEn_base> of the URTEn are defined in the first section of this chapter under the key word “Register addresses”.

**Register access** All registers are accessible in 32-bit width.
Writing to non-existing register bits is ignored, reading of these bits return always 0.

### (1) URTEnCTL0 - URTEn control register 0

This register controls the URTEn basic serial transfer operation.

**Access** This register can be read/written in 8-bit units.

**Address** <URTEn_base> + $00_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| URTEn PW | URTEn TXE | URTEn RXE | 0 | 0 | 0 | 0 | URTEn SLDC |
| R/W | R/W | R/W | R | R | R | R | R/W |

**Table 16-9 URTEnCTL0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | URTEn PW | URTEn enable<br>0: Stop URTEn operation<br>1: Enable URTEn operation<br>Changing this bit initializes all transmission and reception units. |
| 6 | URTEn TXE | Transmission operation enable<br>0: Disable transmission operation<br>1: Enable transmission operation<br>• To start transmission, set URTEnPW to 1 and then set URTEnTXE to 1. To stop transmission clear URTEnTXE to 0 and then URTEnPW to 0.<br>• To initialize the transmission unit, clear URTEnTXE to 0, wait for two cycles of the base clock, and then set URTEnTXE to 1 again. |
| 5 | URTEn RXE | Reception operation enable<br>0: Disable reception operation<br>1: Enable reception operation<br>• To enable reception, set URTEnPW to 1 and then set URTEnRXE to 1. To disable reception clear URTEnRXE to 0 and then URTEnPW to 0.<br>• To initialize the reception unit, clear URTEnRXE to 0, wait for two periods of the base clock, and then set URTEnRXE to 1 again. The reception is enabled after URTEnRXE is set to 1 and two cycles of base clock have passed. The rising edge detection of the RXDD pin is enabled after URTEnRXE is set to 1 and four cycles of the base clock have passed. |
| 0 | URTEn SLDC | Data consistency check enable<br>0: Disable data consistency check<br>1: Enable data consistency check<br>This bit selects the handling of data consistency error checks when transmitting data.<br>When this bit is set to 1, the transmit data and receive data are compared, and if a mismatch is detected, URTEnSTR1.URTEnDCE is set to 1 and a status interrupt request INTUAEnTIS is issued.<br>This bit is accessed only when starting a transmission. Consequently, if this bit value is changed later on during transmission processing, the transmission processing continues to use the value set at the start of processing. |

**Cautions** **1.** In case URTEn is in the status of

- enabled reception and transmission (URTEnCTL0 bits URTEnPW = 1 & URTEnRXE & URTEnTXE = 1) and
- data consistency check is enabled (URTEnCTL0.URTEnSLDC = 1) and
- transmission is ongoing or is completed

transmission shall be disabled, while keeping reception enabled, proceed as follows:

- check that no transmission is pending (URTEnSTR0 bits URTEnSSBT = URTEnSST = 0) and
- check that no reception is ongoing (URTEnSTR0 bits URTEnSSBR = URTEnSSR = 0) then
- disable transmission by URTEnCTL0.URTEnTXE = 0.

The reason for this procedure is that the data consistency error flag URTEnSTR1.URTEnDCE is cleared to 0 by URTEnCTL0.URTEnTXE. Thus a potential data consistency error would not be reported, if transmission is disabled during a data transfer or after its completion.

2. In case URTEn is in the status of
   - enabled reception and transmission (URTEnCTL0 bits URTEnPW = 1 & URTEnRXE & URTEnTXE = 1) and
   - data consistency check is enabled (URTEnCTL0.URTEnSLDC = 1) and
   - transmission is ongoing or is completed

   reception shall be disabled, while keeping transmission enabled, proceed as follows:
   - check that no transmission is pending (URTEnSTR0 bits URTEnSSBT = URTEnSST = 0) and
   - check that no reception is ongoing (URTEnSTR0 bits URTEnSSBR = URTEnSSR = 0) then
   - disable reception by URTEnCTL0.URTEnRXE = 0.

   The reason for this procedure is that the data consistency error flag URTEnSTR1.URTEnDCE is invalid URTEnCTL0.URTEnTXE is cleared. Thus a potential data consistency error of already transmitted data would not be reported.
3. Don’t start any data transmission if
   - data consistency check is enabled (URTEnCTL0.URTEnSLDC = 1) and
   - BF reception is enabled (URTEnSTR0.URTEnSSBR = 1) and
   - detection of a BF during reception is not active (URTEnCTL1.URTEnSLBM = 0)

   A data consistency error will occur under above conditions when the BF reception is completed. The status interrupt INTUAEnTIS will be asserted and the BF reception completion will not be indicated (URTEnSTR1.URTEnBSF remains 0). Consequently the BF reception completion will not be recognized.

### (2) URTEnCTL1 - URTEn control register 1

This register defines the data frame properties of the URTEn serial data transfers.

**Access** This register can be read/written in 16-bit units.

**Address** <URTEn_base> + $20_H$

**Initial Value** $5002_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| URTEn SLBM | URTEn BLG[2:0] | | | 0 | 0 | 0 | URTEn CLG |
| R/W | R/W | R/W | R/W | R | R | R | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| URTEn SLP[1:0] | | URTEn TDL | URTEn RDL | 0 | URTEn SLG | URTEn SLD | URTEn SLIT |
| R/W | R/W | R/W | R/W | R | R/W | R/W | R/W |

**Table 16-10 URTEnCTL1 register contents (1/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | URTEn SLBM | BF receive mode selection<br>0: BF reception during data reception disabled.<br>1: BF reception during data reception enabled.<br>• When this bit is set to 1, data reception processing uses a character bit length of 8 bits and no parity bit, regardless of the values set to the URTEnCLG and URTEnSLP[1:0].<br>• Changing this bit is only allowed, if reception is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnRXE = 0). |
| 14 to 12 | URTEn BLG[2:0] | BF bit length during transmission<br>URTEnBLG2 / URTEnBLG1 / URTEnBLG0 / BF length<br>1 / 0 / 1 / 13 bits<br>1 / 1 / 0 / 14 bits<br>1 / 1 / 1 / 15 bits<br>0 / 0 / 0 / 16 bits<br>0 / 0 / 1 / 17 bits<br>0 / 1 / 0 / 18 bits<br>0 / 1 / 1 / 19 bits<br>1 / 0 / 0 / 20 bits<br>Changing these bits is only allowed, if transmission is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnTXE = 0). |
| 8 | URTEn CLG | Receive/transmit data bit length<br>0: 7 bits<br>1: 8 bits<br>• When the transmission/reception is performed in the LIN format, set URTEnCLG to 1.<br>• Changing this bit is only allowed, if reception and transmission is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnRXE = URTEnCTL0.URTEnTXE = 0). |

**Table 16-10 URTEnCTL1 register contents (2/3)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>7 to 6</td><td>URTEn<br>SLP[1:0]</td><td>Parity bit selection
<table>
<tr><th rowspan="2">URTEn<br>SLP1</th><th rowspan="2">URTEn<br>SLP0</th><th colspan="2">Operation</th></tr>
<tr><th>Transmission</th><th>Reception</th></tr>
<tr><td>0</td><td>0</td><td>Output without parity bit</td><td>Received with no parity</td></tr>
<tr><td>0</td><td>1</td><td>Output with space parity (0-fixed)</td><td>No parity judgment</td></tr>
<tr><td>1</td><td>0</td><td>Output with odd parity</td><td>Judged as odd parity</td></tr>
<tr><td>1</td><td>1</td><td>Output with even parity</td><td>Judged as even parity</td></tr>
</table>
• If “Reception with no parity judgment” is selected during reception, a parity check is not performed. Therefore, since the URTEnSTR1.UTREnPE bit is not set, no error interrupt is output.<br>
• When the transmission/reception is performed in the LIN format, set URTEnSLP[1:0] to $00_B$.<br>
• Changing these bits is only allowed, if reception and transmission is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnRXE = URTEnCTL0.URTEnTXE = 0).</td></tr>
<tr><td>5</td><td>URTEn<br>TDL</td><td>Transmission data level control<br>0 No inverted output of transmit data<br>1 Inverted output of transmit data<br>
• The output level of the URTEnTTXD pin can be inverted using this bit. It inverts the URTEnTTXD output level immediately, regardless of the values of URTEnCTL0.URTEnPW and URTEnCTL0.URTEnTXE. Therefore, if URTEnTDL is set to 1 while the operation is disabled, the URTEnTTXD outputs low level.<br>
• Changing this bit is only allowed, if transmission is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnTXE = 0).</td></tr>
<tr><td>4</td><td>URTEn<br>RDL</td><td>Reception data level control<br>0 No inverted output of receive data<br>1 Inverted output of receive data<br>
• The output level of the URTEnTRXD pin can be inverted using this bit. It inverts the URTEnTRXD input level immediately, regardless of the values of URTEnCTL0.URTEnPW and URTEnCTL0.URTEnRXE. Therefore, if URTEnRDL is set to 1 while the operation is disabled, the URTEnTRXD inputs low level.<br>
• Changing this bit is only allowed, if reception is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnRXE = 0).</td></tr>
</table>

**Table 16-10 URTEnCTL1 register contents (3/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | URTEn SLG | Stop bit number selection for transmission data<br>0 1 bit<br>1 2 bits<br>• The stop bit length during data or BF reception is always handled as "1".<br>• Changing this bit is only allowed, if transmission is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnTXE = 0). |
| 1 | URTEn SLD | Transfer direction selection<br>0 MSB-first transfer<br>1 LSB-first transfer<br>• The stop bit length during data or BF reception is always handled as "1".<br>• When the transmission/reception is performed in the LIN format, set URTEnSLD to 1.<br>• Changing this bit is only allowed, if reception and transmission is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnRXE = URTEnCTL0.URTEnTXE = 0). |
| 0 | URTEn SLIT | Transmission interrupt request (INTUAEnTIT) timing selection<br>0 INTUAEnTIT generated at the start of transmission, i.e. when the transmit data is stored to the transmission shift register<br>1 INTUAEnTIT generated at transmission completion<br>• Changing this bit is only allowed, if transmission is disabled (URTEnCTL0.URTEnPW = 0 or URTEnCTL0.URTEnTXE = 0). |

### (3) URTEnCTL2 - URTEn control register 2

This register defines the baud rates of the URTEn serial data transfers.

**Access** This register can be read/written in 16-bit units.

**Address** <URTEn_base> + $24_H$

**Initial Value** $EFFF_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| URTEnPRS[2:0] | | | 0 | URTEnBRS[11:8] | | | |
| R/W | R/W | R/W | R | R/W | R/W | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| URTEnBRS[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 16-11 URTEnCTL2 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 13 | URTEn PRS[2:0] | Prescaler clock (PRSCLK) division value<br>0: PRSCLK = PCLK / $2^0$<br>1: PRSCLK = PCLK / $2^1$<br>2: PRSCLK = PCLK / $2^2$<br>3: PRSCLK = PCLK / $2^3$<br>4: PRSCLK = PCLK / $2^4$<br>5: PRSCLK = PCLK / $2^5$<br>6: PRSCLK = PCLK / $2^6$<br>7: PRSCLK = PCLK / $2^7$ |
| 11 to 0 | URTEn BRS[11:0] | Baudrate clock (BRCLK) division value (PRSCLK = PCLK / $2^{URTEnPRS[2:0]}$): |

| URTEn BRS[11:0] | Transmit/receive BRCLK | BF receive clock |
|---|---|---|
| $000_H$ | PRSCLK / (2 x 4) | PRSCLK / 4 |
| $001_H$ | | |
| $002_H$ | | |
| $003_H$ | | |
| $004_H$ | | |
| $005_H$ | PRSCLK / (2 x 5) | PRSCLK / 5 |
| ... | PRSCLK / (2 x URTEnBRS[11:0]) | PRSCLK / URTEnBRS[11:0] |
| $FFE_H$ | PRSCLK / (2 x 4094) | PRSCLK / 4094 |
| $FFF_H$ | PRSCLK / (2 x 4095) | PRSCLK / 4095 |

**Caution** Writing to this register is only allowed if the operation of URTEn is disabled (URTEnCTL0.URTEnPW = 0).

**PCLK** The value of the URTEn input clock is defined in the first section of this chapter under the key word “Clock supply”.

### (4) URTEnTRG - URTEn trigger register

This register controls the URTEn transmission/reception trigger of BF.

**Access** This register can be read/written in 8-bit units.

**Address** <URTEn_base> + $04_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | URTEn BRT | URTEn BTT | 0 | 0 | 0 | 0 | 0 |
| R | R/W | R/W | R | R | R | R | R |

**Table 16-12 URTEnTRG register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 6 | URTEn BRT | BF reception trigger<br>0: read value is always 0, writing 0 is ignored<br>1: Enable BF reception<br>• When reception is enabled, writing 1 to this bit enables BF reception (URTEnSTR0.URTEnSSBR = 1) and BF reception processing begins when the falling edge of the receive serial signal is detected.<br>• If 1 is written to this bit during reception processing, the current reception processing is terminated. Consequently, the received data is not stored, the framing, parity and overflow error bits are not updated based on the data that was being received and no interrupts are generated. Meanwhile, the BF counter value is continuously being used.<br>• After BF reception, the reception status is set according to the URTEnCTL1.URTEnSLBM setting.<br>• Setting this bit to 1 is only allowed, if reception is enabled (URTEnCTL0.URTEnPW = URTEnCTL0.URTEnRXE = 1).<br><br>After URTEnBRT is set to 1, completion of BF reception is reported by either of the following two methods, based on the URTEnCTL1.URTEnSLBM setting:<br>• if URTEnCTL1.URTEnSLBM = 0<br>When BF reception is complete, a reception interrupt request INTUAEnTIR is output.<br>• URTEnCTL1.URTEnSLBM<br>When BF reception is complete, URTEnSTR1.URTEnBSF is set to 1 and a status interrupt request INTUAEnTIS is output. |

**Table 16-12 URTEnTRG register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 5 | URTEn BTT | BF transmission trigger<br>0: read value is always 0, writing 0 is ignored<br>1: Triggers BF transmission<br>• When 1 is written to this bit while URTEnSTR0.URTEnSSBT = 0 and (URTETDCM = "H" or URTE0DCE = 0) transmission is enabled, a BF transmit request is set, and URTEnSSBT is set to 1.<br>• When 1 is written to this bit during a data transmission, a BF is transmitted after the current transmission processing is completed. Even if 1 is written to this bit before the BF transmission is completed, a BF is transmitted only once.<br>• When transmission is enabled (URTE0PW = URTE0TXE = 1), writing 1 to this bit clears all previously set data transmit requests (which have not been transmitted), leaving only BF transmit requests. If a write operation occurs in the URTE0TX7 to URTE0TX0 bits after writing 1 to this bit, data is transmitted after the BF is transmitted.<br>• If both a BF transmit request and a data transmit request have been set when a transmission starts, the BF transmission takes priority.<br>• When URTE0DCE = 1 and URTETDCM = "L", writing 1 to this bit is ignored.<br>• When reception is enabled (URTEnCTL0.URTEnPW = URTEnCTL0.URTEnRXE = 1), writing 1 to this bit enables BF reception (URTEnSTR0.URTEnSSBR = 1) and BF reception processing begins when the falling edge of the receive serial signal is detected.<br>• If 1 is written to this bit during reception processing, the current reception processing is terminated. Consequently, the received data is not stored, the framing, parity and overlow errors bits are not updated based on the data that was being received and no interrupts are generated. Meanwhile, the BF counter value is continuously being used.<br>• After BF reception, the reception status is set according to the URTEnCTL1.URTEnSLBM setting.<br>• When transmitting a BF while reception is enabled (URTEnCTL0 bits URTEnPW = URTEnRXE = 1) and BF shall not be detected during reception (URTEnCTL1.URTEnSLBM = 0), write 1 to URTEnBTT after 1 is written to URTEnBRT, then transmit the BF. If URTEnBTT is set to 1 first, a framing error may occur.<br>• Setting this bit to 1 is only allowed, if transmission is enabled (URTEnCTL0.URTEnPW = URTEnCTL0.URTEnTXE = 1). |

### (5) URTEnSTR0 - URTEn status register 0

This register indicates the current status of serial data transmissions.

**Access** This register can be read in 8-bit units.
In case the URTEn operation is disabled (URTEnCTL0.URTEnPW = 0), this register can also be written. If the URTEn is enabled (URTEnCTL0.URTEnPW = 1), any written values are disregarded and the register takes on its initial value.

**Address** <URTEn_base> + $08_H$

**Initial Value** $0000_H$. This register is initialized by any reset and when URTEn operation is disabled by URTEnCTL0.URTEnPW = 0..

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | URTEn SSBR[a] | URTEn SSBT[b] | 0 | 0 | 0 | URTEn SSR[b] | URTEn SST[b] |
| R | R | R | R | R | R | R | R |

a) This bit is also initialized if reception is disabled by URTEnCTL0.URTEnRXE = 0.

b) These bits are also initialized if transmission is disabled by URTEnCTL0.URTEnTXE = 0.

**Table 16-13 URTEnSTR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 6 | URTEn SSBR | BF reception enable status<br>0: BF reception disabled<br>1: BF reception has been enabled by setting URTEnTRG.URTEnBRT to 1 (BF reception standby mode or BF reception busy). |
| 5 | URTEn SSBT | BF transmission enable status<br>0: BF transmission disabled<br>1: BF transmission has been enabled by setting URTEnTRG.URTEnBTT to 1 (BF transmission standby mode or BF transmission busy). |
| 1 | URTEn SSR | Data reception status<br>0: no data reception ongoing<br>1: data reception ongoing (data reception busy) |
| 0 | URTEn SST | Data transmission status<br>0: no transmission pending or ongoing<br>1: data in URTEnTX[7:0] pending to be transmitted or trasmission ongoing |

### (6) URTEnSTR1 - URTEn status register 1

This register indicates results of serial data transmisssions.

**Access** This register can be read in 8-bit units.
In case the URTEn operation is disabled (URTEnCTL0.URTEnPW = 0), this register can also be written. If the URTEn is enabled (URTEnCTL0.URTEnPW = 1), any written values are disregarded and the register takes on its initial value.

**Address** <URTEn_base> + $0C_H$

**Initial Value** $0000_H$. This register is initialized by any reset and when URTEn operation is disabled by URTEnCTL0.URTEnPW = 0..

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | URTEn BSF[a] | URTEn DCE[b] | URTEn PE[a] | URTEn FE[a] | URTEn OVE[a] |
| R | R | R | R | R | R | R | R |

a) These bits are also initialized if reception is disabled by URTEnCTL0.URTEnRXE = 0.

b) This bit is also initialized if transmission is disabled by URTEnCTL0.URTEnTXE = 0.

**Table 16-14 URTEnSTR1 register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 4 | URTEn BSF | BF reception successful flag<br>0: no valid BF frame received<br>1: BF reception completed successfully<br>When the BF receive mode selection bit is set in LIN communication mode, it is necessary to read this bit by the status interrupt processing and to confirm the beginning of a new frame slot.<br>URTEnBSF is cleared by<br>• URTEnCTL0.URTEPW = 1<br>• URTEnCTL0.URTERXE = 0<br>• URTEnCTL1.URTESLBM = 0<br>• URTEnSTC.URTEnCLBS = 1 |
| 3 | URTEn DCE | Data consistency error flag<br>0: no data consistency check error detected<br>1: data consistency check error detected<br>URTEnDCE is cleared by<br>• URTEnCTL0.URTEPW = 0<br>• URTEnCTL0.URTETXE = 0<br>• URTEnCTL1.URTESLDC = 0<br>• URTEnSTC.URTEnCLDC = 1 |
| 2 | URTEn PE | Parity error flag<br>0: no parity error in received data detected<br>1: parity error in received data detected<br>The operation of URTEnPE is controlled by the settings of the URTEn.URTEnSLP[1:0].<br>URTEnPE is cleared by<br>• URTEnCTL0.URTEPW = 0<br>• URTEnCTL0.URTERXE = 0<br>• URTEnSTC.URTEnCLP = 1 |

**Table 16-14 URTEnSTR1 register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | URTEn FE | Framing error flag<br>0: cleared by<br>- URTEnCTL0.URTEPW = 0<br>- URTEnCTL0.URTERXE = 0<br>- URTEnSTC.URTEnCLF = 1<br>1: framing error in received data detected |
| 0 | URTEn OVE | Overrun error flag<br>0: cleared by<br>- URTEnCTL0.URTEPW = 0<br>- URTEnCTL0.URTERXE = 0<br>- URTEnSTC.URTEnCLOV = 1<br>1: overrun error in received data detected<br>When an overrun error occurs, the data is discarded without the next receive data being written to the receive data register URTEnRX. |

**Note** If the bits of these registers are set (1) and cleared (0) at the same time, setting takes priority over clearing.

For further information concerning error detections, refer to the sections *"Transmission data consistency check"* and *"Reception errors"*.

**Caution** In case reception and transmission is enabled and a data consistency check error occurs (URTEnSTR1.URTEnDCE = 1), follow the procedure below prior next data transmission:

- disable transmission by URTEnCTL0.URTEnTXE = 0
- initiate a transmission by URTEnTRG.URTEnBTT = 1 (BT transmission trigger) or writing any data to URTEnTX
- enable transmission by URTEnCTL0.URTEnTXE = 1

Afterwards new transmissions can be started.

### (7) URTEnSTC - URTEn status clear register

This register is used to clear the status bits of the status register 1 URTEnSTR1.

**Access** This register can be read/written in 8-bit units.
Reading this register returns always $00_H$.

**Address** <URTEn_base> $+10_H$

**Initial Value** $0000_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | URTEn CLBS | URTEn CLDC | URTEn CLP | URTEn CLF | URTEn CLOV |
| R | R | R | R/W | R/W | R/W | R/W | R/W |

**Table 16-15 URTEnSTC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 4 | URTEn CLBS | Clear BF reception successful flag<br>0: writing 0 is ignored<br>1: writing 1 clears URTEnSTR1.URTEnBSF |
| 3 | URTEn CLDC | Clear data consistency error flag<br>0: writing 0 is ignored<br>1: writing 1 clears URTEnSTR1.URTEnDCE<br>In case URTEnDCE will be cleared by setting URTEnCLDC = 1, any pending data or BF transmit requests will be ignored. |
| 2 | URTEn CLP | Clear parity error flag<br>0: writing 0 is ignored<br>1: writing 1 clears URTEnSTR1.URTEnPE |
| 1 | URTEn CLF | Clear framing error flag<br>0: writing 0 is ignored<br>1: writing 1 clears URTEnSTR1.URTEnFE |
| 0 | URTEn CLOV | Clear overrun error flag<br>0: writing 0 is ignored<br>1: writing 1 clears URTEnSTR1.URTEnOVE |

### (8) URTEnRX - URTEn receive data register

This register stores received data.

The data stored in the receive shift register is transferred to URTEnRX upon completion of reception of 1 byte of data.

**7-bit transfers** If the data length has been specified as 7 bits (URTEnCTL1.URTEnCLG = 0) and

- LSB-first reception (URTEnCTL1.URTEnSLD = 1),
  the receive data is transferred to URTEnRX[6:0] and the MSB URTEnRX[7] always becomes 0.
- MSB-first reception (URTEnCTL1.URTEnSLD = 0),
  the receive data is transferred to URTEnRX[7:1] and the LSB URTEnRX[0] always becomes 0.

For further information on data formats, refer to the section *"Data formats"*.

**Overrun error** When an overrun error (URTEnSTR1.URTEnOVE = 1) occurs, the receive data at this time is not transferred to URTEnRX and is discarded.

When reception processing ends and data reception is confirmed without any overrun errors, the received data is stored to URTEnRX according to the specified storage format.
Access

This register can be read in 8-bit units.
In case the URTEn operation is disabled (URTEnCTL0.URTEnPW = 0), this register can also be written. If the URTEn is enabled (URTEnCTL0.URTEnPW = 1), any written values are disregarded and the register takes on its initial value.

**Access** This register can be read in 8-bit units.

**Address** <URTEn_base> + $14_H$

**Initial Value** $FF_H$. This register is initialized by any reset and when URTEn operation is enabled by URTEnCTL0.URTEnPW = 1.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| URTEnRX[7:0] | | | | | | | |
| R | R | R | R | R | R | R | R |

**Table 16-16 URTEnRx register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | URTEn RX[7:0] | URTEn receive data |

### (9) URTEnTX - URTEn transmit data register

This register is used to stores data to be transmitted.

Transmit data in URTEnTX is stored to the transmission shift register URTEnTXS according to the specified transmit data format.

**7-bit transfers** If the data length has been specified as 7 bits (URTEnCTL1.URTEnCLG = 0) and

- LSB-first transmission (URTEnCTL1.URTEnSLD = 1), URTEnTX[6:0] with MSB URTEnTX[7] always “0” is transferred to the shift register URTEnTXS.
- MSB-first reception (URTEnCTL1.URTEnSLD = 0), URTEnTX[7:1] with LSB URTEnTX[0] always “0” is transferred to the shift register URTEnTXS.

For further information on data formats, refer to the section *“Data formats”*.

**Access** This register can be read/written in 8-bit units.

**Address** <URTEn_base> + $18_H$

**Initial Value** $FF_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| URTEnTX[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 16-17 URTEnTx register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | URTEn TX[7:0] | URTEn transmit data |

When transmission is enabled (URTEnCTL0 bits URTEnPW = URTETXE = 1), a write to URTEnTX triggers the start of the transmission.

**Note** If the the next data to URTEnTX is written before the ongoing transmission is completed, the continuous transmission is enabled.

### (10) URTEnEMU - URTEn emulation register

This register controls whether the URTEn can be stopped during emulation, for instance upon a breakpoint hit.

**Access** This register can be read/written in 8-bit units.

**Address** <URTEn_base> + $34_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| URTEn SVSDIS | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 16-18 URTEnEMU register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | URTEn SVSDIS | Emulation control<br>0: URTEn can be stopped during emulation<br>1: URTEn continuous operating during emulation |

## 16.5 Interrupt Request Signals

The following four interrupt request signals are generated by URTEn.

- Transmission interrupt request INTUAEnTIT
- Reception interrupt request INTUAEnTIR
- Status interrupt request INTUAEnTIS

### 16.5.1 Transmission interrupt request INTUAEnTIT

If transmit data is transferred from the URTEnTX register to transmit shift register with transmission enabled, the transmission interrupt request INTUAEnTIT is generated.

The condition for generation of a transmit interrupt request depends on the setting of the URTEnCTL1.URTEnSLIT:

- at start of transmission process: URTEnCTL1.URTEnSLIT = 0

A transmission interrupt request is issued when starting transmission of the first bit (this is the start bit for data transmission or the first BF bit for BF transmission). During data transmission, a transmission interrupt request is issued when transmit data in URTEnTX is transferred to the transmission shift register.

- at end of transmission process: URTEnCTL1.URTEnSLIT = 1

A transmission interrupt request is issued after completing transmission of the last bit (the first bit of the stop bit when the stop bit length is 1, or the second bit of the stop bit when the stop bit length is 2).

**Note** If a data consistency error is detecting, this interrupt is not generated.

The following diagrams show the timing of the transmission interrupt request for both cases.

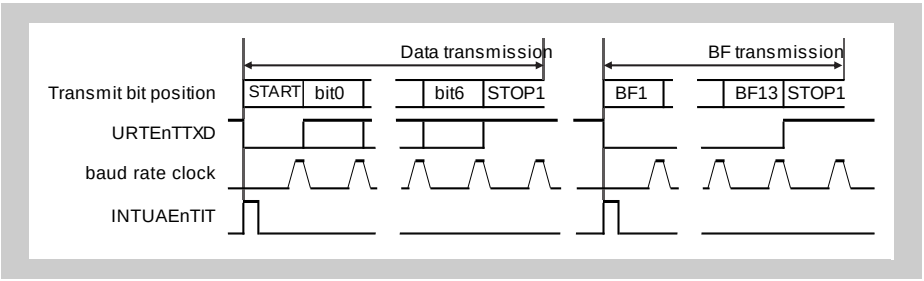



**Figure 16-3 Transmission interrupt request timing for URTEnCTL1.URTEnSLIT = 0**

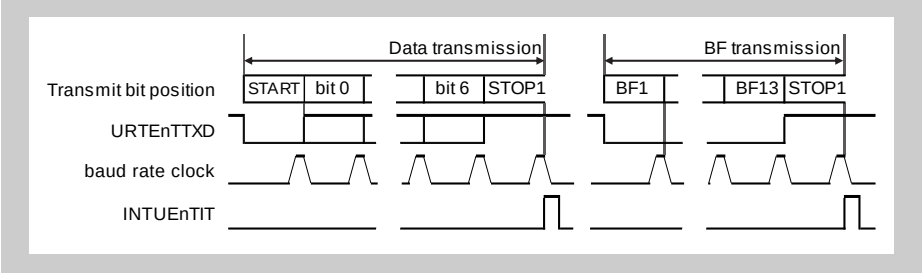



**Figure 16-4 Transmission interrupt request timing for URTEnCTL1.URTEnSLIT = 1**

### 16.5.2 Reception interrupt request INTUAEnTIR

In case of erroneous reception, the status interrupt INTUAEnTIS is generated instead of INTUAEnTIR.

No reception interrupt request INTUAEnTIR is generated in the reception disabled status.

A reception interrupt request is issued when the first bit of the stop bit is sampled (at the end of reception processing).

The following diagrams show the timing of the reception interrupt request during data/BF reception.

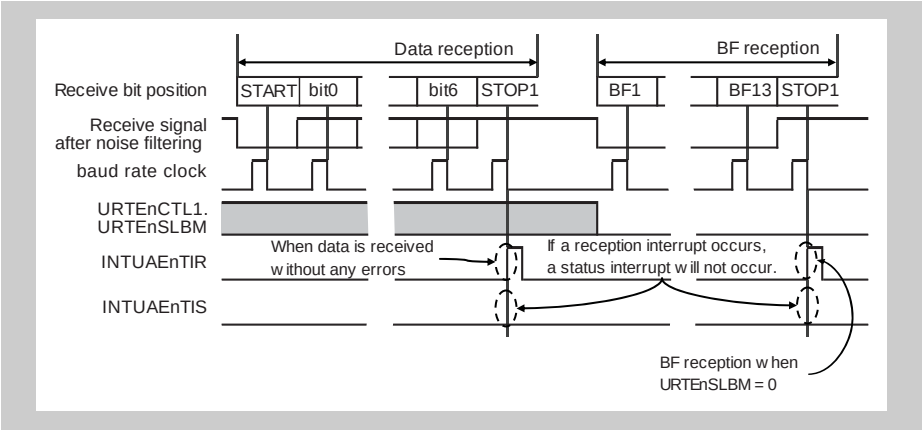



**Figure 16-5 Reception interrupt request timing**

### 16.5.3 Status interrupt request INTUAEnTIS

A status interrupt request is generated if an error condition occurred during reception or transmission, as reflected in the status register 1 URTEnSTR1.

When the BF reception mode selection bit is set in LIN communication mode (URTEnCTL1.URTEnSLBM = 1), the status interrupt request signal is generated when a consecutive low level (BF) of 11 bits or more is received.

## 16.6 Operation

### 16.6.1 Data formats

Full-duplex serial data reception and transmission is performed.

As shown in the figures below, one data frame of transmit/receive data consists of a start bit, character bits, parity bit, and stop bit(s).

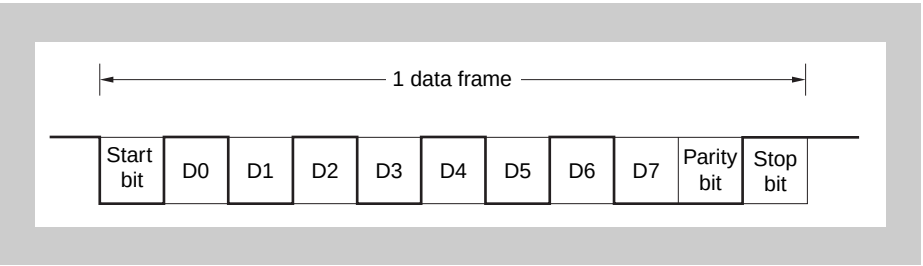
Several properties of a transmit/receive data frame can be specified by control bits of the URTEnCTL1 register:

**Table 16-19 Data format specification**

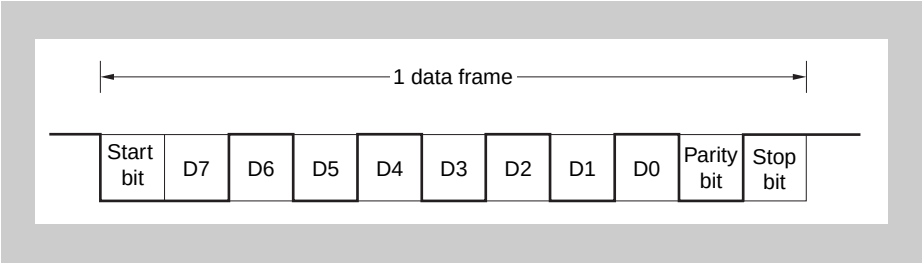
| Item | Options | Control bits |
|---|---|---|
| Start bit | 1 bit | fixed |
| Character bits | 7 bits / 8 bits | URTEnCTL1.URTEnCLG |
| Parity | Even parity/odd parity/0 | URTEnCTL1.URTEnSLP[1:0] |
| Stop bit | 1 bit / 2 bits | URTEnCTL1.URTEnSLG |
| Data order | MSB first / LSB first | URTEnCTL1.URTEnSLD |
| Tx data level | inverted / not inverted | URTEnCTL1.URTEnSLG |
| Rx data level | inverted / not inverted | URTEnCTL1.URTEnRLG |

**(1) URTEn transmit/receive data format**

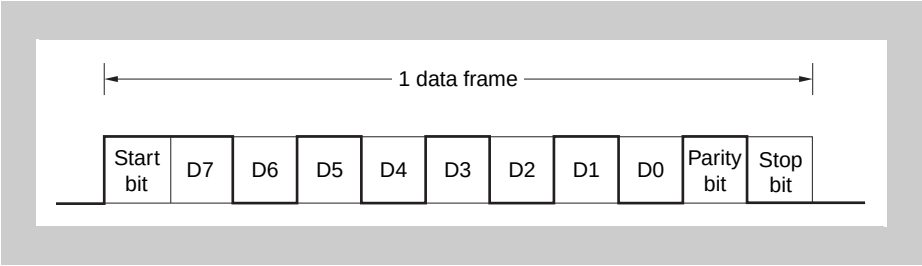
**(a) 8-bit data length, LSB first, even parity, 1 stop bit, transfer data: $55_H$**



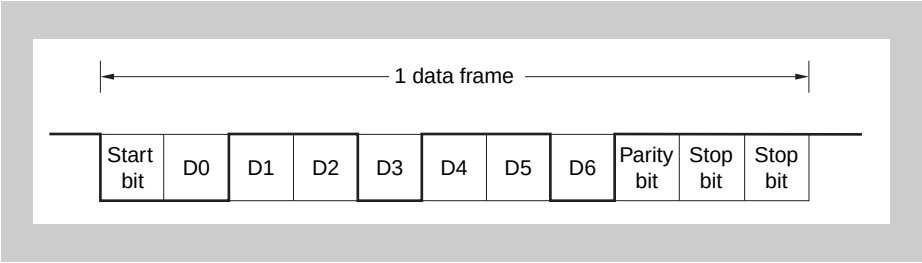
**(b) 8-bit data length, MSB first, even parity, 1 stop bit, transfer data: $55_H$**



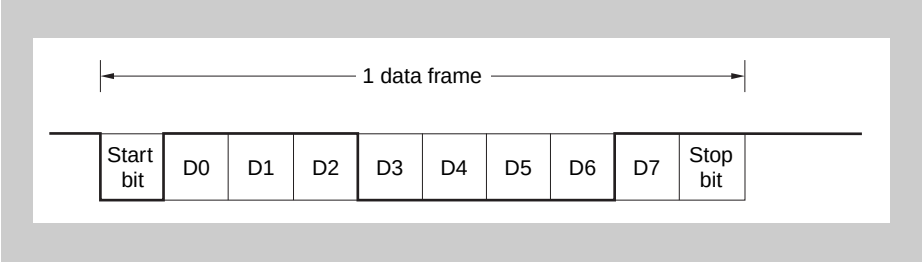

**(c) 8-bit data length, MSB first, even parity, 1 stop bit, transfer data: $55_H$, URTEnTTXD inversion**




**(d) 7-bit data length, LSB first, odd parity, 2 stop bits, transfer data: $36_H$**




**(e) 8-bit data length, LSB first, no parity, 1 stop bit, transfer data: $87_H$**



## 16.6.2 BF transmission/reception format

The URTEn has a BF (Break Field) transmission/reception control function to enable use of the LIN function.

**About LIN**

LIN stands for Local Interconnect Network and is a low-speed (1 to 20 kbps) serial communication protocol intended to aid the cost reduction of an automotive network.

LIN communication is single-master communication, and up to 15 slaves can be connected to one master.

The LIN slaves are used to control the switches, actuators, and sensors, and these are connected to the LIN master via the LIN network.

Normally, the LIN master is connected to a network such as CAN (Controller Area Network).

In addition, the LIN bus uses a single-wire method and is connected to the nodes via a transceiver that complies with ISO9141.

In the LIN protocol, the master transmits a frame with baud rate information and the slave receives it and corrects the baud rate error. Therefore, communication is possible when the baud rate error in the slave is ±14% or less.

The *Figure 16-6 "LIN transmission outline"* and *Figure 16-7 "LIN reception outline"* below outline the transmission and reception manipulations of LIN.

### (1) LIN transmission outline

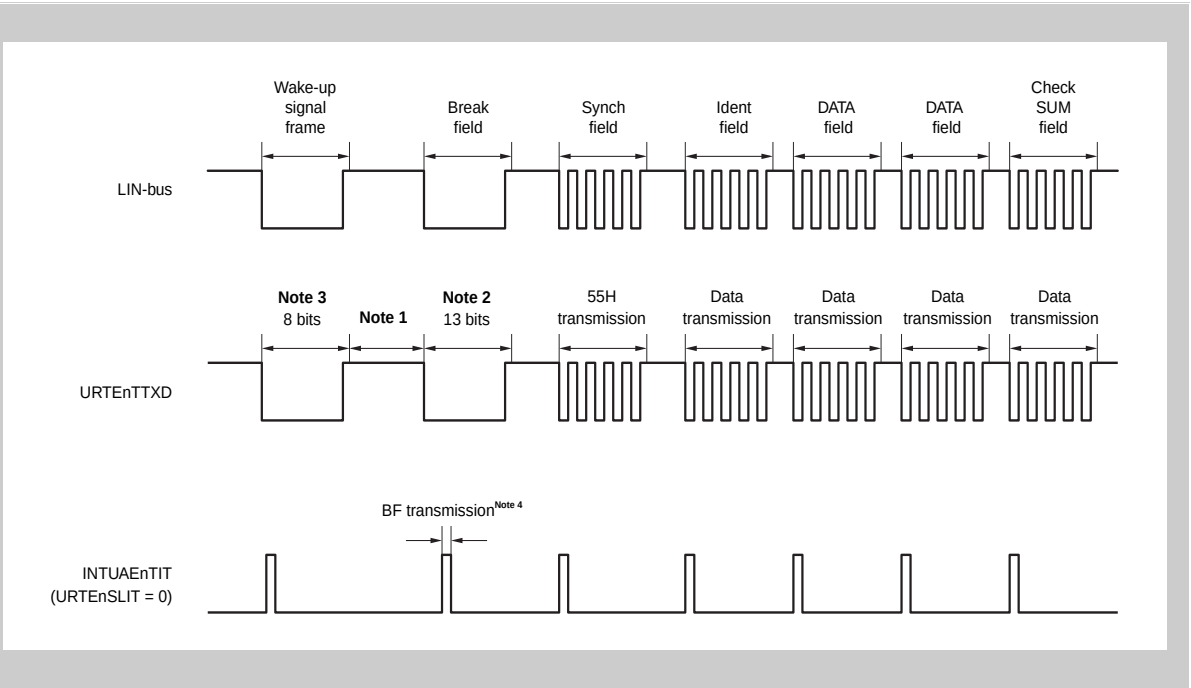



**Figure 16-6 LIN transmission outline**

**Notes**

1. The interval between each field is controlled by software.
2. BF output is performed by hardware. The output width is the bit length set by URTEnCTL1.URTEnBLG[2:0].
   If even finer output width adjustments are required, such adjustments can be performed using URTEnCTLn.URTEnBRS[11:0].

3. $80_H$ transfer in the 8-bit mode is substituted for the wakeup signal frame.
4. A transmission enable interrupt INTUAEnTIT is generated at the start of each transmission. INTUAEnTIT is also generated at the start of each BF transmission.

### (2) LIN reception outline

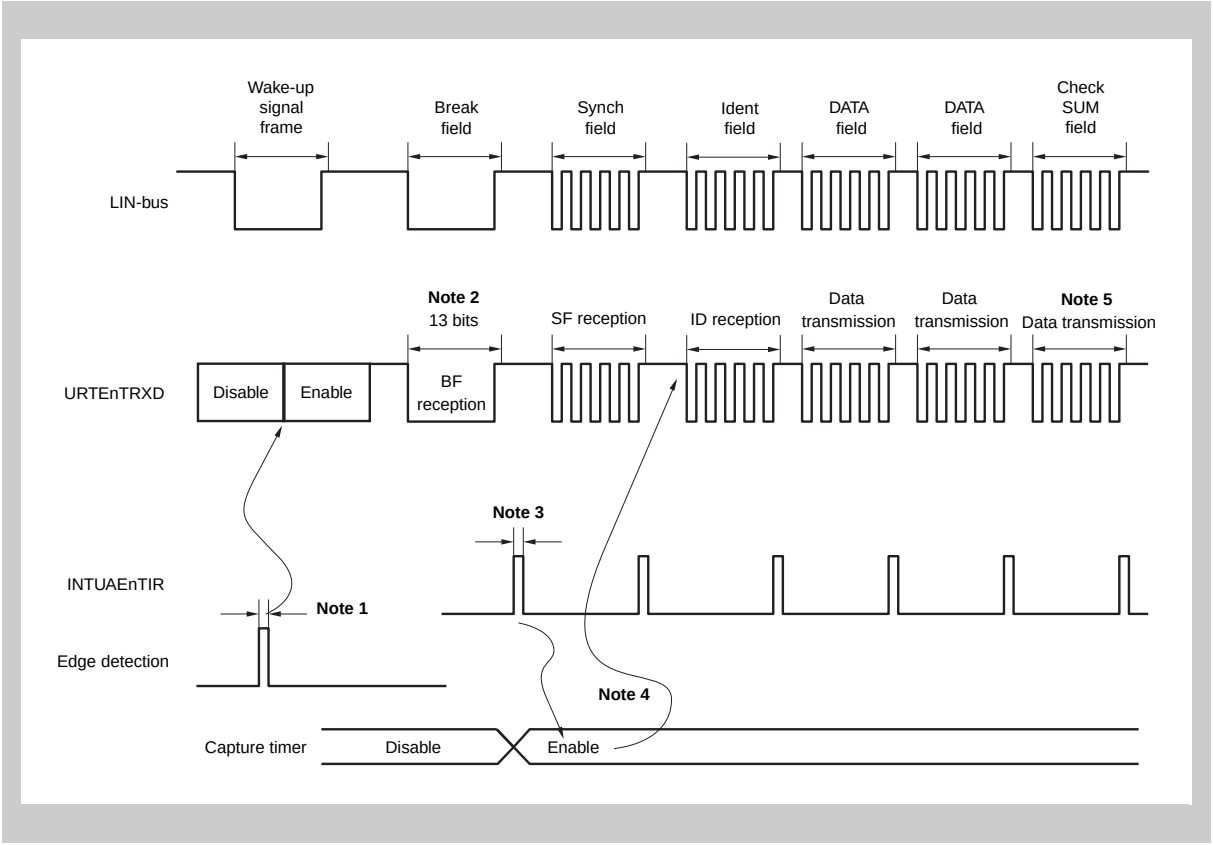



**Figure 16-7 LIN reception outline**

**Notes**

1. The wakeup signal is sent by the port edge detector, URTEn is enabled, and the BF reception mode is set.
2. Upon detection of the BF reception of 11 or more bits, normal BF reception end is judged.
   Upon detection of BF reception of less than 11 bits, a BF reception error is judged, no interrupt is generated, and the mode returns to the BF reception mode.
3. When BF reception ends normally and reception mode selection bit URTEnCTL1.URTEnSLBM
   - is set to "0"
     the reception interrupt INTUAEnTIR is generated
   - is set to "1"
     the status interrupt INTUAEnTIS is generated and the BF reception success flag URTEnSTR1.URTEnBSF is set.

   If the BF reception trigger bit URTEnTRG.URTEnBRT = 1, the error detection for the overrun, parity, and framing is not performed during the BF reception. Moreover, the data transfer from the receive shift register URTEnRXS to the receive data register URTEnRX is not performed, either. At this time, the URTEnRX holds the prior value.

4. In order to adjust the baud rate clock properly, the URTEnTRXD signal needs to be connected to the capture input of a timer. By measuring the time between two URTEnTRXD edges the transfer rate and the baud rate error can be calculated and the baud rate can be adjusted accordingly via the baud rate setting URTEnCTL2.URTEnBRS[11:0].
5. Check-sum field (CSF) distinctions are made by software. URTEn is initialized following CSF reception, and the processing for setting the BF reception mode again is performed by software. When URTEnCTL1.URTEnSLBM = 1, BF reception can be performed automatically without setting to BF reception mode again.

### 16.6.3 BF transmission

When the URTEnCTL0 bits URTEnPW = URTEnTXE = 1, the transmission enabled status is entered, and BF transmission is started by setting the BF transmission trigger URTEnTRG.URTEnBTT = 1.

Thereafter, URTEnSTR0.URTEnSSBT is set to “1” and a low level width of 13 to 20 bits, as specified by URTEnCTL1.URTEnBLG[2:0], is output.
A transmission interrupt INTUAEnTIT is generated upon BF

- transmission start, if URTEnCTL1.URTEnSLIT = 0
- transmission end, if URTEnCTL1.URTEnSLIT = 1.

Following the end of BF transmission, URTEnSTR0.URTEnSSBT is automatically cleared. Thereafter, the URTEn transmission mode is restored.

Transmission is suspended until the data to be transmitted next is written to the URTEnTX register, or until the BF transmission trigger URTEnTRG.URTEnBTT is set to 1 and URTEnSTR0.URTEnSSBT changes to 1.

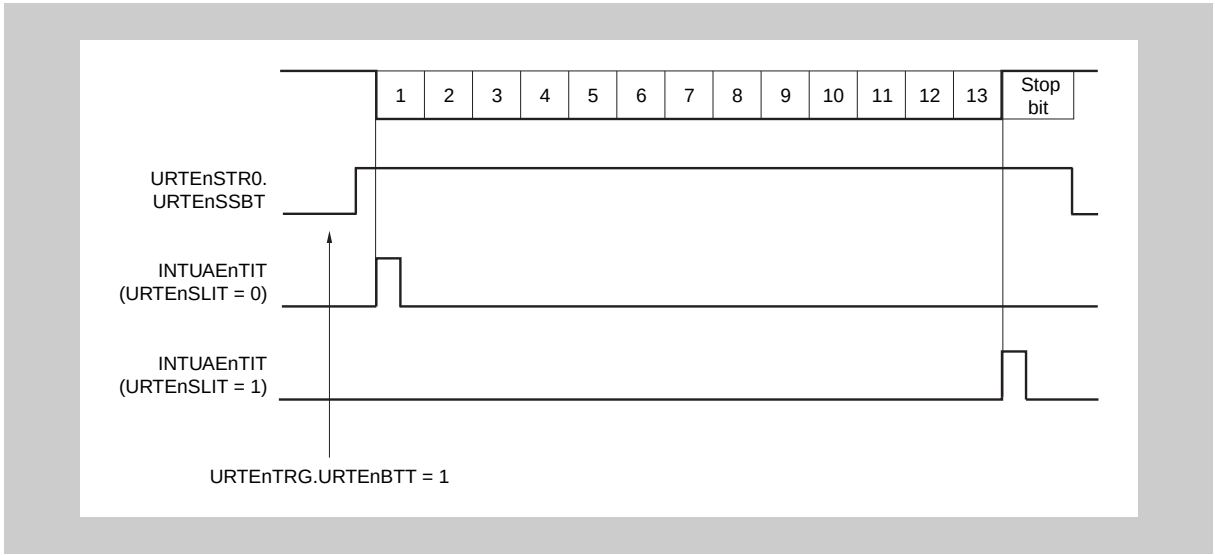



**Figure 16-8 BF transmission**

### 16.6.4 BF reception

The reception enabled status is achieved by setting the URTEnCTL0.URTEnCTL0.URTEnPW bit to 1 and then setting the URTEnCTL0.URTEnRXE bit to 1.

The BF reception wait status is set by setting the BF reception trigger URTEnTRG.URTEnBRT = 1. URTEnSTR0.URTEnSSBR is set to 1.

In the BF reception wait status, similarly to the URTEn reception wait status, the URTEnTRXD pin is monitored and start bit detection is performed.

Following detection of the low level, reception is started and the internal counter counts up according to the set baud rate.

When a low level is received for minimum 11 bits (valid BF reception), while the BF receiving mode selection bit

- URTEnCTL1.URTEnSLBM = 0 (BF Rx during data Rx disabled), the reception interrupt INTUAEnTIR is generated.
- URTEnCTL1.URTEnSLBM = 1 (BF Rx during data Rx enabled), the status interrupt INTUAEnTIS is generated and BF reception success flag URTEnSTR1.URTEnBSF is set at the same time

The URTEnSTR0.URTEnSSBR bit is automatically cleared and BF reception ends.

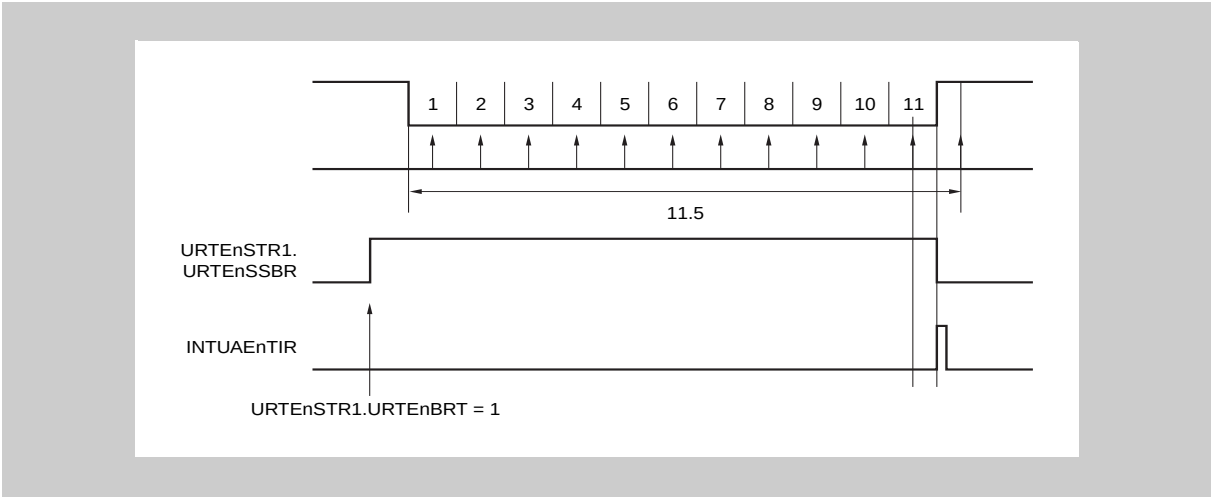



**Figure 16-9 Normal BF reception (stop bit after more than 10.5 “L” bits)**

Error detection for the URTEnSTR1 error flags URTEnOVE, URTEnPE, and URTEnFE is suppressed and URTEn communication error detection processing is not performed.
Moreover, the erroneous data is not stored in URTEnRX, but the initial value $FF_H$ is held.

If the BF width is 10 or fewer bits, reception is terminated as error processing without generating an interrupt, and the BF reception mode is returned to. URTEnSTR0.URTEnSSBR is not cleared at this time.

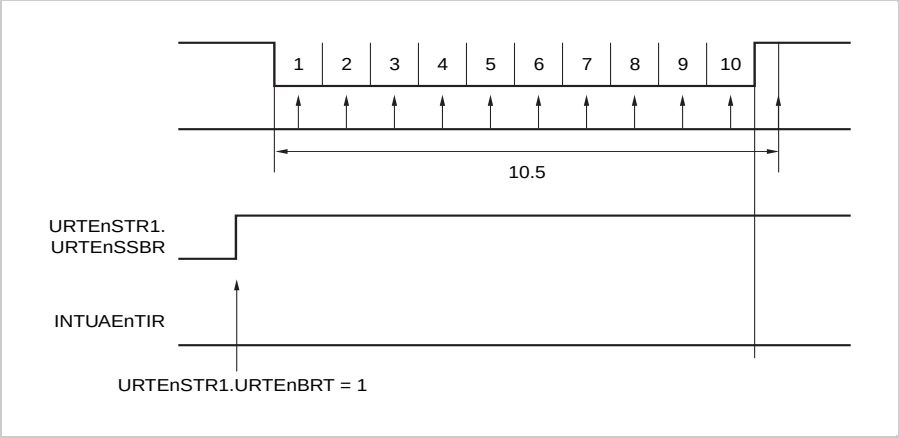



**Figure 16-10 BF reception error (stop bit within 10.5 “L” bits)**

The BF mode can be selected between a single BF receive mode and an any time BF receive mode in by URTEnCTL1.URTEnSLBM. The status of a successful reception of the BF is shown URTEnSTR1.URTEnBSF.

**Note** URTEnSTR0.URTEnSSBR is set to “1”

- by setting URTEnTRG.URTEnBRT to “1"
- cleared by normal BF reception.

### 16.6.5 Transmission data consistency check

The URTEn incorporates a data consistency check function to detect a mismatch between the transmit data written to transmit register URTEnTX and the data on the bus when the device operates in master mode.

Data consistency check is enabled by URTEnCTL0.URTEnSLDC = 1.

The data consistency is checked by comparing the transmit data in the transmit register URTEnTX and the receive data in the receive register URTEnRX. In case of a mismatch the data consistency error flag URTEnSTR1.URTEnDCE is set and a status interrupt request INTUAEnTIS occurs.

The data consistency check of data is not done in reception mode.

The data consistency check of the send data and the input data terminal level is done even if the reception is disabled during sending. In that case also the reception completion interrupt request signal INTUAEnTIR, the URTEnSTR1 status bits URTEnBSF, URTEnFE, URTEnOVE and the status interrupt request signal INTUAEnTIS will not be generated as well. Receive data does not need to be read.

Refer to the description of the *"URTEnSTR1 - URTEn status register 1"* for details.

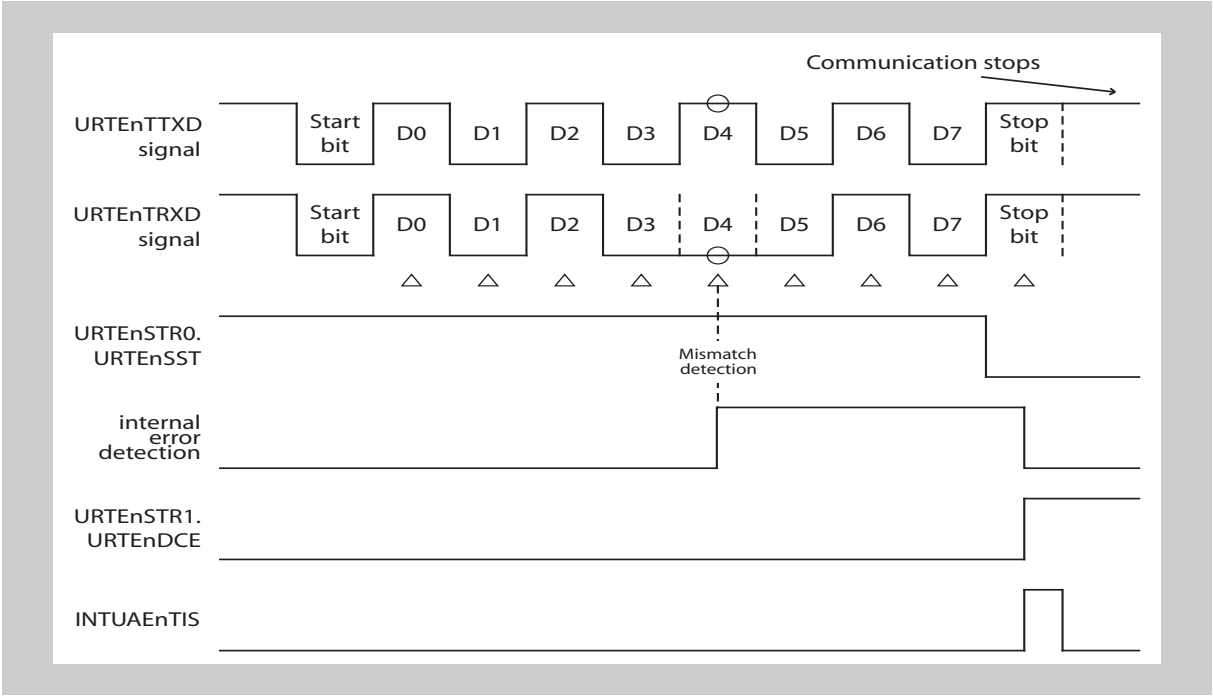



**Figure 16-11 Timing example of data consistency error (no BF reception active, i.e. URTEnSTR0.URTEnSSBR = 0)**

### 16.6.6 URTEn transmission

**Transmission start**

Set the transmission enabled status by performing the following procedures.

- Specify the baud rate by the URTEn control register 2 URTEnCTL2.
- Specify the transmit parity, data character length, stop bit length, transmit data order, transmission interrupt request timing and output logic level by the URTEn control register 1 URTEnCTL1.
- Enable URTEn operation and transmission by URTEnCTL0.URTEnPW = URTEnCTL0.URTEnTXE = 1)

Write of the transmit data to the transmission buffer register URTEnTX starts transmission. The data which is saved in the URTEnTX register is transferred to the transmit shift register URTEnTXS. Then, the start, parity and stop bits are added and the data frame is output serially via URTEnTTXD.

**Transmission stop**

When URTEnCTL0.URTEnPW or URTEnCTL0.URTEnTXE is set to 0, transmission operations are stopped immediately, even during transmission processing.

**Concurrent BF and data transmission**

When a BF transmit request and a data transmit request have both been set, BF transmission takes priority.

**Data consistency check**

When URTETDCM = "L", if a data consistency error is detected, the subsequent data is not transmitted until URTE0CLDC = 1, URTE0PW = 0, or URTE0TXE = 0 is written.

**INTUAEnTIT timing**

The time to generate the transmission interrupt INTUAEnTIT depends on the setting of the URTEnCTL1.URTEnSLIT bit:

- URTEnCTL1.URTEnSLIT = 0
  INTUAEnTIT is generated at the start of transmission, i.e. when the data from the data register URTEnTX is transferred to the transmit shift register URTEnTXS and transmission is started.
- URTEnCTL1.URTEnSLIT = 1
  INTUAEnTIT is generated when the entire data transmission process is completed, i.e. when the last bit of the data frame has been transitted.

Once INTUAEnTIT is generated, the next data can be written to URTEnTX.

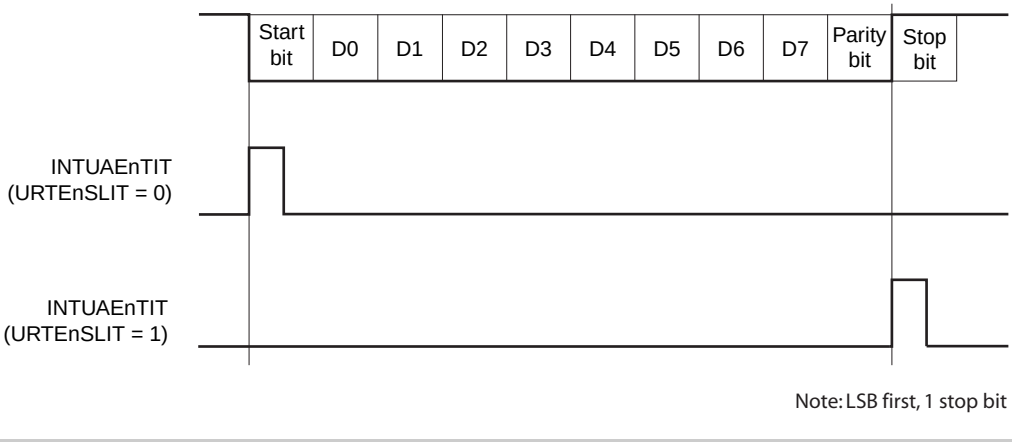



**Figure 16-12 Transmission interrupt timing**

### 16.6.7 Continuous transmission procedure

Continuous transmission is achieved by writing the next transmit data to the transmit data register URTEnTX, while shifting out of the previous data from the transmission shift register URTEnTXS is ongoing.

**Note** In order to maintain correct write timing, the transmission interrupt INTUAEnTIT must be generated at the start of each transmission (URTEnCTL1.URTEnSLIT = 0).

**Caution** If the value is written to the URTEnTX register before the INTUAEnTIT is generated, the transmit data set before is overwritten by the new transmit data.

To initialize the transmission unit, confirm that no transmission is ongoing (URTEnSTR0 bits URTEnSSBT = URTEnSST = 0). If the initialization is performed during an ongoing transmission, the transmission is aborted.

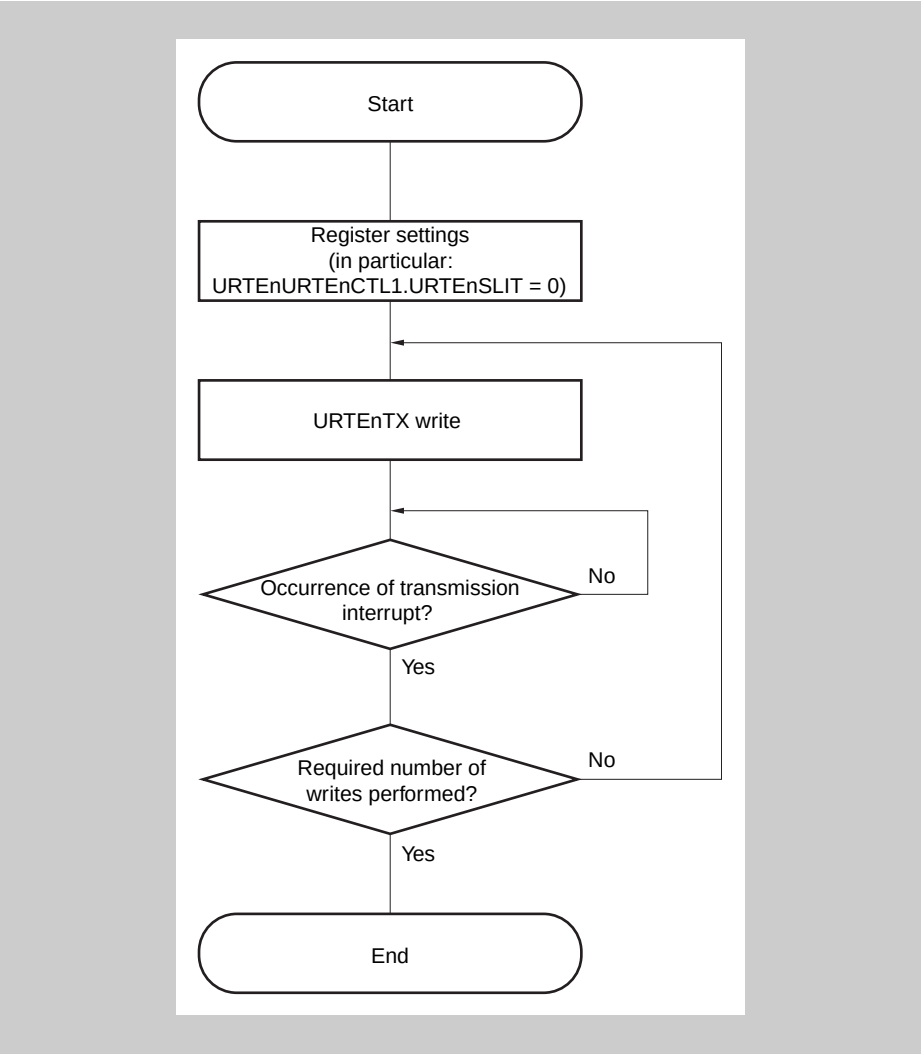



**Figure 16-13 Continuous transmission processing flow**

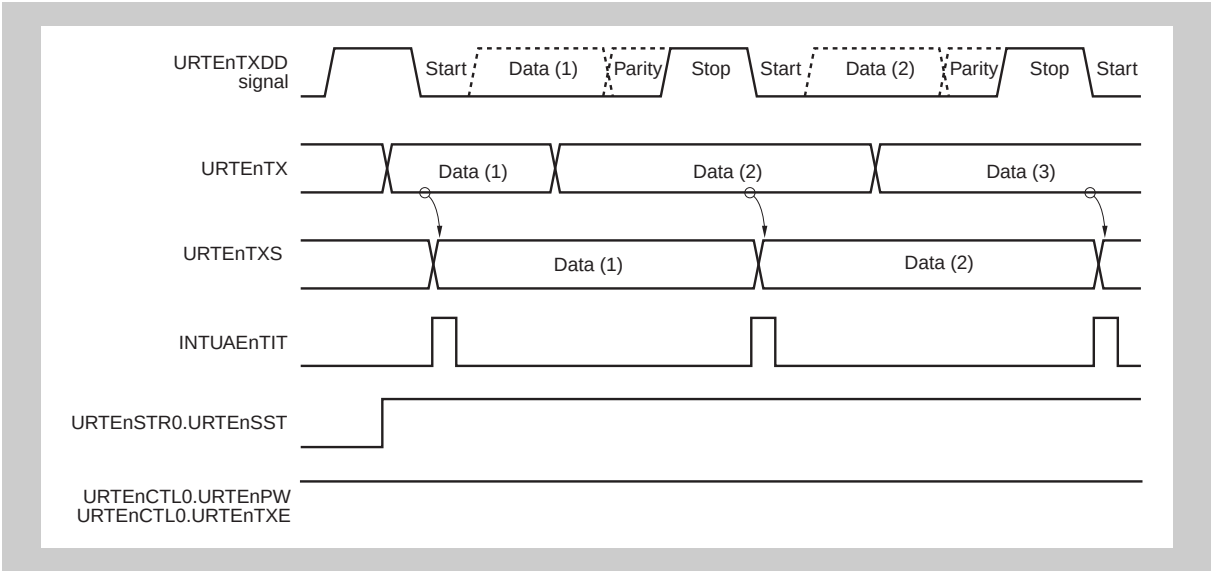



**Figure 16-14 Continuous transmission operation timing - transmission start**

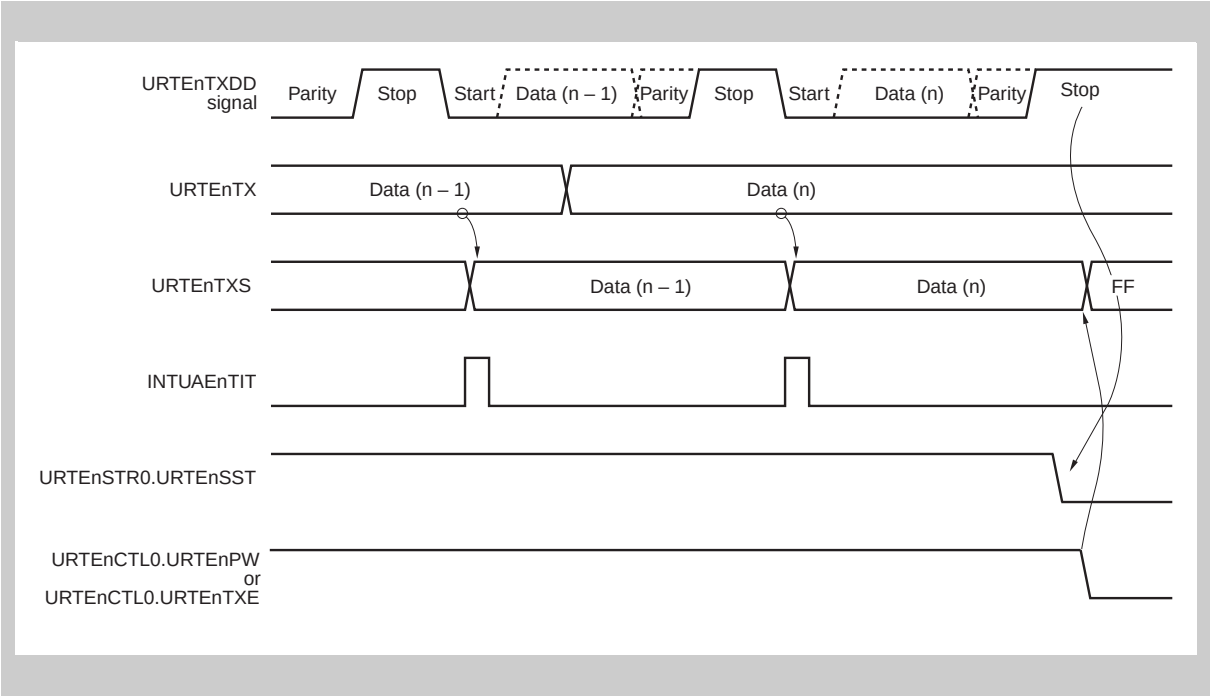



**Figure 16-15 Continuous transmission operation timing - transmission end**

## 16.6.8 URTEn reception

**Reception start**

Set the reception enabled status by the following procedure:

- Specify the baud rate by the URTEn control register 2 URTEnCTL2.
- Specify the receive parity, data character length, stop bit length, receive data order and output logic level by the URTEn control register 1 URTEnCTL1.
- Enable URTEn operation and reception by URTEnCTL0.URTEnPW = URTEnCTL0.URTEnRXE = 1).

When the sampling of the input level of the URTEnTRXD pin is performed and the falling edge is detected, the data sampling of the URTEnTRXD input is started. The start bit is recognized if the URTEnTRXD pin is low level after the time of a half bit is passed after the detection of the falling edge (shown in the figure below). After a start bit has been recognized, the receive operation starts, and serial data is stored in the receive shift register according to the set baud rate. When the reception interrupt INTUAEnTIR is asserted upon reception of the stop bit, the data stored in the receive shift register URTEnRXS is written to the receive data register URTEnRX.

**Reception stop**

When URTEnCTL0.URTEnPW or URTEnCTL0.URTEnRXE is set to 0, reception operations are stopped immediately, even during reception processing.

**Reception errors**

If an overrun error occurs (URTEnSTR1.URTEnOVE = 1), the receive data at this time is not transferred to the URTEnRX register and is discarded.
Even if a parity error (URTEnSTR1.URTEnPE = 1) or a framing error (URTEnSTR1.URTEnFE = 1) occurs during reception, reception continues until the reception position of the first stop bit, and the reception data is transferred to the URTEnRX.
In any case of the reception errors, the status interrupt INTUAEnTIS is

generated after the following reception completion, but not the reception interrupt INTUAEnTIR.

When the receive data order, parity, data character length, and the stop bit length are changed, clear the power bit (URTEnCTL0.URTEnPW = 0) or clear both the transmission enabled bit and the reception enabled bit (URTEnTXE = 0, URTEnRXE = 0), and then change the setting.

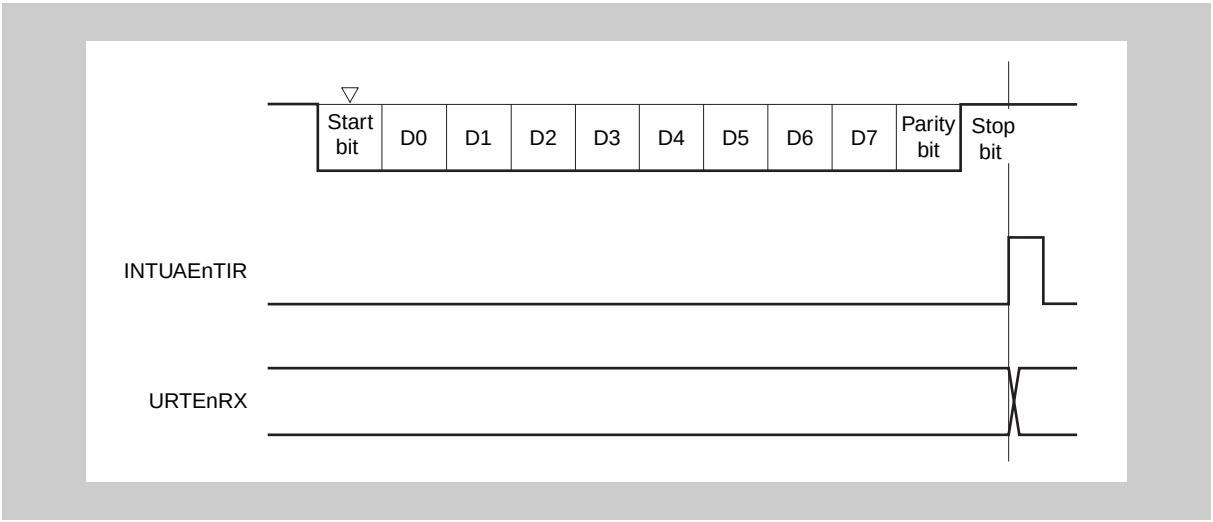



**Figure 16-16 URTEn reception**

**Cautions**

1. Be sure to read the URTEnRX register even when a reception error occurs. If the URTEnRX register is not read, an overrun error occurs during reception of the next data.
2. The operation during reception is performed assuming that there is only one stop bit. A second stop bit is ignored.
3. When reception is completed, read the URTEnRX register after the reception interrupt INTUAEnTIR has been generated, and clear the URTEnCTL0.URTEnPW or URTEnCTL0.URTEnRXE bit to 0.
   If the URTEnCTL0.URTEnPW or URTEnCTL0.URTEnRXE bit is cleared to 0 before the INTUAEnTIR is generated, the read value of the URTEnRX register cannot be guaranteed.
4. If receive completion processing (INTUAEnTIR interrupt generation) and the URTEnCTL0.URTEnPW bit = 0 or URTEnCTL0.URTEnRXE bit = 0 conflict, INTUAEnTIR may be generated in spite of these being no data stored in the URTEnRX register.

**Notes**

1. If low level is always input to the URTEnTRXD pin, it is not judged as the start bit.
2. In continuous reception, immediately after the stop bit is detected at the first reception bit (when the reception interrupt is generated), the next start bit can be detected.

### 16.6.9 Reception errors

Errors during a receive operation are of three types: parity errors, framing errors, and overrun errors. Data reception result error flags are set in the URTEnSTR1 register and a status interrupt request signal INTUAEnTIS is generated when an error occurs.

It is possible to ascertain which error occurred during reception by reading the contents of the URTEnSTR1 register.

Clear a reception error flag by writing 1 to its associated bit in the status clear register URTEnSTC.

**Table 16-20 Reception error causes**

| Error flag in URTEnSTR1 | Reception error | Cause |
|---|---|---|
| URTEnPE | Parity error | Received parity bit does not match the setting |
| URTEnFE | Framing error | Stop bit not detected |
| URTEnOVE | Overrun error | Reception of next data completed before data was read from receive buffer |

**Note** Even in case of a parity or framing error, data is transferred from the receive shift register URTEnRXS to the receive data register URTEnRX. Consequently the data from URTEnRX must be read. Otherwise an overrun error URTEnSTR1.URTEnOVE will occur at reception of the next data.

In case of an overrun error, the receive shift register data is not transferred to URTEnRX, thus the previous data is not overwritten.

### 16.6.10 Parity types and operations

**Caution** When using the LIN function, fix the URTEnCTL1.URTEnSLP[1:0] to $00_B$.

The parity bit is used to detect bit errors in the communication data. Normally the same parity is used on the transmission side and the reception side.

In the case of even parity and odd parity, it is possible to detect odd-count bit errors. In the case of 0 parity and no parity, errors cannot be detected.

**(1) Even parity**

- During transmission
  The number of bits whose value is “1” among the transmit data, including the parity bit, is controlled so as to be an even number. The parity bit values are as follows:
  - Odd number of bits whose value is “1” among transmit data:1
  - Even number of bits whose value is “1” among transmit data:0
- During reception
  The number of bits whose value is “1” among the reception data, including the parity bit, is counted, and if it is an odd number, a parity error is output.

**(2) Odd parity**

- During transmission
  Opposite to even parity, the number of bits whose value is “1” among the transmit data, including the parity bit, is controlled so that it is an odd number. The parity bit values are as follows.
  - Odd number of bits whose value is “1” among transmit data: 0
  - Even number of bits whose value is “1” among transmit data: 1
- During reception
  The number of bits whose value is “1” among the receive data, including the parity bit, is counted, and if it is an even number, a parity error is output.

**(3) 0 parity**

During transmission, the parity bit is always made 0, regardless of the transmit data.

During reception, parity bit check is not performed. Therefore, no parity error occurs, regardless of whether the parity bit is 0 or 1.

**(4) No parity**

No parity bit is added to the transmit data.

Reception is performed assuming that there is no parity bit. No parity error occurs since there is no parity bit.

### 16.6.11 Digital receive data noise filter

The receive data signal input URTEnTRXD is equipped with a digital noise filter to eliminate noise and spikes.

This filter samples the URTEnTRXD pin using the prescaler output clock PRSCLK.

When the same sampling value is read twice, the URTEnTRXD signal is validated as the input data.
Therefore, data not exceeding the width of 2 prescaler output clocks is judged to be noise and thus eliminated.

The noise filter causes a delay of 4 prescaler output clock PRSCLK cycles when capturing the serial data URTEnTRXD, until it is forwarded as valid.

## 16.7 Baud Rate Generator

The transmission and reception baudrate BRCLK are derived from the PBUS bus clock PCLK by use of a prescaler and a baudrate generator, as shown in the figure below.

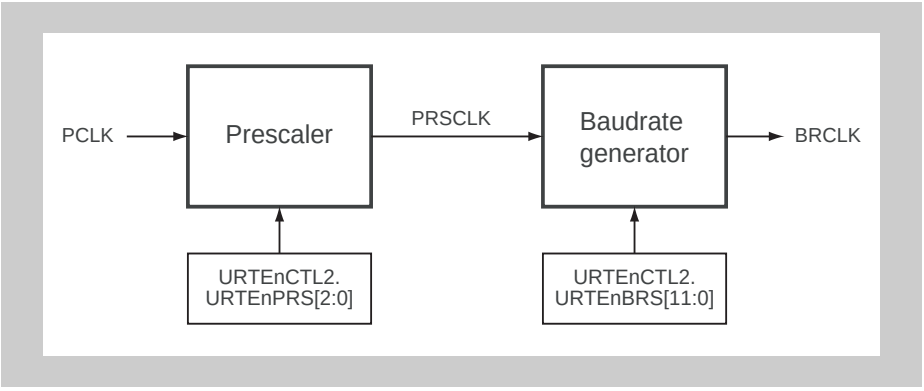



**Figure 16-17 Configuration of baud rate generator**

The prescaler output clock PRSCLK is a fraction of PCLK, the divisor is set up by the value URTEnCTL2.URTEnPRS[2:0]:

$$PRSCLK = PCLK / 2^{URTEnPRS[2:0]}$$

PRSCLK is further divided by the baudrate generator by a value, determined by URTEnCTL2.URTEnBRS[11:0].
The baudrate generator distinguishes between the baudrate for data frames and BF receptions, as listed in the table below. The BF reception clock is the double of the baudrate clock BRCLK.

**Table 16-21 Baudrate generator clocks output**

| URTEnCTL2. URTEnBRS[11:0] | Transmit/receive BRCLK | BF receive clock |
|---|---|---|
| $000_H$ | PCLK / [$2^{URTEnPRS[2:0]}$ × (2 × 4)] | PCLK / [$2^{URTEnPRS[2:0]}$ × 4] |
| $001_H$ | | |
| $002_H$ | | |
| $003_H$ | | |
| $004_H$ | | |
| $005_H$ | PCLK / [$2^{URTEnPRS[2:0]}$ × (2 × 5)] | PCLK / [$2^{URTEnPRS[2:0]}$ x 5] |
| ... | PCLK / [$2^{URTEnPRS[2:0]}$ × (2 × URTEnBRS[11:0])] | PCLK / [$2^{URTEnPRS[2:0]}$ × URTEnBRS[11:0] |
| $FFE_H$ | PCLK / [$2^{URTEnPRS[2:0]}$ × (2 × 4094)] | PCLK / [$2^{URTEnPRS[2:0]}$ × 4094] |
| $FFF_H$ | PCLK / [$2^{URTEnPRS[2:0]}$ × (2 × 4095)] | PCLK / [$2^{URTEnPRS[2:0]}$ × 4095] |

# Chapter 17 LIN Master Controller (LMA)

This chapter contains a generic description of the LIN Master Controller.

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 17.1 V850E2/Fx4-G LMAn Features

**LMAn instances**

This microcontroller has the following number of instances of the LIN Master Controller LMAn.

**Table 17-1 Instances of LMA**

| LIN Master Controller | V850E2/FF4-G | V850E2/FG4-G |
|---|---|---|
| Instance | 2 | 3 |
| Name | LMA10, LMA11 | LMA2, LMA10, LMA11 |

**LMAn instances index n**

Throughout this chapter, the instance of a LIN Master Controller is identified by the index “n” (n = 2, 10, 11), for example, LMAnCTLL for the LMAn control register L.

**CNTAm instances**

This microcontroller has following number of instances of the LIN Master Scheduler Counters CNTAm.

**Table 17-2 Instances of LIN Master Scheduler Counters CNTAm**

| LIN Master Scheduler Counter | V850E2/FF4-G | V850E2/FG4-G |
|---|---|---|
| Instance | 1 | 2 |
| Name | CNTA2 | CNTA1, CNTA2 |

**CNTAm instances index m**

Throughout this chapter, the instance of a LIN Master Scheduler Counter is identified by the index “m” (m = 1, 2), for example, CNTAmCTL for the CNTAm control register.

**LMAn register addresses**

All LMAn register addresses are given as address offsets to the individual base address <LMAn_base>.
The <LMAn_base> address of each LMAn are listed in the following table:

**Table 17-3 LMAn register base addresses <LMAn_base>**

| LMAn instance | <LMAn_base> address |
|---|---|
| LMA2 | FF5E $0080_H$ |
| LMA10 | FF66 $0080_H$ |
| LMA11 | FF67 $0080_H$ |

**CNTAm register addresses**

All CNTAm register addresses are given as address offsets to the individual base address <CNTAm_base>.
The <CNTAm_base> address of each CNTAm are listed in the following table:

**Table 17-4 CNTAm register base addresses <CNTAm_base>**

| CNTAm instance | <CNTAm_base> address |
|---|---|
| CNTA1 | FF5D $4000_H$ |
| CNTA2 | FF5E $4000_H$ |

**Clock supply** All LIN Master Controller and LIN Master Scheduler Counters provide one clock input.

**Table 17-5 LMAn/CNTAm clock supply**

| LMAn/CNTAm instance | LMAn clock | Connected to |
|---|---|---|
| LMA2 | PCLK | Clock Controller CKSCLK_034 |
| CNTA1 | PCLK | Clock Controller CKSCLK_034 |
| LMA10 | PCLK | Clock Controller CKSCLK_011 |
| LMA11 | PCLK | Clock Controller CKSCLK_011 |
| CNTA2 | PCLK | Clock Controller CKSCLK_011 |

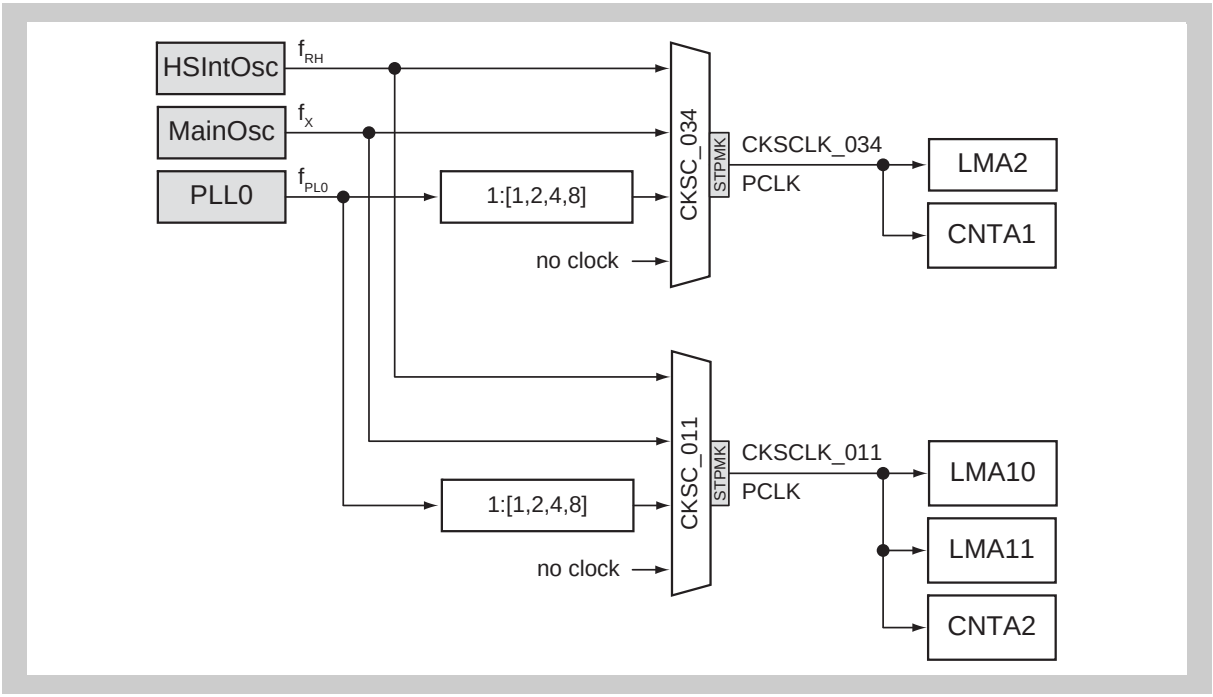



**Figure 17-1 LMAn clock supply**

**Interrupts and DMA**

The LIN Master Controllers can generate following interrupt and DMA requests:

**Table 17-6 LMAn interrupt and DMA requests**

| LMAn signals | Function | Connected to |
|---|---|---|
| **LMA2:** | | |
| INTLMA2TIT | Transmission interrupt | Interrupt Controller INTLMA2IT<br>DMA Controller trigger 52 |
| INTLMA2TIR | Reception interrupt | Interrupt Controller INTLMA2IR |
| INTLMA2TIS | Status interrupt | Interrupt Controller INTLMA2IS |
| **LMA10:** | | |
| INTLMA10TIT | Transmission interrupt | Interrupt Controller INTLMA10IT[a]<br>DMA Controller trigger 59 |
| INTLMA10TIR | Reception interrupt | Interrupt Controller INTLMA10IR[a]<br>DMA Controller trigger 58 |
| INTLMA10TIS | Status interrupt | Interrupt Controller INTLMA10IS[a] |
| **LMA11:** | | |
| INTLMA11TIT | Transmission interrupt | Interrupt Controller INTLMA11IT[a]<br>DMA Controller trigger 61 |
| INTLMA11TIR | Reception interrupt | Interrupt Controller INTLMA11IR[a]<br>DMA Controller trigger 60 |
| INTLMA11TIS | Status interrupt | Interrupt Controller INTLMA11IS[a] |

a) These interrupts can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

**LMA H/W reset**

The LIN Master Controllers and their registers are initialized by the following reset signal:

**Table 17-7 LMAn reset signal**

| LMAn | Reset signal |
|---|---|
| LMAn | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

Timer interrupt 0**LIN Master Scheduler Counters**

Following table shows the assignment of the LIN Master Controllers to the LIN Master scheduler counters:

**Table 17-8 CNTAm to LMAn assignment**

| CNTAm instance | LMAn instance |
|---|---|
| CNTA1 | LMA2 |
| CNTA2 | LMA10 to LMA11 |

For a detailed description of the LIN Master Scheduler Counters CNTAm refer to section *19.1 "LIN Master Scheduler Counters (CNTA)" on page 1* .

## 17.2 LIN Master Scheduler Counters (CNTA)

The LIN master scheduler counter consists of a free-running 16-bit counter. The count clock is derived from the CNTA input clock PCLK, that is divided by a prescaler.

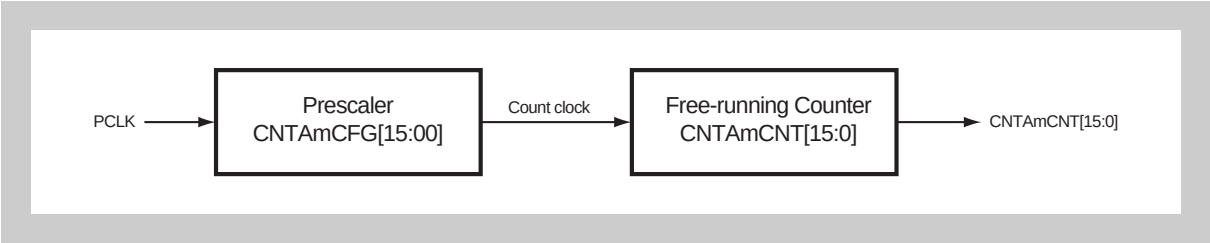



**Figure 17-2 LIN master scheduler counter**

**CNTAm enable**

Before the LIN Master Controller is enabling the scheduler, the counter CNTAm must be enabled by CNTAmCTL.CNTAmPW = 1.

**Prescaler division**

The division factor of the prescaler is determined by the value CNTAmCFG.CNTAmPRS[15:00]:

- CNTAmPRS[15:00] = $FFFF_H$: count clock = PCLK / 1
- else: count clock = PCLK / (CNTAmPRS[15:00]+2)

### 17.2.1 CNTAm registers

The CNTAm is controlled and operated by means of the following registers:

**Table 17-9 CNTAm registers**

| Register function | Name | Address |
|---|---|---|
| Control register | CNTAmCTL | <CNTAm_base> + $00_H$ |
| Configuration register | CNTAmCFG | <CNTAm_base> + $04_H$ |

**<CNTAm_base>**

The base addresses <CNTAm_base> of the CNTAm are defined in the first section of this chapter under the key word “CNTAm register addresses”.

### (1) CNTAmCTL - CNTAm control register

This register enables/disables the CNTAm operation.

**Access** This register can be read/written in 16-bit units.

**Address** <CNTAm_base> + $00_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNTAm PW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-10 CNTAmCTL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | CNTAm PW | CNTAm operation enable<br>0: CNTAm disabled<br>1: CNTAm enabled |

### (2) CNTAmCFG - CNTAm configuration register

This register sets the division factor for the clock prescaler.

**Access** This register can be read/written in 16-bit units.

**Address** <CNTAm_base> + $04_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNTAmPRS[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-11 CNTAmCTL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | CNTAm PRS[15:00] | CNTAm prescaler division ratio<br>$FFFF_H$: PCLK/1<br>$0000_H$: PCLK/2<br>$0001_H$: PCLK/3<br>$0002_H$: PCLK/4<br>...<br>$FFFE_H$: PCLK/65536 |

## 17.3 Functional Overview

The LMAn module is connected to a UARTEn module. This combination provides a LIN master interface, but can be used also as a buffered UARTEn.

- UART through mode
- UART buffer mode, full-duplex operation
  - 12 byte Tx buffer
  - 12 byte Rx buffer
- LIN master mode
  - automatic checksum generation and check
  - automatic transmission of Break Field (BF), Sync Field (SF), and checksum
  - Scheduler and automatic frame start function

The block diagram shows the environment of the LIN Master Controller.

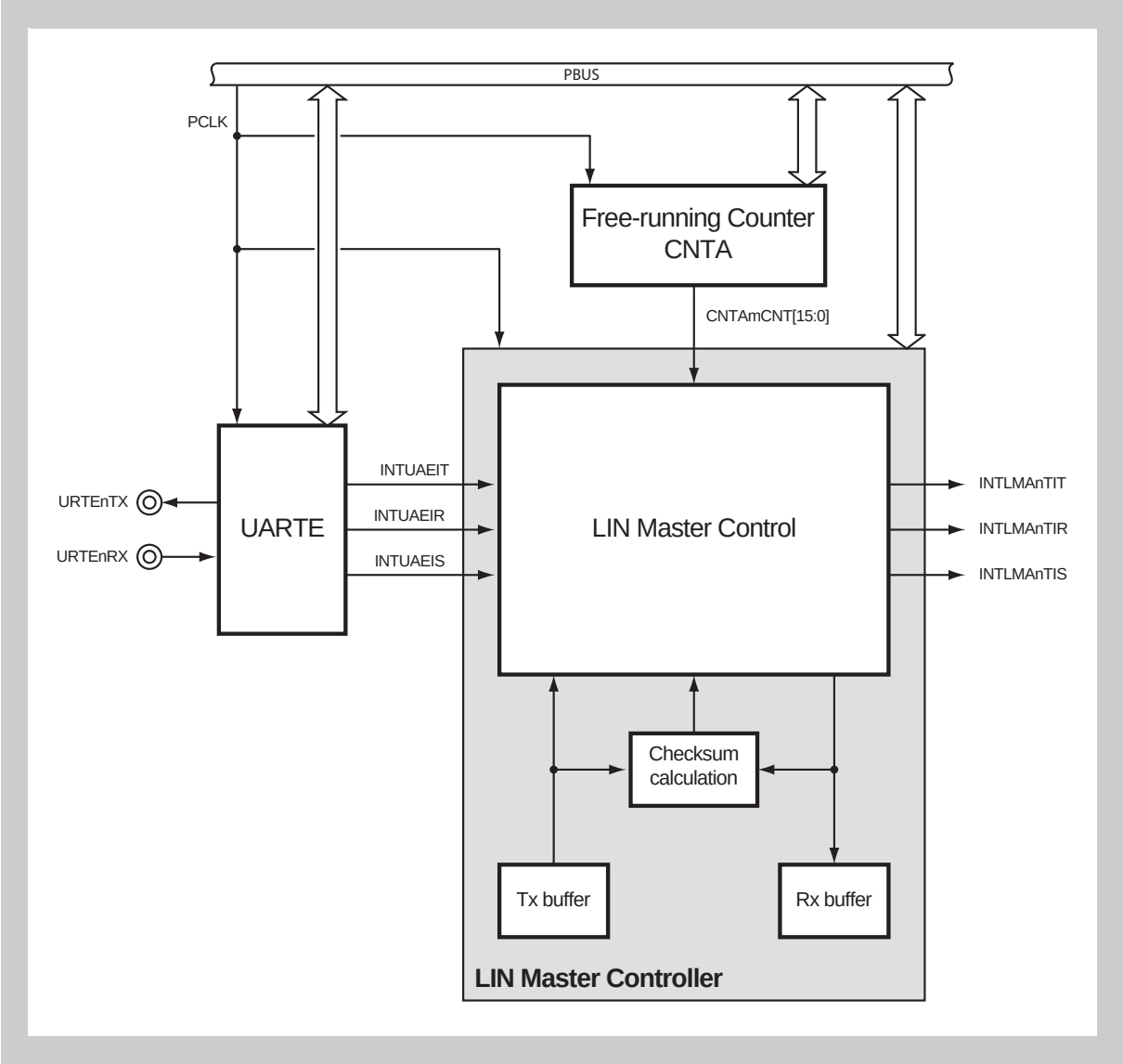



**Figure 17-3 LIN Master Controller environment**

The LIN Master Controller is tightly coupled to the UARTE and utilizes the UARTE as a asynchronous serial interface function with LIN capabilities.

In LIN master mode the UARTE is completely controlled by the LIN Master Controller, all data transfers between the Tx and Rx buffers are managed by the LIN Master Controller.

The UARTE interrupt signals INTUAEIT, INTUAEIR and INTUAEIS are handled by the LIN Master Controller, which generates the interrupt signals INTLMATIT, INTLMATIR and INTLMATIS towards the microcontroller's Interrupt Controller.

For a detailed description of the UARTE refer to the chapter "Asynchronous Serial Interface E (URTE)".

For using the scheduler and the automatic frame start function, a free-running counter is connected to the LIN Master Controller. Refer to the section "LIN Master Scheduler Counter" earlier in this chapter.

## 17.4 Functional Description

The LIN Master can be set up in three different basic modes by setting the LMAnCTLL.LMAnMD[1:0] bits:

- LMAnCTLL.LMAnMD[1:0] = $00_B$: UARTEn through mode
  In this mode the LMAn is bypassed and the connected UARTEn is operated as without LMAn connected.
- LMAnCTLL.LMAnMD[1:0] = $01_B$: UARTEn buffer mode
  In this mode the LMAn operates as a UARTEn Rx and Tx buffer, each providing a storage capacity of 12 byte.
- LMAnCTLL.LMAnMD[1:0] = $1x_B$: LIN master mode
  In this mode the LMAn operates in combination with the UARTEn as a LIN master bus controller, providing Rx and Tx buffers of 12 bytes each in order to handle entire LIN frame transactions without any interaction by the CPU.

### 17.4.1 UART through mode

In UART through mode (LMAnCTLL.LMAnMD[1:0] = $00_B$) the LMAn is bypassed and the UART is used without any LMAn functions.

As this is the default LMAn mode, the UARTEn can be controlled and operated without any LMAn intervention.

**Note** In order to keep power consumption at a minimum, it is recommended to keep LMAnCTLH.LMAnPW = 0.

**Interrupts** All LMAn interrupts request are identical to the UART interrupt requests:

- transmission interrupt request: INTLMAnTIT = INTUAEnTIT
- reception interrupt request: INTLMAnTIR = INTUAEnTIR
- status interrupt request: INTLMAnTIS = INTUAEnTIS

**Data transmission** Data to be transmitted is written to the UARTEn transmit data register URTEnTX.

**Data reception** Received data is read from the UARTEn receive data register URTEnRX.

The UARTE status registers URTEnSTR0 and URTEnSTR1 provide information about data transaction status and error detections.

### 17.4.2 UART buffer mode

In this mode the UARTEn - LMAn combination acts a UARTEn with Rx and Tx buffers of 12 byte size each. This mode is a full-duplex mode, thus receive and transmit transactions are separately controllable and are handled simultaneously.

**(1) Initialization**

**UARTE settings**

The UARTEn must be set up as follows:

- URTEnCTL2
  - URTEnPRS[2:0], URTEnBRS[11:0]: baudrate setting
- URTEnCTL1
  - URTEnSLBM = 1: BF reception during data reception
- URTEnCTL0
  - URTEnPW = 1: UARTEn enabled
  - URTEnCTL0.URTEnTXE = x: transmission enabled/disabled
  - URTEnCTL0.URTEnRXE = x: reception enabled/disabled
  - URTEnCTL0.URTEnSLDC = 0: no data consistency check
  - URTEnSLIT = 0: transmission interrupt request at start of transmission

All other UARTEn settings can be set as required.

**LMAn settings**

The LMAn must be set up as follows:

- LMAnCTLL
  - LMAnMD[1:0] = $01_B$: UART buffer mode
  - LMAnACSE = 0: automatic checksum disabled
  - LMAnSCHE = 0: scheduler disabled
  - LMAnAFE = 0: automatic frame start function disabled
  - LMAnITMK = 0: INTLMAnTIT not masked
  - LMAnIRMK = 0: INTLMAnRIT not masked
- LMAnCTLH
  - LMAnPW = 1: LMAn enabled

**(2) Interrupts**

**INTLMAnTIT**

The transmission interrupt request is generated if the number of data, set up in the Tx buffer and specified by LMAnTCTLL.LMAnTLG[3:0], have been transmitted.

**INTLMAnTIR**

The reception interrupt request is generated if the number of data, specified by LMAnRCTLL.LMAnRLG[3:0], have been stored in the Rx buffer.
In case of continuous reception (LMAnRCTLL.LMAnRLG[3:0] = 0), INTLMAnTIR is generated after storage of the 12th data, i.e. when the Rx buffer is full.

**INTLMAnTIS** The status interrupt request is generated under following conditions:

- UART has detected an error during data reception:
  - parity error: URTEnSTR1.URTEnPE = 1
  - framing error: URTEnSTR1.URTEnFE = 1
  - overrun error: URTEnSTR1.URTEnOVE = 1

**(3) Data transmission**

For data transmission the data to be transmitted have to be written to the Tx buffer (via the LMAnTX01 to LMAnTXAB registers) and the number of bytes to be transmitted has to be specified in LMAnTCTLL.LMAnTLG[3:0] prior starting the transmission by setting the transmit request LMAnTCTLL.LMAnTRQ = 1.

The transmission interrupt request INTLMAnTIT indicates the transmission of the last data byte.

Following values are allowed for the transmit length:

- LMAnTCTLL.LMAnTLG[3:0] = 0: 12 data bytes are transmitted
- LMAnTCTLL.LMAnTLG[3:0] = 1 to 12: 1 to 12 data bytes are transmitted

Setting LMAnTCTLL.LMAnTLG[3:0] > 12 is prohibited.

The following diagram shows the principle transmission process.

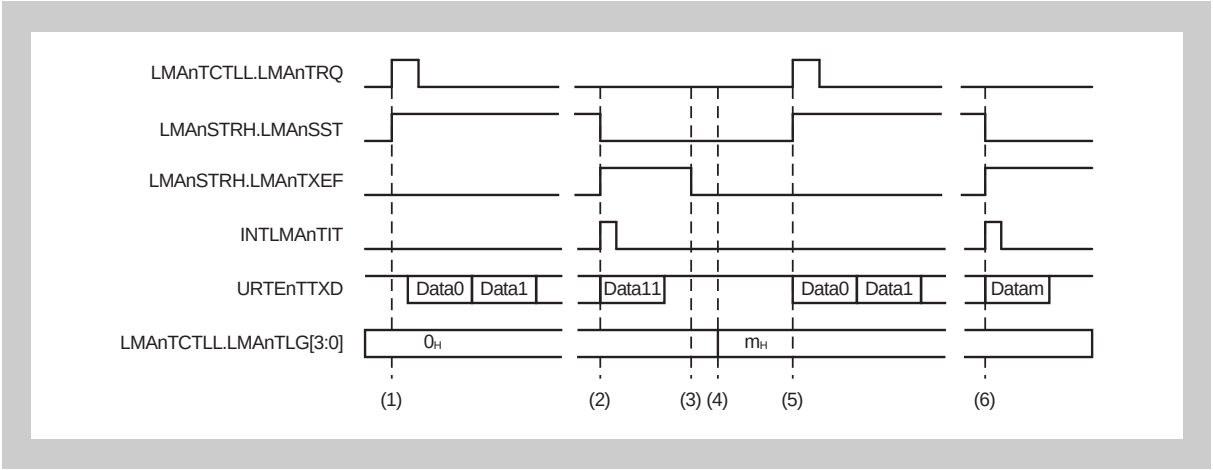



**Figure 17-4 Data transmission in UART buffer mode**

**Precondition** LMAn has been

- set into UART buffer mode (LMAnCTLL.LMAnMD[1:0] = $01_B$)
- enabled (LMAnCTLH.LMAnPW = 1)
- Tx buffer empty flag LMAnSTRH.LMAnTXEF is cleared

**Procedure**

1. Write 12 bytes of data to the Tx buffer LMAnTX01 to LMAnTXAB and set the transmit request LMAnTCTLL.LMAnTRQ = 1. Data Tx starts afterwards.
   The start of transmission is indicated by LMAnSTRH.LMAnSST = 1.
   Since LMAnTCTLL.LMAnTLG[3:0] = 0, 12 byte will be transmitted.
2. Upon Tx start of the last data byte Data11 the transmission interrupt request INTLMAnTIT is asserted and LMAnSTRH.LMAnTXEF = 1 to indicate Tx buffer empty. LMAnSTRH.LMAnSST is cleared to 0.

3. Clear the Tx buffer empty flag by LMAnSTCH.LMAnCLTXEF = 1.
4. The next Tx buffer is prepared to transmit m byte of data by writing m data byte to the Tx buffer and set LMAnTCTLL.LMAnTLG[3:0] = $0C_H$.
5. Afterwards the next transmission is started by LMAnTCTLL.LMAnTRQ = 1.
6. Upon Tx start of the $m^{th}$ data byte Datam the transmission interrupt request INTLMAnTIT is asserted and LMAnSTRH.LMAnTXEF = 1 to indicate Tx buffer empty. LMAnSTRH.LMAnSST is cleared to 0.

**Note** No errors are detected and indicated in case a transmit request is issued, though Tx buffer empty is indicated (LMAnSTRH.LMAnTXEF = 1) or the Tx length is set to incorrect values (LMAnTCTLL.LMAnTLG[3:0] > $0C_H$).

**Tx abort** For stopping an ongoing data transmission, a Tx abort request must be issued by LMAnTCTLH.LMAnTAB = 1. No new data from the Tx buffer is sent to the UARTE and LMAnSTRH.LMAnSST is cleared.
The UARTE completes any ongoing data transmissions. The UARTE Tx completion can be confirmed by URTEnSTR0.URTEnSST = 0.

**Caution** After setting an Tx abort request LMAnTCTLH.LMAnTAB = 1, a transmit interrupt request may occur. Thus mask INTLMAnTIT in the Interrupt Controller before the Tx abort request.

**(4) Data reception**

For data reception the number of bytes to be received has to be specified in LMAnRCTLL.LMAnRLG[3:0] prior starting the reception by setting the receive request LMAnRCTLL.LMAnRRQ = 1.
The reception interrupt request INTLMAnTIR indicates the reception of the last data byte.

Following values are allowed for the transmit length:

- LMAnRCTLL.LMAnRLG[3:0] = 0: continuous Rx mode
  In this mode received data are continuously stored to the Rx buffer without the need to set further receive requests (LMAnRCTLL.LMAnRRQ = 1).Each time 12 data bytes have been stored in the Rx buffer, a reception interrupt request INTLMAnTIR is asserted.
- LMAnRCTLL.LMAnRLG[3:0] = 1 to 12: 1 to 12 data bytes are stored in the Rx buffer

Setting LMAnRCTLL.LMAnTLG[3:0] > 12 is prohibited.

The following diagram shows the principle reception process.

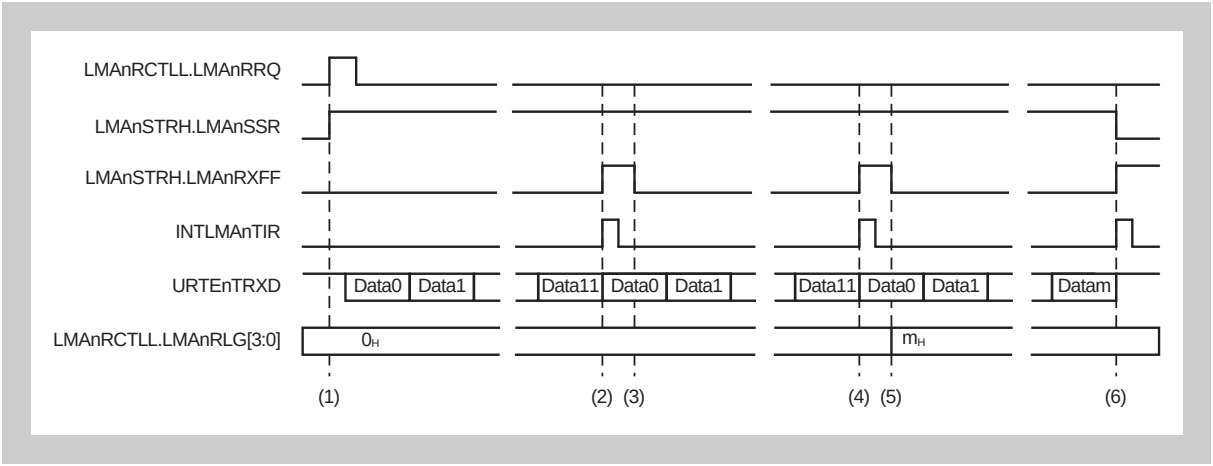



**Figure 17-5 Data reception in UART buffer mode**

**Precondition** LMAn has been

- set into UART buffer mode (LMAnCTLL.LMAnMD[1:0] = $01_B$)
- enabled (LMAnCTLH.LMAnPW = 1)
- Rx buffer full flag LMAnSTRH.LMAnRXFF is cleared

**Procedure**

1. Set the receive request LMAnRCTLL.LMAnRRQ = 1. Data Rx starts afterwards.
   The start of reception is indicated by LMAnSTRH.LMAnSSR = 1.
   Since LMAnRCTLL.LMAnRLG[3:0] = 0, 12 byte will be received in continuous reception mode.
2. Upon Rx of the last data byte 11 the reception interrupt request INTLMAnTIR is asserted and LMAnSTRH.LMAnRXFF = 1 to indicate Rx buffer full.
   Because of continuous reception mode, Data1 is received without a new receive request.
3. The Rx buffer needs to be read via the LMAnRX00 to LMAn RXAB registers and the Rx buffer full flag is cleared by LMAnSTCH.LMAnCLRXFF = 1.
4. After reception of the next 12 data byte, only m data bytes shall be received and reception shall be stopped. Therefore Rx buffer length is set to LMAnRCTLL.LMAnRLG[3:0] = m and the Rx buffer full flag is cleared by LMAnSTCH.LMAnCLRXFF = 1.
5. If the m data bytes have been stored in the Rx buffer, reception is stopped (LMAnSTRH.LMAnSSR = 0).

**Rx abort** For stopping an ongoing data reception, a Rx abort request must be issued by LMAnRCTLH.LMAnRAB = 1. No new data is stored to the TR buffer, and LMAnSTRH.LMAnSSt is cleared.
The UARTE completes any ongoing data reception, but the finally received data is not stored in the Rx buffer. The UARTE Rx completion can be confirmed by URTEnSTR0.URTEnSSR = 0.

**Caution** After setting an Rx abort request LMAnRCTLH.LMAnRAB = 1, a receive interrupt request may occur. Thus mask INTLMAnTIR in the Interrupt Controller before the Rx abort request.

### (5) UARTE Rx errors

Error detections during data reception can be initiated by the UARTE as well as by the LMAn.

**UARTEn errors**

If the UARTE detects a parity, framing or overrun error during data reception, the received data is stored to the Rx buffer and the assigned Rx data error flag LMAnSTRL.LMAnRXBE[11:00] is set. The number of RXBE[11:00] is associated to the data byte in the Rx buffer:

- LMAnSTRL.LMAnRXBE[00] = 1 for LMAnRX01.LMAnRX00B[7:0] error
- ...
- LMAnSTRL.LMAnRXBE[11] = 1 for LMAnRXAB.LMAnRX11B[7:0] error

If the specified number of data bytes (LMAnRCTLL.LMAnRLG[3:0]) have been stored in the Rx buffer, the status interrupt request INTLMAnTIS is asserted and reception stops (LMAnSTRH.LMAnSSR = 0), even if operating in continues reception mode (LMAnRCTLL.LMAnRLG[3:0] = 0).

Note that in this case no receive interrupt request INTLMAnTIR is generated.

The Rx error flags LMAnSTRL.LMAnRXBE[11:00] are cumulative, i.e. each reception error, that is detected until the specified number of bytes are received, set its error flag. Upon the end of reception, indicated by INTLMAnTIS, LMAnSTRL.LMAnRXBE[11:00] can be read in order to identify all Rx buffer data, that have causes error flag settings.

The following diagram shows an example in continuous reception mode (LMAnRCTLL.LMAnRLG[3:0] = 0).

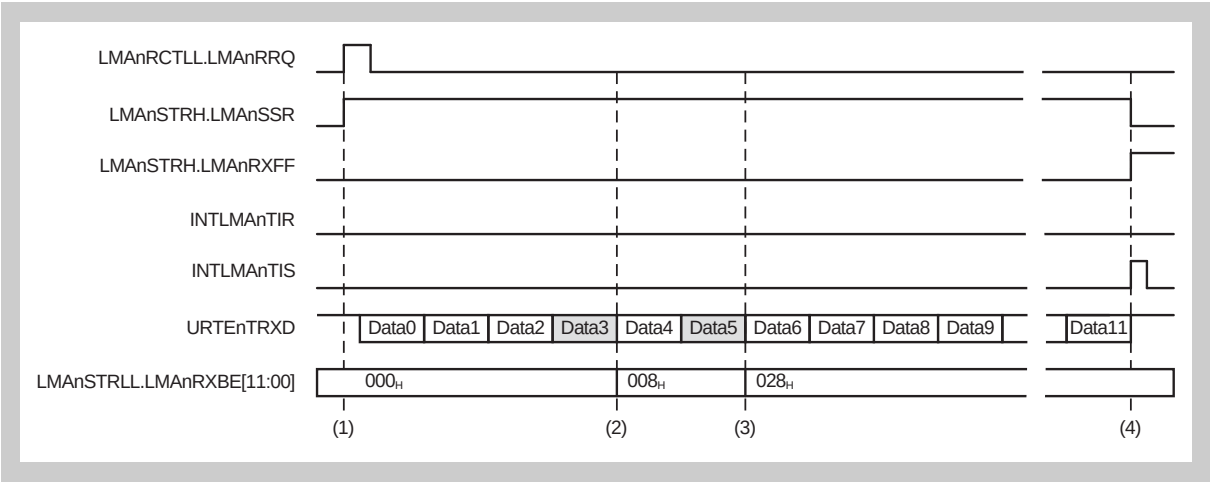



**Figure 17-6 Data reception in UART buffer mode with UART reception errors**

**Precondition**

LMAn has been

- set into UART buffer mode (LMAnCTLL.LMAnMD[1:0] = $01_B$)
- enabled (LMAnCTLH.LMAnPW = 1)
- Rx buffer full flag LMAnSTRH.LMAnRXFF is cleared
- Rx error flags LMAnSTRL.LMAnRXBE[11:00] has been cleared

**Procedure**

1. Rx is started by LMAnRCTLL.LMAnRRQ = 1.
2. During reception of Data3 the UARTE has detected an error. Data3 is stored to the Rx buffer LMAnRX23.LMAnRX03B[7:0] and the associated error flag LMAnSTRL.LMAnRXBE[03] is set, thus LMAnSTRL.LMAnRXBE[11:00] = $008_H$.
3. During reception of Data5 the UARTE has detected another error. Data5 is stored to the Rx buffer LMAnRX45.LMAnRX05B[7:0] and the associated error flag LMAnSTRL.LMAnRXBE[05] is set, thus LMAnSTRL.LMAnRXBE[11:00] = $028_H$.
4. After reception of the last data Data11, the status interrupt INTLMAnTIS is generated (instead of INTLMAnTIR) and the Rx process is stopped (LMAnSTRH.LMAnSSR = 0).

**LMAn Rx buffer overflow error**

If the Rx buffer full flag LMAnSTRH.LMAnRXFF is set while new data has been received,

- the Rx buffer overflow flag LMAnSTRH.LMAnROVE is set
- the status interrupt request INTLMAnTIS is asserted
- the received data is not stored in the Rx buffer
- Rx process is stopped (LMAnSTRH.LMAnSSR = 0).

Since the received data, that is not stored in the Rx buffer, remains in the URATE's receive register URTEnRX, it is possible to read from there. However if a next data is received, this data is lost since the UARTE discards this data, i.e. it does not overwrite the URTEnRX register, but sets its overrun error flag URTEmSTR1.URTEnOVE = 1.

### 17.4.3 LIN master modes

In this mode the LMAn in combination with the UARTE provides a LIN master interface with automatic LIN master Tx and Rx frame transactions, including sending of Break and Sync Fields (BF and SF) to initiate a LIN master frame transaction, automatic checksum functions, scheduler and automatic frame start facilities. The separate 12-byte Tx and Rx buffers allow complete LIN frame transactions without any intervention from the CPU side.

**Principle of operation**

A LIN master frame transactions is initiated by sending the frame header (BF and SF). This is done automatically when the transaction is started.

In either LIN master Tx or Rx mode the PID is be written to the Tx buffer.

In *Tx mode* the data to be transmitted (maximum 8 data bytes) has to written to the Tx buffer and the length of the Tx frame has to be specified in LMAnTCTLL.LMAnTLG[3:0].
If *automatic checksum* is enabled, the checksum is automatically calculated and appended to the Tx data bytes in the Tx buffer. In case automatic checksum is disabled, the CPU needs to calculate the checksum and to write it to the Tx buffer.
After start of the frame transmission by setting the transmit request LMAnTCTLL.LMAnTRQ = 1 transmission starts sending “BF - SF - PID - TxData bytes - Tx Checksum”.
Simultaneously the sent data is received and stored in the Rx buffer for performing the *data consistency check*, which is executed during transmission of the entire LIN frame.
If automatic checksum calculation is enabled, the checksum of the received data is automatically calculated and checked against the received checksum in the Rx buffer. In case of mismatch a checksum error is reported. If automatic checksum calculation is disabled, the checksum over the received data needs to be calculated and checked against the received checksum by the CPU.

In *Rx mode* the frame length to be received (maximum 8 data bytes) has to be specified in LMAnTCTLL.LMAnTLG[3:0].
Frame reception is also started by setting the transmit request LMAnTCTLL.LMAnTRQ = 1. The LIN master starts sending “BF - SF - PID” and then waits to receive the number of specified data bytes and finally the checksum from the LIN slave.
If *automatic checksum* calculation is enabled, the checksum of the received data is automatically calculated and checked against the received checksum in the Rx buffer. In case of mismatch a checksum error is reported. If automatic checksum calculation is disabled, the checksum over the received data needs to be calculated and checked against the received checksum by the CPU.

**Note** In either LIN master Tx or Rx mode the frame transaction control is done via the Tx control register LMAnTCT.

If the *scheduler* is enabled, status interrupts are generated in defined periods to ensure a minimum LIN interframe space (which is the time between two LIN frames). This minimum interframe space may be required by some slaves. For that purpose the length of the LIN frame slot length FRSL, which spans the LIN frame plus the interframe space, is to be specified in the Tx buffer prior starting the LIN frame transaction.

If the scheduler and the *automatic frame start function* is enabled, a new LIN frame transaction is automatically started after the LIN frame slot length FRSL.

**LIN frame header errors**

Two modes are provided to react on errors during the LIN frame header (BF: Break Field, SF: Sync Field) transmission:

- LMAnCTLL.LMAnMD[1:0] = $10_B$: LIN master mode without break in header
  Data transfers are carried on after error detections during header transmission.
- LMAnCTLL.LMAnMD[1:0] = $11_B$: LIN master mode with break in header
  Data transfers are stopped after error detections during header transmission.

**Enhanced checksum**

Automatic checksum calculation can be performed in two ways:

- LMAnTCTLL.LMAnSLEC = 0: only the data bytes are used to calculate the checksum (classic checksum)
- LMAnTCTLL.LMAnSLEC = 1: the PID and the data bytes are used to calculate the checksum (enhanced checksum)

**LIN frame length**

The number of bytes to be transferred, i.e. the length of the LIN frame, needs to be specified in LMAnTCTLL.LMAnTLG[3:0].
The LMAnTCTLL.LMAnTLG[3:0] value includes the number of data bytes (maximum 8), the PID and the checksum. Thus

LMAnTCTLL.LMAnTLG[3:0] = 2 to 10

All other values are prohibited.

**Tx/Rx abort**

For stopping an ongoing data transaction, a Tx abort request must be issued by LMAnTCTLH.LMAnTAB = 1.
No new data from the Tx buffer is sent to the UARTE respectively received from the UARTE and stored in the Rx buffer.LMAnSTRH.LMAnSST is cleared. The UARTE completes any ongoing data transmissions. The UARTE data transmission/reception completion can be confirmed by URTEnSTR0.URTEnSST =URTEnSTR0.URTEnSST = 0.

**Caution**

After setting an Tx abort request LMAnTCTLH.LMAnTAB = 1, a transmit, reception or status interrupt request may occur. Thus mask INTLMAnTIT, INTLMAnTIR, and INTLMAnTIS in the Interrupt Controller before the Tx abort request.

**(1) Initialization**

**CNTAn settings**

If the scheduler is to be used, the scheduler counter must be set up as follows:

- CNTAmCTL
  - CNTAmPW = 1: counter enabled
- CNTAmCFG
  - CNTAmPRS[15:00]: division ratio

**UARTE settings**

The UARTEn must be set up as follows:

- URTEnCTL2
  - URTEnPRS[2:0], URTEnBRS[11:0]: baudrate setting
- URTEnCTL1
  - URTEnSLBM = 0: no BF reception during data reception
  - URTEnBLG[2:0] = 0: BF bit length
  - URTEnCLG = 1: 8-bit data
  - URTEnSLP[1:0] = $00_B$: no parity
  - URTEnTDL = 0: no Tx data inversion
  - URTEnRDL = 0: no Rx data inversion
  - URTEnSLG = 0: 1 stop bit
  - URTEnSLD = 1: LSB first
  - URTEnSLIT = 0: transmission interrupt request at start of transmission
- URTEnCTL0
  - URTEnPW = 1: UARTEn enabled
  - URTEnCTL0.URTEnTXE = 1: transmission enabled
  - URTEnCTL0.URTEnRXE = 1: reception enabled
  - URTEnCTL0.URTEnSLDC = 1: data consistency check enabled

**LMAn settings**

The LMAn must be set up as follows:

- LMAnCTLL
  - LMAnMD[1:0] = $1x_B$: LIN master mode
  - LMAnACSE = x: automatic checksum enabled/disabled
  - LMAnSCHE = x: scheduler enabled/disabled
  - LMAnAFE = x: automatic frame start function enabled/disabled
  - LMAnITMK = x: INTLMAnTIT masked/not masked
  - LMAnIRMK = x: INTLMAnRIT masked/not masked
- LMAnCTLH
  - LMAnPW = 1: LMAn enabled

**(2) Interrupts**

Since both transmission and reception are involved during a LIN master frame transaction, transmit (INTLMAnTIT) and receive (INTLMAnRIT) interrupts are generated in Tx as well as in Rx mode.

**Note** Both interrupt requests can be separately suppressed:

- LMAnCTLL.ITMK = 1: INTLMAnTIT is masked and will not be generated
- LMAnCTLL.IRMK = 1: INTLMAnRIT is masked and will not be generated

**INTLMAnTIT**

- in Tx mode (LMAnTCTLL.LMAnSLRT = 0)
  INTLMAnTIT is generated if the number of data, specified by LMAnTCTLL.LMAnTLG[3:0], have been transmitted.
- in Rx mode (LMAnTCTLL.LMAnSLRT = 1)
  if SF has been transmitted

**INTLMAnTIR**

- in Tx mode (LMAnTCTLL.LMAnSLRT = 0)
  Generation of INTLMAnTIR depends also on the auto-checksum function. If the checksum has been received for checksum control and
  - auto-checksum function is disabled (LMAnCTL.LMAnACSE = 0), INTLMAnTIR is always asserted.
  - auto-checksum function is enabled (LMAnCTL.LMAnACSE = 1), INTLMAnTIR is asserted, if the received checksum matches the automatically calculated checksum. In case of a mismatch a checksum error is indicated (LMAnSTRH.LMAnFCSE = 1) and the status interrupt request INTLMAnTIS is generated instead.
- in Rx mode (LMAnTCTLL.LMAnSLRT = 1)
  INTLMAnTIR is generated if the number of data, specified by LMAnTCTLL.LMAnTLG[3:0], have been stored in the Rx buffer.

**INTLMAnTIS** The status interrupt request is generated under various conditions:

- UARTE has detected a framing error.
- UARTE has detected an overrun error.
- UARTE has detected a data consistency error.
- UARTE has detected a BF transmission error.
- UARTE has detected a SF transmission error.
- LMAn has detected an auto-checksum error.
- LMAn has detected a buffer preparation error.
- LMAn scheduler ready event occurred.

**(3) Data transmission**

The LIN master Tx mode is selected by LMAnCTLL.LMAnMD[1:0] = $1x_B$ and LMAnTCTLL.LMAnSLRT = 0.

For transmission of a LIN master frame the Tx buffer has to be prepared in the following format prior starting the frame transmission:

**Table 17-12 Tx buffer preparation in LIN master Tx mode**

| Tx buffer register | | Tx buffer for 8 data byte | Tx buffer for 5 data byte |
|---|---|---|---|
| LMAnTXAB. | LMAnTX11B[7:0] | FRSLH[a] | FRSLH[a] |
| | LMAnTX10B[7:0] | FRSLL[a] | FRSLL[a] |
| LMAnTX89. | LMAnTX9B[7:0] | Tx checksum[b] | – |
| | LMAnTX8B[7:0] | TxData7 | – |
| LMAnTX67. | LMAnTX7B[7:0] | TxData6 | – |
| | LMAnTX6B[7:0] | TxData5 | Tx checksum[b] |
| LMAnTX45. | LMAnTX5B[7:0] | TxData4 | TxData4 |
| | LMAnTX4B[7:0] | TxData3 | TxData3 |
| LMAnTX23. | LMAnTX3B[7:0] | TxData2 | TxData2 |
| | LMAnTX2B[7:0] | TxData1 | TxData1 |
| LMAnTX01. | LMAnTX1B[7:0] | TxData0 | TxData0 |
| | LMAnTX0B[7:0] | PID | PID |

a) The frame slot length FRSLL/FRSLH is only effective if the scheduler is enabled (LMAnCTLL.LMAnSCHE = 1).

b) The Tx checksum is automatically stored, if automatic checksum is enabled (LMAnCTLL.LMAnACSE = 1). Otherwise the checksum needs to be calculated and stored by software.

After starting the LIN frame transmission the sent data is stored in the Rx buffer for checksum confirmation. When the entire frame has been transmitted, the Rx buffer looks as follows:

**Table 17-13 Rx buffer after LIN frame transmission**

| Rx buffer register | | Rx buffer for 8 data byte | Rx buffer for 5 data byte |
|---|---|---|---|
| LMAnRxAB. | LMAnRx11B[7:0] | – | – |
| | LMAnRx10B[7:0] | – | – |
| LMAnRx89. | LMAnRx9B[7:0] | Rx checksum | – |
| | LMAnRx8B[7:0] | RxData7 | – |
| LMAnRx67. | LMAnRx7B[7:0] | RxData6 | – |
| | LMAnRx6B[7:0] | RxData5 | Rx checksum |
| LMAnRx45. | LMAnRx5B[7:0] | RxData4 | RxData4 |
| | LMAnRx4B[7:0] | RxData3 | RxData3 |
| LMAnRx23. | LMAnRx3B[7:0] | RxData2 | RxData2 |
| | LMAnRx2B[7:0] | RxData1 | RxData1 |
| LMAnRx01. | LMAnRx1B[7:0] | RxData0 | RxData0 |
| | LMAnRx0B[7:0] | PID | PID |

If automatic checksum is enabled (LMAnCTLL.LMAnACSE = 1), the checksum (as selected by LMAnTCTLL.LMAnSLEC) is calculated and compared with the received checksum. Otherwise the checksum needs to be calculated and compared by software.

The following diagram shows the principle LIN frame transmission process with the maximum of 8 data byte.

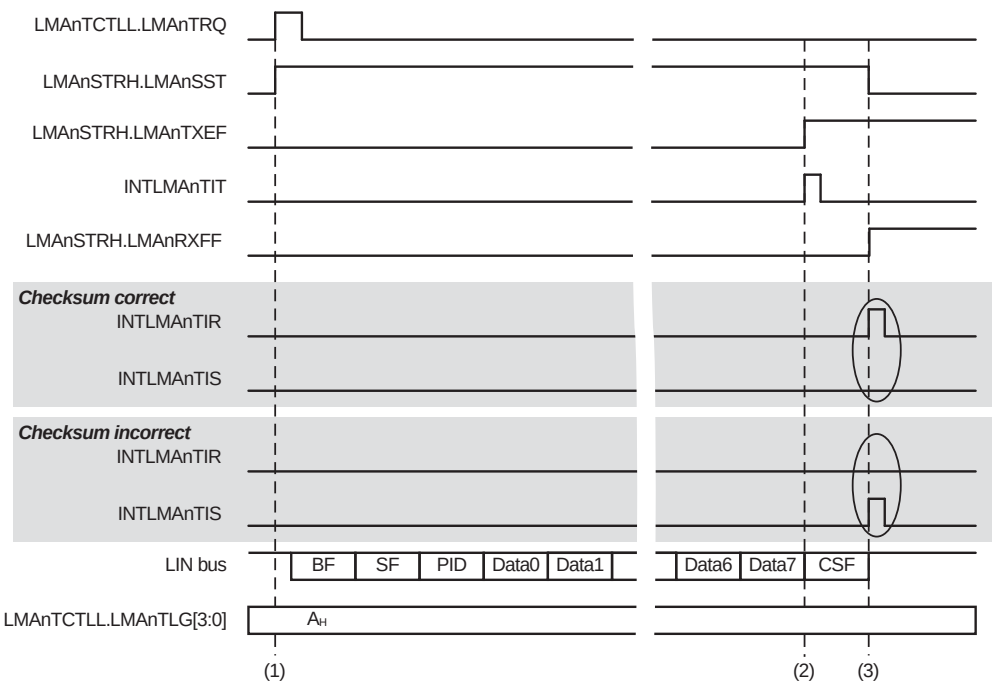



**Figure 17-7 LIN frame transmission in LIN master mode**

**Precondition** LMAn has been

- set into LIN master Tx mode (LMAnCTLL.LMAnMD[1:0] = $1x_B$, LMAnTCTLL.LMAnSLRT = 0)
- neither scheduler and, thus, nor automatic frame start function used (LMAnSCHE and LMAnAFE of LMAnCTLL register both cleared)
- enabled (LMAnCTLH.LMAnPW = 1)
- Tx buffer empty flag LMAnSTRH.LMAnTXEF is cleared
- Rx buffer full flag LMAnSTRH.LMAnRXFF is cleared

**Procedure**

1. Write the PID to Tx buffer LMAnTX01.LMAnTX00B[7:0] and 8 bytes of data to the Tx buffer LMAnTX01.LMAn01B[7:0] to LMAnTX89.LMAn08B[7:0]. If automatic checksum is disabled (LMAnCTLL.LMAnACSE = 0), calculate the checksum CSF (over the bytes including respectively excluding PID, depending on LMAnTCTLL.LMAnSLEC) and store it to LMAnTX89.LMAn09B[7:0]. Otherwise CSF will be calculated and added automatically.
   Set the frame length LMAnTCTLL.LMAnTLG[3:0] = $A_H$ (10 = PID + 8 data bytes + CSF).
   Start frame transmission by LMAnTCTLL.LMAnTRQ = 1, and transmission start is indicated by LMAnSTRH.LMAnSST = 1.
2. Upon start of the Checksum Field (CSF) transmission, Tx buffer empty is indicated by LMAnSTRH.LMAnTXEF = 1 and the transmit interrupt request INTLMAnTIT is asserted.
3. After the checksum field CSF has been transmitted, the Rx buffer full flag LMAnSTRH.LMAnRXFF is set and checksum control is performed,

provided automatic checksum calculation is enabled (LMAnCTLL.LMAnACSE = 1.
If the checksum is correct, the receive interrupt request INTLMAnTIR is asserted.
If the checksum is incorrect, the status INTLMAnTIS instead of the receive interrupt request is asserted.

### (4) Data reception

The LIN master Rx mode is selected by LMAnCTLL.LMAnMD[1:0] = $1x_B$ and LMAnTCTLL.LMAnSLRT = 1.

For data reception in LIN master mode the Tx buffer has to be prepared in the following format prior starting the frame reception:

**Table 17-14 Tx buffer preparation in LIN master Rx mode**

| Tx buffer register | | Tx buffer |
|---|---|---|
| LMAnTXAB. | LMAnTX11B[7:0] | FRSLH[a] |
| | LMAnTX10B[7:0] | FRSLL[a] |
| LMAnTX89. | LMAnTX9B[7:0] | – |
| | LMAnTX8B[7:0] | – |
| LMAnTX67. | LMAnTX7B[7:0] | – |
| | LMAnTX6B[7:0] | – |
| LMAnTX45. | LMAnTX5B[7:0] | – |
| | LMAnTX4B[7:0] | – |
| LMAnTX23. | LMAnTX3B[7:0] | – |
| | LMAnTX2B[7:0] | – |
| LMAnTX01. | LMAnTX1B[7:0] | – |
| | LMAnTX0B[7:0] | PID |

a) The frame slot length FRSLL/FRSLH is only effective if the scheduler is enabled (LMAnCTLL.LMAnSCHE = 1).

After starting the LIN frame reception the LIN master frame header (BF, SF, PID) is transmitted to the slaves and the data, received from the slave afterwards, is stored in the Rx buffer. When the entire frame has been transmitted, the Rx buffer looks as follows:

**Table 17-15 LIN master Rx buffer after LIN frame reception (1/2)**

| Rx buffer register | | Rx buffer for 8 data byte | Rx buffer for 5 data byte |
|---|---|---|---|
| LMAnRxAB. | LMAnRx11B[7:0] | – | – |
| | LMAnRx10B[7:0] | – | – |
| LMAnRx89. | LMAnRx9B[7:0] | Rx checksum | – |
| | LMAnRx8B[7:0] | RxData7 | – |
| LMAnRx67. | LMAnRx7B[7:0] | RxData6 | – |
| | LMAnRx6B[7:0] | RxData5 | Rx checksum |
| LMAnRx45. | LMAnRx5B[7:0] | RxData4 | RxData4 |
| | LMAnRx4B[7:0] | RxData3 | RxData3 |

**Table 17-15 LIN master Rx buffer after LIN frame reception (2/2)**

| Rx buffer register | | Rx buffer for 8 data byte | Rx buffer for 5 data byte |
|---|---|---|---|
| LMAnRx23. | LMAnRx3B[7:0] | RxData2 | RxData2 |
| | LMAnRx2B[7:0] | RxData1 | RxData1 |
| LMAnRx01. | LMAnRx1B[7:0] | RxData0 | RxData0 |
| | LMAnRx0B[7:0] | PID | PID |

If automatic checksum is enabled (LMAnCTLL.LMAnACSE = 1), the checksum (as selected by LMAnTCTLL.LMAnSLEC) is calculated and compared with the received checksum. Otherwise the checksum needs to be calculated and compared by software.

The following diagram shows the principle LIN frame reception process with the maximum of 8 data byte.

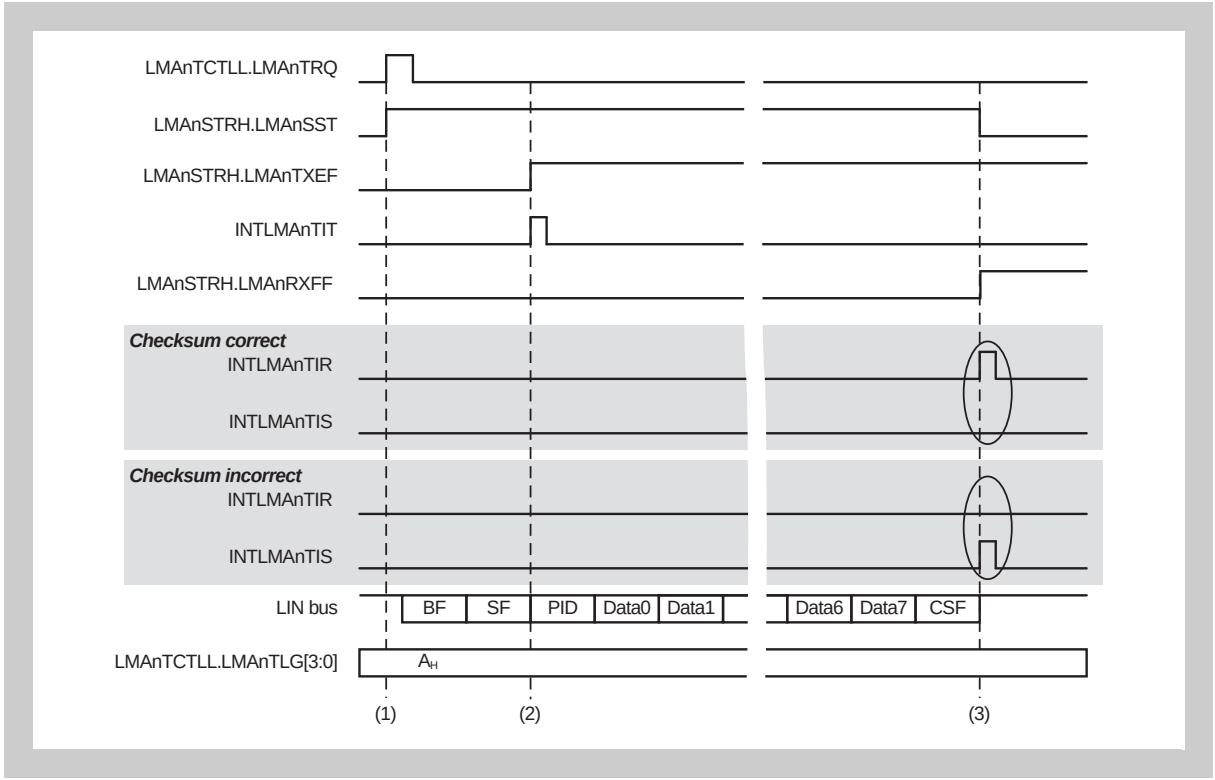



**Figure 17-8 LIN frame reception in LIN master mode**

**Precondition** LMAn has been

- set into LIN master Rx mode (LMAnCTLL.LMAnMD[1:0] = $1x_B$, LMAnTCTLL.LMAnSLRT = 1)
- neither scheduler and, thus, nor automatic frame start function used (LMAnSCHE and LMAnAFE of LMAnCTLL register both cleared)
- enabled (LMAnCTLH.LMAnPW = 1)
- Tx buffer empty flag LMAnSTRH.LMAnTXEF is cleared
- Rx buffer full flag LMAnSTRH.LMAnRXFF is cleared

**Procedure**

1. Write the PID to Tx buffer LMAnTX01.LMAnTX00B[7:0].
   The number of data
   Start frame reception by LMAnTCTLL.LMAnTRQ = 1, and transmission start of the frame header is indicated by LMAnSTRH.LMAnSST = 1.
2. Upon start of the PID transmission, Tx buffer empty is indicated by LMAnSTRH.LMAnTXEF = 1 and the transmit interrupt request INTLMAnTIT is asserted.
3. After the checksum field CSF has been transmitted, the Rx buffer full flag LMAnSTRH.LMAnRXFF is set and checksum control is performed, provided automatic checksum calculation is enabled (LMAnCTLL.LMAnACSE = 1.
   If the checksum is correct, the receive interrupt request INTLMAnTIR is asserted.
   If the checksum is incorrect, the status INTLMAnTIS instead of the receive interrupt request is asserted.

### (5) LIN master mode transaction errors

During LIN master mode transactions the UARTEn as well as the LMAn can detected and indicate various transaction errors.

**Table 17-16 LIN master mode transaction errors**

| Detection module | Error | Indicator | Error cause |
|---|---|---|---|
| UARTE | Framing error | URTEnSTR1.URTEnFE = 1 | No stop bit was detected after reception of the 8th bit of the SF, PID, data, or CSF (checksum) byte. A framing error generates also a data consistency error. |
| | Overrun error | URTEnSTR1.URTEnOVE = 1 | New data received, while the UARTE receive register URTEnRX holds data, that has not been stored in the Rx buffer. |
| | Data consistency error | URTEnSTR1.URTEnDCE = 1 | Data sent during transmission is erroneous.<br>A data consistency error during BF transmission is indicated as a BF transmission error.<br>A data consistency error during SF transmission is indicated as a SF transmission error.<br>Note that any framing error generates also a data consistency error. |
| | BF transmission error | URTEnSTR1.URTEnBSF = 0<br>LMAnSTRH.LMANBFE = 1 | Data consistency occurs during BF Tx |
| | SF transmission error | URTEnSTR1.URTEnDCE = 1<br>LMAnSTRH.LMANSFE = 1 | Data consistency error occurred during SF transmission. |
| LMAn | Auto-checksum error | LMAnSTRH.LMAnFCSE = 1 | Calculated checksum does not match the received checksum after completion of a LIN frame transaction. provided the auto-checksum function is enabled by LMAnCTLL.LMAnACSE = 1. |
| | Rx/Tx buffer preparation error | LMAnSTRH.LMAnPIE = 1 | LIN frame transaction is started (by LMAnTCTLL.TRQ = 1 or in auto frame start mode), while the Tx and Rx buffer are not set up correctly, i.e.<br>• Tx buffer empty (LMAnSTRH.LMAnTXEF = 1)<br>• Rx buffer full (LMAnSTRH.LMAnRXFF = 1)<br>• transmit length incorrect (LMAnTCTLL.LMAnTLG[3:0] = 0, 1, 11 to 15) |

The procedure taking place upon an UARTE error detection depends on the mode the LMAn is operating in and the type of erroneous data.

The main difference between LIN master mode with and without break in header refers to the different behaviour, when a data consistency check error is detected in the LIN frame header (BF/SF) in Rx mode: “with break in header” stops any further transactions, while “without break in header” continuous transactions.

- LIN master mode with break in header

  If a *data consistency check error* was detected in any type of data, further transactions are stopped, the associated Rx buffer error flag LMAnSTRL.LMAnRXBE is set and a status interrupt request INTLMAnTIS is generated.

  In case of *framing/overrun error* detections during transmission of the BF/SF transactions are continued.
  If PID/data/checksum are received with framing/overrun errors in Rx mode, further transactions are stopped, the associated Rx buffer error flag LMAnSTRL.LMAnRXBE is set and a status interrupt request INTLMAnTIS is generated.

- LIN master mode without break in header

  If a *data consistency check error* was detected in PID/data/checksum, or during transmission of BF/SF, further transactions are stopped, the associated Rx buffer error flag LMAnSTRL.LMAnRXBE is set and a status interrupt request INTLMAnTIS is generated. If data inconsistency during BF/SF in Rx mode, transactions continue, the associated Rx buffer error flag LMAnSTRL.LMAnRXBE is set and a status interrupt request INTLMAnTIS is generated.

  In case of *framing/overrun error* detections during transmission of the BF or SF transactions are continued.
  If PID/data/Checksum are received with framing/overrun errors in Rx mode, further transactions are stopped, the associated Rx buffer error flag LMAnSTRL.LMAnRXBE is set and a status interrupt request INTLMAnTIS is generated.

The following table summarizes these procedures.

**Figure 17-9 UARTE errors in LIN master mode**

| Rx/Tx mode | Framing error[a]/overrun error in | | Data consistency check error in | |
|---|---|---|---|---|
| | BF/SF[b] | PID/data/CSF | BF / SF | PID / Data / CSF |
| **LIN master mode with break in header (LMAnCTLL.LMAnMD[1:0] = $11_B$)** | | | | |
| Tx mode (LMAnTCTLL. LMAnSLRT = 0) | Transaction continued | | • Transaction stopped at STOP bit of erroneous field<br>• LMAnSTRL.LMAnRXBE[i] = 1<br>• INTLMAnTIS asserted | |
| Rx mode (LMAnTCTLL. LMAnSLRT = 1) | Transaction continued | • Transaction stopped at STOP bit of erroneous field<br>• LMAnSTRL.LMAnRXBE[i] = 1<br>• INTLMAnTIS asserted | | |
| **LIN master mode without break in header (LMAnCTLL.LMAnMD[1:0] = $10_B$)** | | | | |
| Tx mode (LMAnTCTLL. LMAnSLRT = 0) | Transaction continued | | • Transaction stopped after PID transmitted<br>• LMAnSTRL.LMAnRXBE[i] = 1<br>• INTLMAnTIS asserted after PID transmitted | • Transaction stopped at STOP bit of erroneous field<br>• LMAnSTRL.LMAnRXBE[i] = 1<br>• INTLMAnTIS asserted |
| Rx mode (LMAnTCTLL. LMAnSLRT = 1) | Transaction continued | • Transaction stopped at STOP bit of erroneous field<br>• LMAnSTRL.LMAnRXBE[i] = 1<br>• INTLMAnTIS asserted | • Transaction continued<br>• LMAnSTRL.LMAnRXBE[i] = 1<br>• INTLMAnTIS asserted after CSF reception completed | |

a) A framing error during transmission of any type of data a data consistency error is detected simultaneously (in Tx mode: BF/SF/PID/data/CSF are transmitted, in Rx mode: BF/SF/PID/ are transmitted).

b) Any error detection in BF or SF sets the respective error flag LMAnSTRH.LMAnBFE respectively LMAnSTRH.LMAnSFE.

**Note** During data reception in LIN Master mode, the LMAn cannot distinguish between missing responses from LIN slaves and incomplete responses from LIN slaves.
In both cases, the data reception will not finish and needs to be aborted by software.
There is no indication, how many data bytes have been received up to the abortion.

### 17.4.4 Automatic checksum function

The automatic checksum function allows to automatically generate respectively control checksums.

The automatic checksum function is control by:

- LMAnCTLL.LMAnACSE = 0: automatic checksum function disabled
- LMAnCTLL.LMAnACSE = 1: automatic checksum function enabled

In LMAnCTLL.LMAnACSE = 0: automatic checksum function is enabled, the automatic checksum function performs the following:

**Tx mode** In Tx mode (LMAnTCTLL.LMAnSLRT = 0), the checksum is automatically calculated and appended to the transmit data in the Tx buffer upon LIN frame transaction start.
After transmission completion the checksum the same procedure takes place as in Rx mode (see below).

**Rx mode** After reception completion the checksum is calculated from the received data and automatically compared to the received checksum, stored in the Rx buffer. In case both match, the receive interrupt request INTLMAnTIR is generated. If they don't match, the status interrupt request INTLMAnTIS is generated and the checksum error flag LMAnSTRH.LMAnFCSE is set.

**Checksum format** The data incorporated in the calculation of the checksum in automatic checksum mode can be chosen:

- LMAnTCTLL.LMAnSLEC = 0: classic checksum
  Only the data bytes, stored in the Tx or Rx buffer, are used to calculate the checksum.
- LMAnTCTLL.LMAnSLEC = 1: enhanced checksum
  The data bytes and the PID, stored in the Tx or Rx buffer, are used to calculate the checksum.

### 17.4.5 Scheduler

The scheduler allows to generate status interrupts INTLMAnTIS in defined time distances. INTLMAnTIS can be used to initiate the next LIN master frame transaction.
By this a minimum LIN interframe space (which is the time between two LIN frames) can be ensured. This minimum interframe space may be required by some slaves.

For that purpose the LIN frame slot length FRSL, which spans the LIN frame plus the interframe space, is to be specified in the Tx buffer prior starting the LIN frame transaction.

**Scheduler counter**

The scheduler makes use of a scheduler counter CNTAm. CNTAm is a free-running counter. Its count clock can be selected by the prescaler CNTAmCFG.CNTAmPS[15:0].
A detailed description of the scheduler counter CNTAm is given in the first section of this chapter in “Scheduler Counter A”.

#### (1) Scheduler operation

**Note**

Before any LIN master frame transaction is started with scheduler, the employed scheduler counter CNTAm has to be enabled (CNTAmCTL.CNTAmPWR = 1) and its prescaler has to be set up (CNTAmCFG.CNTAmPS[15:0]). CNTAmCFG.CNTAmPS[15:0] determines the scheduler count clock SCHECLK. The scheduler clocks are counting up the scheduler counters count value CNTAmCNT.

The scheduler needs to be enabled by LMAnCTLL.LMAnSCHE = 1.

Before starting a LIN master frame transaction (LMAnTCTLL.LMAnTRQ = 1) with scheduler function, the length of the 16-bit LIN frame slot length FRSL[15:0] has to be written to the Tx buffer:

- LMAnTXAB.LMAnTX10B[7:0] = FRSLL = FRSL[7:0]
- LMAnTXAB.LMAnTX11B[7:0] = FRSLH = FRSL[15:8]

FRSL[15:0] defines the number of scheduler clocks SCHECLK for the frame slot length.

If the LIN master frame transaction is started, the current value of the scheduler counter CNTAmCNT[15:0] is added to the defined frame slot length FRSL[15:0] and stored to the compare register LMAnCMPL.LMAnCMP[15:0].

The scheduler counter value CNTAmCNT[15:0] is continuously compared with the compare register LMAnCMP[15:0]. If both match, the status interrupt request INTLMAnTIS is generated and the scheduler ready flag LMAnSTRH.LMAnSRF is set.

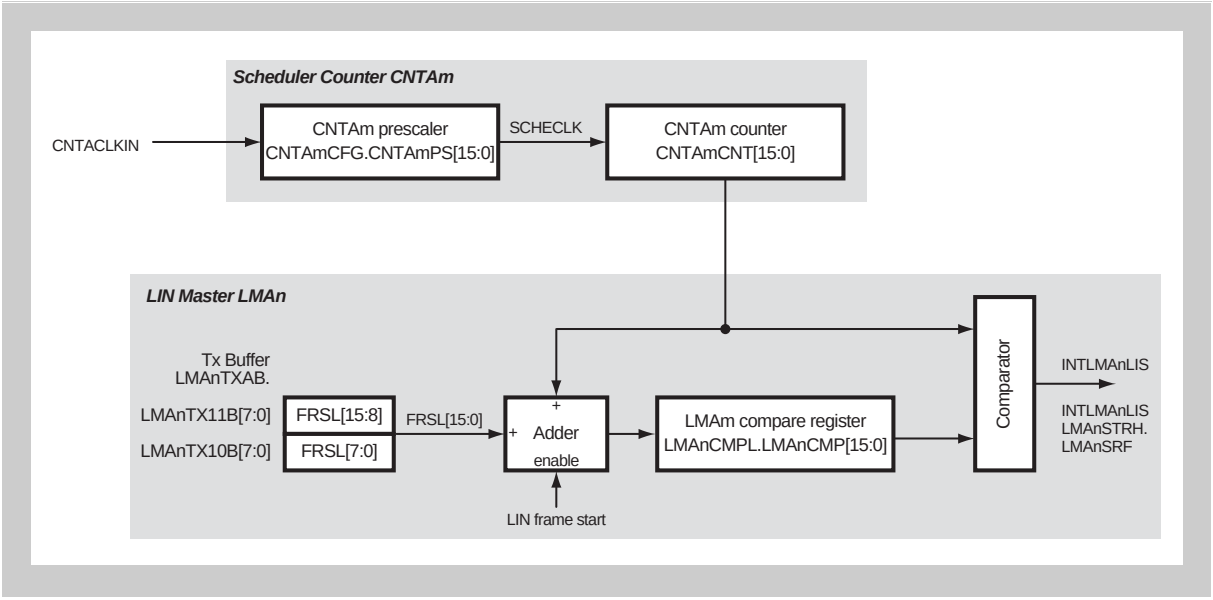



**Figure 17-10 Scheduler functional overview**

**Interrupt handling**

Upon occurrence of the status interrupt INTLMAnTIS transaction of a next LIN master frame can be started immediately, provided the frame slot length FRSL includes also a minimum interframe space. Thus the receive and transmit interrupts may not be necessary to be processed and could be suppressed by masking LMAnCTLL.LMAnITMK = LMAnCTLL.LMAnIRMK = 1.

The diagram below shows the principle timing of a LIN frame transmission with use of the scheduler.

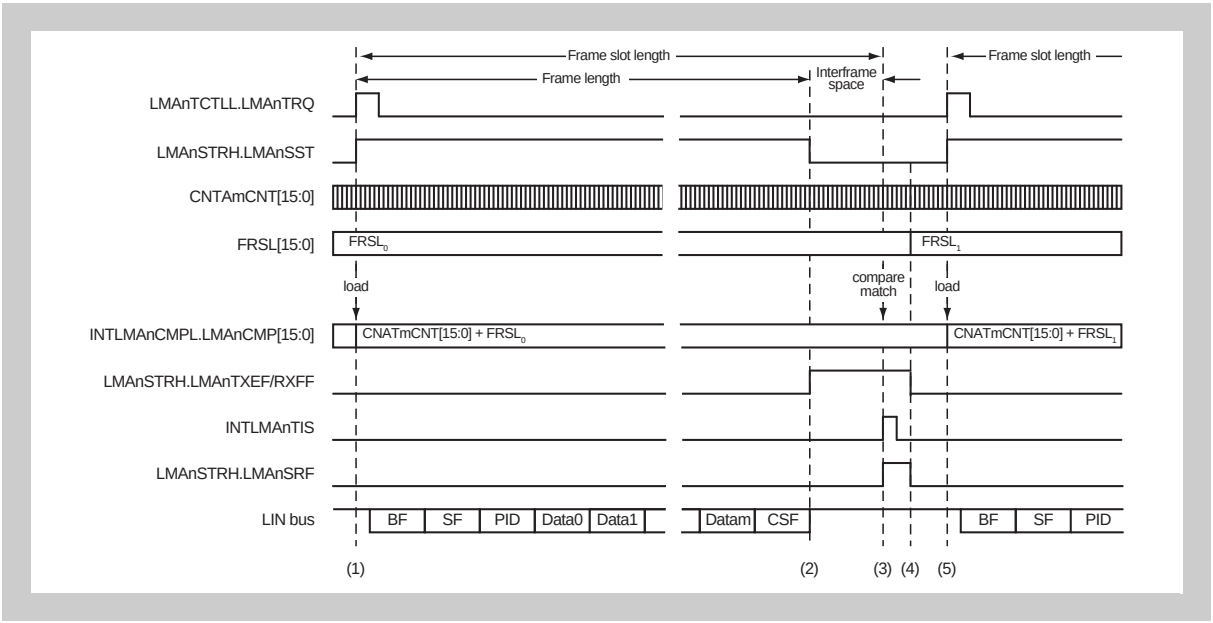



**Figure 17-11 LIN frame transaction with scheduler**

**Precondition**

The scheduler counter is operating and scheduler clock frequency has been set up.
LMAn has been

- set into LIN master mode (LMAnCTLL.LMAnMD[1:0] = $1x_B$)
- scheduler is enabled (LMAnCTLL.LMAnSCHE = 1)
- automatic frame start function is disabled (LMAnCTLL.LMAnAFE = 0)
- enabled (LMAnCTLH.LMAnPW = 1)
- Tx buffer empty flag LMAnSTRH.LMAnTXEF and Rx buffer full flag LMAnSTRH.LMAnRXFF are cleared
- Tx buffer has been set up correctly with frame slot length $FRSL_0$

**Procedure**

1. Start frame transmission by LMAnTCTLL.LMAnTRQ = 1, and transmission start is indicated by LMAnSTRH.LMAnSST = 1.
   Sum of the current schedule counter value CNTAmCNT[15:0] and the frame slot length $FRSL_0$ is stored to LMAnCMP.CMP[15:0].
2. After transmission of the checksum field CSF, i.e. after the frame length, the Tx buffer is indicated as empty (LMAnSTRH.LMAnTXEF = 1) and the Rx buffer as full (LMAnSTRH.LMAnRXFF = 1).
3. Upon match of the scheduler counter CNTAmCNT[15:0] and LMAnCMP.LMAnCMP[15:0] the status interrupt request INTLMAnTIS indicates the end of the frame slot length and the scheduler ready flag LMAnSTRH.LMAnSRF is set.
   If a sufficient interframe space was regarded, when defining the frame slot length $FRSL_0$, the next frame transaction could already be started at this point in time.
4. The buffer status flags and the scheduler ready flag are cleared (LMAnCLTXEF = LMAnCLRXFF = LMAnCLSRF of LMAnSTCH register set to 1).
   The Tx buffer and its control register LMAnTCTLL are prepared for the next frame transaction, with the next frame slot length $FRSL_1$.
5. Next frame transaction is started by LMAnTCTLL.LMAnTRQ = 1, and transmission start is indicated by LMAnSTRH.LMAnSST = 1.
   Sum of the current schedule counter value CNTAmCNT[15:0] and the frame slot length $FRSL_1$ is stored to LMAnCMP.CMP[15:0].

### (2) Scheduler operation with automatic frame start function

The automatic start function, in combination with the scheduler, allows to eliminate unnecessary gaps on the LIN bus by enabling the CPU to make all preparations within the interframe space. The next LIN frame transaction is started automatically after the interframe space, thus maintaining a maximum data transmission performance on the bus.

The automatic frame start function needs to be enabled by LMAnCTLL.LMAnAFE = 1.
Note that the scheduler has to be enabled as well by LMAnCTLL.LMAnSCHE = 1.

In order to start LIN frame transactions with automatic frame start function the first Tx request bit LMAnTCTLL.LMAnFRQ has to be set to 1 in addition to the first Tx request LMAnTCTLL.LMAnTRQ. After LMAnTRQ has returned to 0, it can be set again to 1, although the frame transfer is ongoing (LMAnSTRH.LMAnSST = 1). Thus the next frame transaction is started immediately after the next interframe space has passed.

During the interframe space all preparations (Rx/Tx buffer setup, etc.) have to be finished for the next frame transaction. These preparations can be triggered by the reception interrupt request INTLMAnTIR.

**Caution** The reception interrupt request INTLMAnTIR must be used to initiate frame preparations in Tx as well as in Rx mode.

The diagram below shows the principle timing of a LIN frame transmission with use of the scheduler and automatic frame start function.

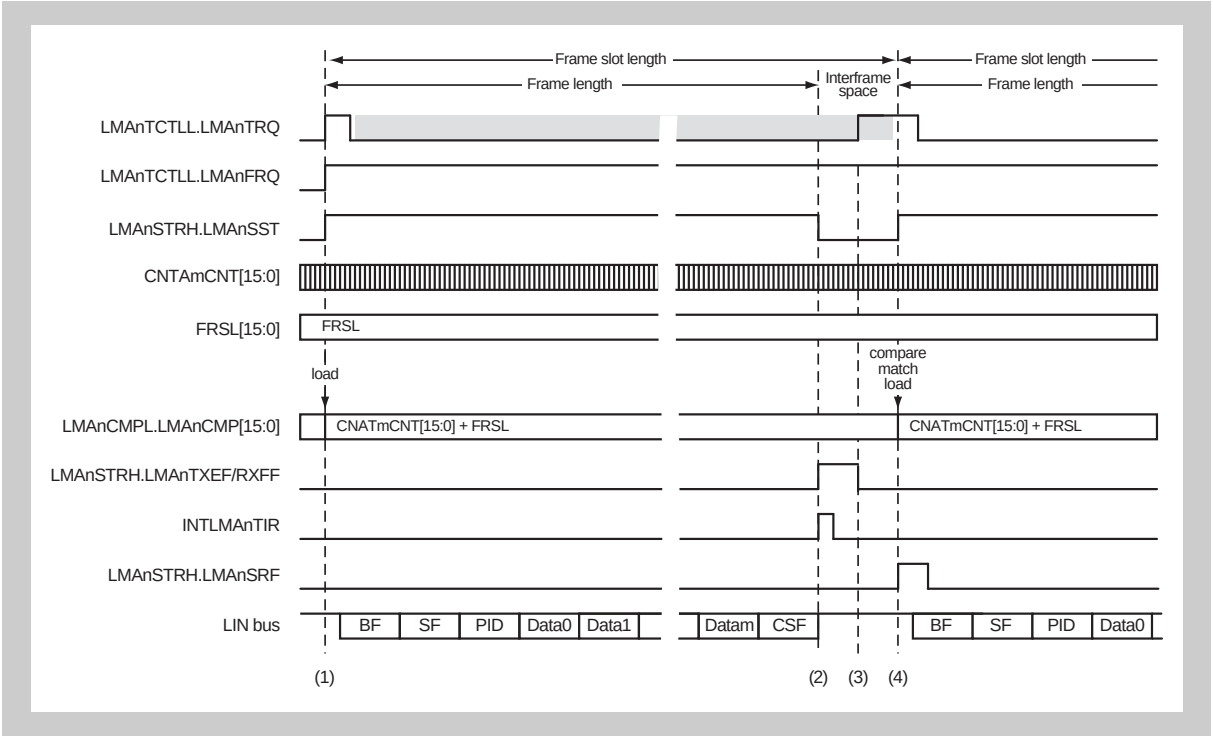



**Figure 17-12 LIN frame transmission with scheduler and automatic frame start function**

**Precondition** The scheduler counter is operating and scheduler clock frequency has been set up.
LMAn has been

- set into LIN master mode (LMAnCTLL.LMAnMD[1:0] = $1x_B$)
- scheduler is enabled (LMAnCTLL.LMAnSCHE = 1)
- automatic frame start function is enabled (LMAnCTLL.LMAnAFE = 1)
- enabled (LMAnCTLH.LMAnPW = 1)
- Tx buffer empty flag LMAnSTRH.LMAnTXEF and Rx buffer full flag LMAnSTRH.LMAnRXFF are cleared
- Tx buffer has been set up correctly with frame slot length FRSL

**Procedure**

1. Start first transmission with automatic frame start function by LMAnTCTLL.LMAnTRQ = 1 and LMAnTCTLL.LMAnFRQ = 1.Transmission start is indicated by LMAnSTRH.LMAnSST = 1.
   Sum of the current schedule counter value CNTAmCNT[15:0] and the frame slot length FRSL is stored to LMAnCMP.CMP[15:0].

After LMAnTCTLL.LMAnTRQ has returned to 0, the next transmission request can be set. At the latest it needs to be set before the interframe space has ended, i.e. in the gray area.
However in the diagram LMAnTRQ is set in the interframe space.

2. After transmission of the checksum field CSF, i.e. after the frame length, the Tx buffer is indicated as empty (LMAnSTRH.LMAnTXEF = 1) and the Rx buffer as full (LMAnSTRH.LMAnRXFF = 1). The reception interrupt request INTLMAnTIR is asserted.
3. All preparations are performed for the next frame transmission.
The buffer status flags and the scheduler ready flag are cleared (LMAnCLTXEF = LMAnCLRXFF = LMAnCLSRF of LMAnSTCH register set to 1).
The next Tx request is set (LMAnTCTLL.LMAnTRQ = 1).
4. Upon match of the scheduler counter CNTAmCNT[15:0] and LMAnCMP.LMAnCMP[15:0], next frame transmission is started.

**Preparation error**

If the next frame is not completely and correctly prepared,

- LMAnTCTLL.LMAnTRQ and LMAnSTRH.LMAnSST are cleared and frame transaction is not started
- LMAnCMP.LMAnCMP[15:0] is loaded with CNATmCNT[15:0] + FRSL
- the preparation incomplete error flag LMAnSTRH.LMAnPIE is set
- the status interrupt request LMAnTIS is asserted

If frame preparation is completed with start of the next frame, i.e. if CNATmCNT[15:0] = LMAnCMP.LMAnCMP[15:0], frame transmission starts. Otherwise the next LMAnTIS interrupt request is asserted to indicate another preparation incomplete error.

## 17.5 LMAn Registers

This section contains a description of all registers of the LIN Master A.

### 17.5.1 LMAn registers overview

The LMAn is controlled and operated by means of the registers, listed in the table below.

One set of 16-bit and one set of 32-bit registers are provided, at which the bits of two 16-bit registers can be accessed via one 32-bit register. Note that the offset addresses for both sets are different.

Throughout this document all descriptions refer to the 16-bit register set. However the described functions of 16-bit registers bits apply also to bits with the same name in their associated 32-bit register.

**Table 17-17 LMAn registers**

| Register function | 16-bit access registers | | 32-bit access registers | |
|---|---|---|---|---|
| | Name | Address | Name | Address |
| **Control and status register:** | | | | |
| Control register | LMAnCTLL | <LMAn_base> + $80_H$ | LMAnCTL0W | <LMAn_base> + $180_H$ |
| | LMAnCTLH | <LMAn_base> + $84_H$ | | |
| Status register | LMAnSTRL | <LMAn_base> + $88_H$ | LMAnSTR0W | <LMAn_base> + $188_H$ |
| | LMAnSTRH | <LMAn_base> + $8C_H$ | | |
| Status clear register | LMAnSTCL | <LMAn_base> + $90_H$ | LMAnSTC0W | <LMAn_base> + $190_H$ |
| | LMAnSTCH | <LMAn_base> + $94_H$ | | |
| Compare register | LMAnCMPL | <LMAn_base> + $98_H$ | LMAnCMP0W | <LMAn_base> + $198_H$ |
| | LMAnCMPH | <LMAn_base> + $9C_H$ | | |
| Tx control register | LMAnTCTLL | <LMAn_base> + $D8_H$ | LMAnTCTL0W | <LMAn_base> + $1D8_H$ |
| | LMAnTCTLH | <LMAn_base> + $DC_H$ | | |
| Rx control register | LMAnRCTLL | <LMAn_base> + $F8_H$ | LMAnRCTL0W | <LMAn_base> + $1F8_H$ |
| | LMAnRCTLH | <LMAn_base> + $FC_H$ | | |
| **Tx buffer register:** | | | | |
| Tx buffer register 01 | LMAnTX01 | <LMAn_base> + $C0_H$ | LMAnTX00W | <LMAn_base> + $1C0_H$ |
| Tx buffer register 23 | LMAnTX23 | <LMAn_base> + $C4_H$ | | |
| Tx buffer register 45 | LMAnTX45 | <LMAn_base> + $C8_H$ | LMAnTX04W | <LMAn_base> + $1C8_H$ |
| Tx buffer register 67 | LMAnTX67 | <LMAn_base> + $CC_H$ | | |
| Tx buffer register 89 | LMAnTX89 | <LMAn_base> + $D0_H$ | LMAnTX08W | <LMAn_base> + $1D0_H$ |
| Tx buffer register AB | LMAnTXAB | <LMAn_base> + $D4_H$ | | |
| **Rx buffer register:** | | | | |
| Rx buffer register 01 | LMAnRX01 | <LMAn_base> + $E0_H$ | LMAnRX00W | <LMAn_base> + $1E0_H$ |
| Rx buffer register 23 | LMAnRX23 | <LMAn_base> + $E4_H$ | | |
| Rx buffer register 45 | LMAnRX45 | <LMAn_base> + $E8_H$ | LMAnRX04W | <LMAn_base> + $1E8_H$ |
| Rx buffer register 67 | LMAnRX67 | <LMAn_base> + $EC_H$ | | |
| Rx buffer register 89 | LMAnRX89 | <LMAn_base> + $F0_H$ | LMAnRX08W | <LMAn_base> + $1F0_H$ |
| Rx buffer register AB | LMAnRXAB | <LMAn_base> + $F4_H$ | | |

| | |
|---|---|
| **<LMAn_base>** | The base addresses <LMAn_base> of the LMAn are defined in the first section of this chapter under the key word “Register addresses”. |
| **Register access** | All registers are accessible in 16-bit width.<br>Writing to non-existing register bits is ignored, reading of these bits return always 0. |

## 17.5.2 LMAn registers details

### (1) LMAnCTLL - LMAn control register L

This register selects the LMAn operation modes.

**Note** This register can only be changed when the LMAn module is disabled (LMAnCTLH.LMAnPW = 0).

**Access** This register can be read/written in 16-bit units.

**Address** <LMAn_base> + $80_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | LMAn MD[1:0] | | LMAn ACSE | LMAn SCHE | LMAn AFE | LMAn ITMK | LMAn IRMK |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-18 LMAnCTLL register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 6 to 5 | LMAn MD[1:0] | LMAn operation mode selection |

| MD[1:0] | Operation mode |
|---|---|
| $00_B$ | UART through mode<br>The LMAn is bypassed and the connected UART is used as a normal UART. |
| $01_B$ | UART buffer mode<br>The connected UART is used UART with buffer. |
| $10_B$ | LIN master mode without break in header<br>The connected UART is used as LIN master. Data transmission is continued even if data consistency errors within the LIN header (BF, SF) are detected. However, in case of an data consistency error in the PID data transmission is stopped. |
| $11_B$ | LIN master mode with break in header<br>The connected UART is used as LIN master.<br>Data transmission is stopped if data consistency errors within the LIN header (BF, SF) are detected. In this case the status interrupt signal INTLMAnTIS is generated. Afterwards a new frame transaction with BF can be started. |

**Table 17-18 LMAnCTLL register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 4 | LMAn ACSE | Automatic checksum calculation enable<br>0: automatic checksum calculation disabled<br>1: automatic checksum calculation enabled<br>In UART through or UART buffer mode (LMAnMD[1:0] = $0x_B$) this bit must be set to 0.<br><br>If automatic checksum calculation is disabled, the<br>• Rx checksum needs to be calculated by software and compared with the received checksum.<br>• Tx checksum need to be calculated by software and to be set to the TX buffer prior starting data transmission.<br><br>If automatic checksum calculation is enabled, the<br>• Rx checksum is automatically calculated from the received data and compared with the received checksum.<br>• Tx checksum is automatically calculated and set to the TX buffer upon start of a data transmission. |
| 3 | LMAn SCHE | Scheduler enable<br>0: scheduler disabled<br>1: scheduler enabled<br>In UART through or UART buffer mode (LMAnMD[1:0] = $0x_B$) this bit must be set to 0.<br>Before the scheduler is enabled, the scheduler counter, that is connected to LMAn, needs to be started. |
| 2 | LMAn AFE | Automatic frame start function enable<br>0: automatic frame start function disabled<br>1: automatic frame start function enabled<br>In UART through or UART buffer mode (LMAnMD[1:0] = $0x_B$) this bit must be set to 0.<br>If the automatic frame start function is disabled, a frame transmission is started by software, when LMAnTCTLL.LMAnTRQ is set to 1.<br>If the automatic frame start function is enabled, a frame transmission is automatically started by the scheduler immediately after the interframe space, if LMAnTCTLL.LMAnTRQ = 1. |
| 1 | LMAn ITMK | Transmission interrupt request (INTLMAnTIT) mask<br>0: INTLMAnTIT not masked (INTLMAnTIT generated)<br>1: INTLMAnTIT masked (INTLMAnTIT is not generated)<br>In UART through or UART buffer mode (LMAnMD[1:0] = $0x_B$) this bit must be set to 0. |
| 0 | LMAn IRMK | Reception interrupt request (INTLMAnTIR) mask<br>0: INTLMAnTIR not masked (INTLMAnTIR generated)<br>1: INTLMAnTIR masked (INTLMAnTIR is not generated)<br>In UART through or UART buffer mode (LMAnMD[1:0] = $0x_B$) this bit must be set to 0. |

### (2) LMAnCTLH - LMAn control register H

This register enables/disables the LMAn operation.

**Access** This register can be read/written in 16-bit units.

**Address** <LMAn_base> + $84_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LMAn PW | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-19 LMAnCTLH register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | LMAn PW | LMAn enable<br>0: LMAn disabled (internal clock stopped)<br>1: LMAn enabled (internal clock operating)<br>When LMAnPW is set to 0, all operations are stopped and LMAn is reset. |

### (3) LMAnSTRL - LMAn status register L

This register provides Rx process status indications.

**Access** This register can be read in 16-bit units.

**Address** <LMAn_base> + $88_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LMAn SSB[2:0] | | | 0 | LMAn RXBE[11:00] | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 17-20 LMAnSTRL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 13 | LMAn SSB[2:0] | Rx buffer mode status flags<br>**SSB[2:0]** / **Rx buffer mode**<br>$0_H$ / Idle state (no data was received)<br>$5_H$ / Data has been received, but the Rx data length has not been reached. An Rx abort is necessary by LMAnRCTLH.LMAnRAB = 1.<br>other / Abnormal operation has occurred. An Rx abort is necessary by LMAnRCTLH.LMAnRAB = 1.<br>LMAnSSB[2:0] can be read for diagnostic purposes when a Rx process stalls. |
| 11 to 0 | LMAn RXBE[11:00] | Rx buffer error flag<br>0: no UART error detections of data in the Rx buffer 11 to 00<br>1: UART has detected an error of data in the Rx buffer 11 to 00<br>The bit number [11:00] corresponds to the Rx buffer number:<br>0: Rx byte 0 (LMAnRX01.LMAnRX00B[7:0] caused an error<br>...<br>11: Rx byte (LMAnRXAB.LMAnRX11B[7:0] caused an error<br><br>Each bit remains at 1 until it is cleared by LMAnSTCL.LMAnCLRRXBE[11:00] = 1. |

### (4) LMAnSTRH - LMAn status register H

This register provides Rx process status indications.

**Access** This register can be read in 16-bit units.

**Address** <LMAn_base> + $8C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LMAn SST | LMAn SSR | LMAn TXEF | LMAn RXFF | LMAn ROVE | LMAn FCSE | LMAn SRF | LMAn PIE | LMAn BFE | LMAn SFE | LMAn SSL[5:0] | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 17-21 LMAnSTRH register contents (1/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | LMAn SST | Tx status flag<br>0: no Tx request issued<br>1: Tx request issued<br><br>LMAnSST is automatically set if a Tx request was set by LMAnTCTLL.LMAnTRQ= 1.<br>If automatic frame start function in LIN mode is used (LMAnCTLL.LMAnAFE = 1), is automatically set upon automatic start of a new frame.<br><br>This flag is not set in UART through mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• Tx process completion (frame transmission completed in LIN master mode) |
| 14 | LMAn SSR | Rx status flag<br>0: no Rx request issued<br>1: Rx request issued<br><br>LMAnSSR is automatically set if a Rx request was set by LMAnRCTLL.LMAnRRQ= 1.<br><br>This flag is not set in UART through or LIN master mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• Rx process completion |
| 13 | LMAn TXEF[a] | Tx buffer empty flag<br>0: data remaining in Tx buffer to be transmitted<br>1: Tx buffer empty: last Tx data transmitted<br><br>This flag is not set in UART through mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLTXEF = 1 |

**Table 17-21 LMAnSTRH register contents (2/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 12 | LMAn RXFF[a] | Rx buffer full flag<br>0: data remaining to be received and to be stored in Rx buffer<br>1: Rx buffer full: last Rx data received<br><br>This flag is not set in UART through mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLRXFF = 1 |
| 11 | LMAn ROVE[a] | Rx buffer overflow flag<br>0: no Rx buffer overflow occurred<br>1: Rx buffer overflow occurred<br><br>When an overflow occurs during data reception, the new data is not stored but discarded.<br><br>This flag is not set in UART through or LIN master mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLROVE = 1 |
| 10 | LMAn FCSE[a] | Checksum error flag<br>0: no checksum error occurred<br>1: checksum error occurred<br><br>LMAnFCSE indicates the result of the checksum control during a LIN frame reception. An error is indicated, if the checksum, calculated from the received data, does not match the received checksum.<br><br>This flag is only effective in LIN master mode with automatic checksum function enabled (LMAnCTLL.LMAnACSE = 1).<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLFCSE = 1 |
| 9 | LMAn SRF[a] | Scheduler ready flag<br>0: no scheduler ready event occurred<br>1: scheduler ready event occurred<br><br>A scheduler ready event occurs, when the value of the compare register LMAnCMPL.LMAnCMP[15:0] matches the value of the scheduler counter. In that case a status interrupt request INTLMAnTIS is also asserted.<br><br>This flag is only effective in LIN master mode with scheduler function enabled (LMAnCTLL.LMAnSCHE = 1), while automatic frame start function remains disabled (LMAnCTLL.LMAnAFE = 0).<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLSRF = 1 |

**Table 17-21 LMAnSTRH register contents (3/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 8 | LMAn PIE[a] | Preparation incomplete error flag<br>0: Rx/Tx buffer preparation correct<br>1: Rx/Tx buffer preparation incorrect<br><br>LMAnPIE is set, if a LIN master frame transfer is started, though the Rx respectively Tx buffer has not been prepared correctly. An incorrect buffer preparation is detected upon any of the following conditions:<br>• LMAnTXEF = 1 (Tx buffer empty)<br>• LMAnSTRH.LMAnRXFF = 1 (Rx buffer full)<br>• LMAnSTRH.LMAnTCTLL.LMAnTLG[3:0] holds incorrect value<br><br>This flag is only effective in LIN master mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLPIE = 1 |
| 7 | LMAn BFE[a] | BF (Break Field) error flag<br>0: BF transmission successful<br>1: BF transmission failed<br><br>LMAnBFE is set, if a framing, overrun or data consistency check error was detected during BF transmission at start of a LIN frame transmission. In case of an error detection the status interrupt request INTLMAnTIS is asserted.<br><br>This flag is only effective in LIN master mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLBFE = 1 |
| 6 | LMAn SFE[a] | SF (Sync Field) error flag<br>0: SF transmission successful<br>1: SF transmission failed<br><br>LMAnSFE is set, if a framing, overrun or data consistency check error was detected during SF transmission at start of a LIN frame transmission. In case of an error detection the status interrupt request INTLMAnTIS is asserted.<br><br>This flag is only effective in LIN master mode.<br><br>It is cleared by<br>• LMAnCTLH.LMAnPW = 0<br>• LMAnSTCH.LMAnCLSFE = 1 |
| 5 to 0 | LMAn SSL[5:0] | LIN master mode status flag<br><br>**LMAnSSL[5:0]** / **LIN master mode status**<br>$0_H$ / Idle state (no operation)<br>$19_H$ / No response from slave. An Tx abort is necessary by LMAnTCTLH.LMAnTAB = 1.<br>other / Abnormal operation has occurred. An Tx abort is necessary by LMAnTCTLH.LMAnTAB = 1.<br><br>LMAnSSL[2:0] can be read for diagnostic purposes when a LIN master transmission process stalls. |

a) Before starting a data transmission process, these flags should be cleared by setting the corresponding status clear bit in LMAnSTC register to 1.

### (5) LMAnSTCL - LMAn status clear register L

This register is used to clear the status and error bits of the LMAn status register L LMAnSTRL.

**Access** This register can be written in 16-bit units.
Reading this register returns an undefined value.

**Address** <LMAn_base> + $90_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | LMAn CLRXBE[11:00] | | | | | | | | | | | |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 17-22 LMAnSTCL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 11 to 0 | LMAn CLRXBE [11:00] | Clear Rx buffer error flags<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnRXBE[11:00] |

### (6) LMAnSTCH - LMAn status clear register H

This register is used to clear the status and error bits of the LMAn status register L LMAnSTRH.

**Access** This register can be written in 16-bit units.
Reading this register returns an undefined value.

**Address** <LMAn_base> + $94_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | LMAnCLTXEF | LMAnCLRXFF | LMAnCLROVE | LMAnCLFCSE | LMAnCLSRF | LMAnCLPIE | LMAnCLBFE | LMAnCLSFE | 0 | 0 | 0 | 0 | 0 | 0 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 17-23 LMAnSTCH register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 13 | LMAn CL TXEF | Tx buffer empty flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnTXEF |
| 12 | LMAn CL RXFF | Rx buffer full flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnRXFF |
| 11 | LMAn CL ROVE | Rx buffer overflow flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnROVE |
| 10 | LMAn CL FCSE | Checksum error flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnFCSE |
| 9 | LMAn CL SRF | Scheduler ready flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnSRF |
| 8 | LMAn CL PIE | Preparation incomplete error flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnPIE |
| 7 | LMAn CL BFE | BF (Break Field) error flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnBFE |
| 6 | LMAn CL SFE | SF (Sync Field) error flag<br>0: writing 0 is ignored<br>1: writing 1 clears LMAnSFE |

### (7) LMAnCMPL - LMAn compare register L

This register holds the lower 16 bit of scheduler comparison value.

**Access** This register can be read in 16-bit units.

**Address** <LMAn_base> + $98_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LMAnCMP[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 17-24 LMAnCMPL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | LMAn CMP[15:00] | Current scheduler comparison value<br>LMAnCMP[15:00] is loaded with the sum of the current free-running counter value CNATmCNT[15:0] and the frame slot length FRSL upon start of a LIN master frame. |

### (8) LMAnCMPH - LMAn compare register H

This register holds the upper 16 bit of scheduler comparison value.
These bits are always $0000_H$.

**Access** This register can be read in 16-bit units.

**Address** <LMAn_base> + $9C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LMAnCMP[31:16] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 17-25 LMAnCMPH register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | LMAn CMP[31:16] | Current scheduler comparison value<br>LMAnCMP[31:16] are always $0000_H$. |

### (9) LMAnTCTLL - LMAn Tx control register L

This register controls the LMAn Tx buffer.

**Access** This register can be read/written in 16-bit units.

**Address** <LMAn_base> + $D8_H$

**Initial Value** $0000_H$. This register is initialized by any reset and by LMAnCTLH.LMAnPW = 0.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | LMAn SLEC | LMAn SLRT | LMAn FRQ | LMAn TRQ | LMAn TLG[3:0] | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-26 LMAnTCTLL register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | LMAn SLEC | Enhanced checksum control<br>0: classic checksum<br>Only the data bytes are used for checksum calculation.<br>1: enhanced checksum<br>Checksum is calculated from the data and the PID.<br>LMAnSLEC is only effective, if automatic checksum calculation is enabled (LMAnCTLL.LMAnACSE = 1). |
| 6 | LMAn SLRT | LIN master mode operation control<br>0: Tx mode<br>1: Rx mode<br>LMAnSLRT is only effective in LIN master mode. |
| 5 | LMAn FRQ | First Tx request control<br>0: LIN frame transaction starts with the scheduler ready event<br>1: LIN frame transaction starts immediately by requesting transmission (LMAnTRQ = 1), in case no Tx operation has been requested (LMAnSTRH.LMAnSST = 0).<br>In case LMAnFRQ is set to 1 while LMAnSTRH.LMAnSST = 1, LIN frame transaction starts with the next scheduler event, and thus behaves as with LMAnFRQ = 0.<br>Set LMAnFRQ = 1 together with LMAnTRQ = 1.<br><br>LMAnFRQ is only effective in LIN master mode with using the scheduler (LMAnCTLL/LMAnSCHE = 1) and auto frame start enabled (LMAnCTLL.LMAnAFE = 1). In all other modes, LMAnFRQ shall be set to 0. |
| 4 | LMAn TRQ | Tx request control<br>0: Tx operation has started or has not been requested.<br>1: Tx operation request<br>In LIN master mode LMAnTRQ = 1 triggers LIN frame transaction also in LIN master Rx mode (LMAnSLRT = 1.<br><br>After setting LMAnTRQ = 1, LMAnTRQ returns automatically to 0 when transmission has started.<br><br>Writing 0 to LMAnTRQ has no effect.<br><br>In UART through (LMAnMD[1:0] = $00_B$) this bit must be set to 0.<br><br>**Caution:**<br>It is prohibited to set LMAnTRQ = 1 during a pending transmission request (LMAnSTRH.LMAnSST = 1), except LMAn is operating in LIN master mode with scheduler and auto frame start (LMAnCTLL register bits LMAnSCHE = LMAnAFE = 1). |

**Table 17-26 LMAnTCTLL register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>3 to 0</td><td>LMAn TLG[3:0]</td><td>Tx buffer length specification<br>LMAnTLG[3:0] are only effective in UART buffer and LIN master mode.<br>• in UART buffer mode
<table>
<tr><th>LMAnTLG[3:0]</th><th>Tx buffer length</th></tr>
<tr><td>0</td><td>12 byte transmission</td></tr>
<tr><td>1 to 12</td><td>1 to 12 byte transmission in UART buffer mode</td></tr>
<tr><td>13 to 15</td><td>prohibited</td></tr>
</table>
• in LIN master mode
<table>
<tr><th>LMAnTLG[3:0]</th><th>Tx buffer length</th></tr>
<tr><td>0 to 1</td><td>prohibited</td></tr>
<tr><td>2 to 10</td><td>2 to 10 byte transmission in LIN master mode</td></tr>
<tr><td>11 to 15</td><td>prohibited</td></tr>
</table>
The value set to LMAnTLG[3:0] includes the PID and checksum bytes, thus a maximum of 8 data byte can be transmitted.<br>If any of the prohibited values are set to LMAnTLG[3:0], a preparation incomplete error is detected and indicated by LMAnSTRH.LMAnPIE = 1.</td></tr>
</table>

### (10) LMAnTCTLH - LMAn Tx control register H

This register controls the Tx abort process.

**Access** This register can be read/written in 16-bit units.

**Address** <LMAn_base> + $DC_H$

**Initial Value** $0000_H$. This register is initialized by any reset and by LMAnCTLH.LMAnPW = 0.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | LMAn TAB |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-27 LMAnTCTLH register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | LMAn TAB | Tx abort request<br>0: reading returns always 0, writing 0 has no effect<br>1: writing 1 requests Tx abort<br>If LMAnTAB is set to 1, Tx operation is stopped accordingly and the Tx status flag LMAnSTRH.LMAnSST is cleared.<br><br>This bit has no effect in UART through mode.<br><br>This bit is also effective during Rx process in LIN master mode. |

**Note** If Tx is aborted by LMAnTAB = 1, the LMAn stops to send any further data to the UART. However ongoing transmissions are completed by the UART. The complete stop of transmissions can be confirmed by URTEnSTR0.URTEnSST = 0.

### (11) LMAnRCTLL - LMAn Rx control register L

This register controls the LMAn Rx buffer.

**Access** This register can be read/written in 16-bit units.

**Address** <LMAn_base> + $F8_H$

**Initial Value** $0000_H$. This register is initialized by any reset and by LMAnCTLH.LMAnPW = 0.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | LMAn RRQ | LMAn RLG[3:0] | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-28 LMAnRCTLL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 4 | LMAn RRQ | Rx request control<br>0: Rx operation has started or has not been requested.<br>1: Rx operation request<br>After setting LMAnRRQ = 1, LMAnRRQ returns automatically to 0 when reception data storage to the Rx buffer has started.<br><br>Writing 0 to LMAnRRQ has no effect.<br><br>This bit is only effective in UART buffer mode. In all other modes this bit must not be set to 1. |
| 3 to 0 | LMAn RLG[3:0] | Rx buffer length specification<br>LMAnRLG[3:0] are only effective in UART buffer mode.<br>**LMAnRLG[3:0] / Rx buffer length**<br>0 / continuous data reception<br>1 to 12 / 1 to 12 byte reception to Rx buffer<br>13 to 15 / prohibited<br>In continuous data reception (LMAnRLG[3:0] = 0) received data is stored to the Rx buffer continuously. After each 12 byte storage a reception interrupt INTLMAnTIR or status interrupt INTLMAnTIS request is generated. To stop continuous storage set LMAnRLG[3:0] to 1 to 12. |

### (12) LMAnRCTLH - LMAn Rx control register H

This register controls the Rx abort process.

**Access** This register can be read/written in 16-bit units.

**Address** <LMAn_base> + $FC_H$

**Initial Value** $0000_H$. This register is initialized by any reset and by LMAnCTLH.LMAnPW = 0.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | LMAn RAB |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 17-29 LMAnRCTH register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | LMAn RAB | Rx abort request<br>0: reading returns always 0, writing 0 has no effect<br>1: writing 1 requests Rx abort<br>If LMAnRAB is set to 1, Rx operation is stopped accordingly and the Rx status flag LMAnSTRH.LMAnSSR is cleared.<br><br>This bit is only effective in UART buffer mode. In all other modes this bit must not be set to 1.<br><br>This bit is also effective during Rx process in LIN master mode. |

**Note** If Rx is aborted by LMAnRAB = 1, the LMAn stops to store any data in the Rx buffer. However ongoing reception by the UART will be completed.
The complete stop of reception can be confirmed by URTEnSTR0.URTEnSSR = 0.

# Chapter 18 CAN Controller (FCN)

The microcontroller features on-chip CAN (Controller Area Network) controllers that complies with the CAN protocol as standardized in ISO 11898.

This chapter contains a generic description of the CAN Controller (FCN).

The first section describes all V850E2/Fx4-G specific properties, such as instances, register base addresses, input/output signal names, etc. The subsequent sections describe the features that apply to all implementations.

## 18.1 V850E2/Fx4-G FCN Features

**Instances**

This microcontroller has following number of instances of the CAN Controller.

**Table 18-1 Instances of FCN**

| CAN Controller | V850E2/FF4-G (μPD70F4177) V850E2/FG4-G (μPD70F4179) | V850E2/FF4-G (μPD70F4178) V850E2/FG4-G (μPD70F4180) |
|---|---|---|
| Instance | 3 | 6 |
| Name | FCN0 to FCN2 | FCN0 to FCN5 |

**Instances index n**

Throughout this chapter, the instance of a CAN Controller is identified by the index “n” (n = 0 to 5), for example, FCNnGMCLCTL for the FCNn control register.

**Message buffers index m**

Throughout this chapter, the FCN message buffer registers are identified by “m” (for m refer to the table below), for example FCNnMmDAT4B for FCN instance n, message data byte 4 of message buffer register m. The number of message buffers for each instance of FCN is given in the following table:

**Table 18-2 Message buffers of FCNn**

| FCNn instance | Number m of message buffers |
|---|---|
| FCN0 | 32 |
| FCN1 | 32 |
| FCN2 | 32 |
| FCN3 | 32 |
| FCN4 | 32 |
| FCN5 | 32 |

**Register addresses** All CAN Controller register addresses are given as address offsets to the individual base address <FCNn_base>.
The <FCNn_base> address of each FCNn is given in the following table:

**Table 18-3 Register base addresses <FCNn_base>**

| FCNn instance | <FCNn_base> address |
|---|---|
| FCN0 | FF48 0000$_H$ |
| FCN1 | FF4A 0000$_H$ |
| FCN2 | FF4C 0000$_H$ |
| FCN3 | FF4E 0000$_H$ |
| FCN4 | FF50 0000$_H$ |
| FCN5 | FF52 0000$_H$ |

**Clock supply** All CAN Controllers provide one clock input:

**Table 18-4 FCNn clock supply**

| FCNn instance | FCNn clock | Connected to |
|---|---|---|
| FCN0 | PCLK (CANCLK $f_{CAN}$) | Clock Controller CKSCLK_033 |
| FCN1 | PCLK (CANCLK $f_{CAN}$) | Clock Controller CKSCLK_033 |
| FCN2 | PCLK (CANCLK $f_{CAN}$) | Clock Controller CKSCLK_033 |
| FCN3 | PCLK (CANCLK $f_{CAN}$) | Clock Controller CKSCLK_033 |
| FCN4 | PCLK (CANCLK $f_{CAN}$) | Clock Controller CKSCLK_033 |
| FCN5 | PCLK (CANCLK $f_{CAN}$) | Clock Controller CKSCLK_033 |

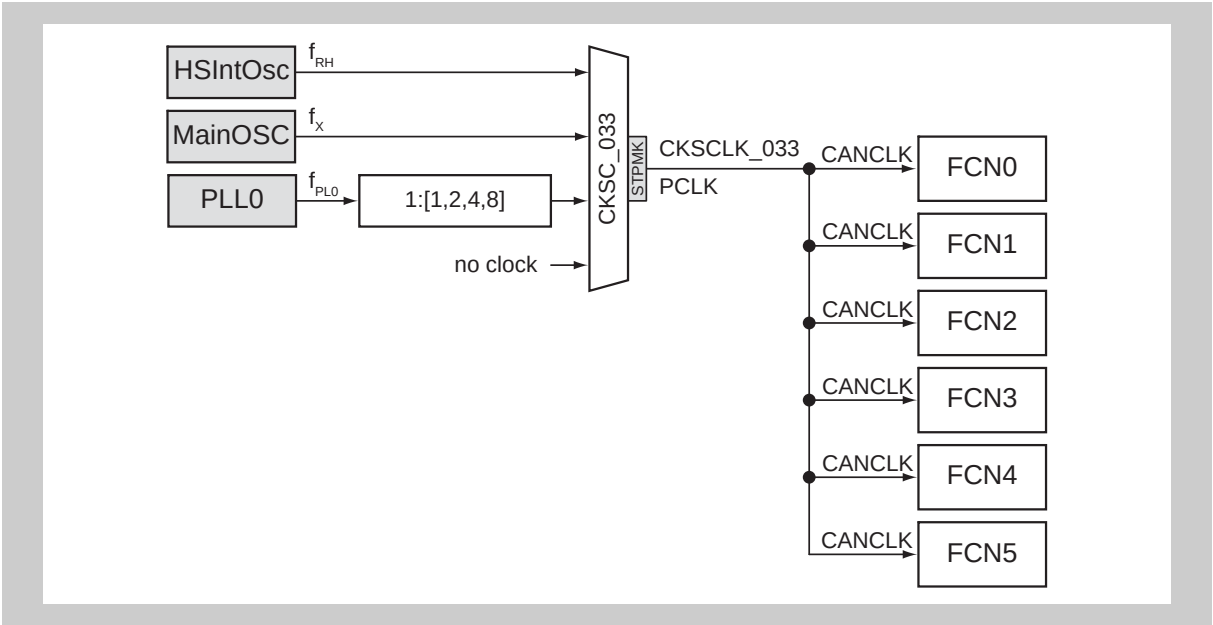



**Figure 18-1 FCN clock supply**

**Interrupts and DMA**

The CAN Controllers can generate the following interrupt and DMA requests:

**Table 18-5 FCNn interrupt and DMA requests (1/2)**

| FCNn signals | Function | Connected to |
|---|---|---|
| **FCN0:** | | |
| INTC0ERR | Error indication | Interrupt Controller INTFCN0ERR |
| INTC0REC | Reception completion | Interrupt Controller INTFCN0REC[a] |
| INTC0TRX | Transmission completion | Interrupt Controller INTFCN0TRX DMA Controller trigger 28 |
| INTC0WUP | Sleep wake-up / transmission abortion | Interrupt Controller INTFCNWUP[b] |
| **FCN1:** | | |
| INTC1ERR | Error indication | Interrupt Controller INTFCN1ERR |
| INTC1REC | Reception completion | Interrupt Controller INTFCN1REC[a] |
| INTC1TRX | Transmission completion | Interrupt Controller INTFCN1TRX DMA Controller trigger 29 |
| INTC1WUP | Sleep wake-up / transmission abortion | Interrupt Controller INTFCNWUP[b] |
| **FCN2:** | | |
| INTC2ERR | Error indication | Interrupt Controller INTFCN2ERR |
| INTC2REC | Reception completion | Interrupt Controller INTFCN2REC[a] |
| INTC2TRX | Transmission completion | Interrupt Controller INTFCN2TRX DMA Controller trigger 30 |
| INTC2WUP | Sleep wake-up / transmission abortion | Interrupt Controller INTFCNWUP[b] |
| **FCN3:** | | |
| INTC3ERR | Error indication | Interrupt Controller INTFCN3ERR |
| INTC3REC | Reception completion | Interrupt Controller INTFCN3REC[a] |
| INTC3TRX | Transmission completion | Interrupt Controller INTFCN3TRX DMA Controller trigger 31 |
| INTC3WUP | Sleep wake-up / transmission abortion | Interrupt Controller INTFCNWUP[b] |
| **FCN4:** | | |
| INTC4ERR | Error indication | Interrupt Controller INTFCN4ERR |
| INTC4REC | Reception completion | Interrupt Controller INTFCN4REC[a] |
| INTC4TRX | Transmission completion | Interrupt Controller INTFCN4TRX DMA Controller trigger 32 |
| INTC4WUP | Sleep wake-up / transmission abortion | Interrupt Controller INTFCNWUP[b] |

**Table 18-5 FCNn interrupt and DMA requests (2/2)**

| FCNn signals | Function | Connected to |
|---|---|---|
| **FCN5:** | | |
| INTC5ERR | Error indication | Interrupt Controller INTFCN5ERR |
| INTC5REC | Reception completion | Interrupt Controller INTFCN5REC[a] |
| INTC5TRX | Transmission completion | Interrupt Controller INTFCN5TRX DMA Controller trigger 33 |
| INTC5WUP | Sleep wake-up / transmission abortion | Interrupt Controller INTFCNWUP[b] |

a) These interrupts can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

b) The Interrupt Controller input INTFCNWUP is a combination of the wake-up interrupts of the CAN Controllers FCN0 to FCN5. Thus any of these wake-up interrupts asserts the interrupt request INTFCNWUP.

**FCN H/W reset**

The CAN Controllers and their registers are initialized by the following reset signal:

**Table 18-6 FCNn reset signal**

| FCNn | Reset signal |
|---|---|
| FCNn | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

**I/O signals** The I/O signals of the CAN Controllers are listed in the table below.

**Table 18-7 CAN Controllers I/O signals**

| FCNn signals | Function | Connected to |
|---|---|---|
| **FCN0:** | | |
| CRXD0 | CAN bus receive input | Port FCN0RX[a] or FCN1RX[b] |
| CTXD0 | CAN bus transmit output | Port FCN0TX |
| **FCN1:** | | |
| CRXD1 | CAN bus receive input | Port FCN1RX[a] |
| CTXD1 | CAN bus transmit output | Port FCN1TX |
| **FCN2:** | | |
| CRXD2 | CAN bus receive input | Port FCN2RX[a] |
| CTXD2 | CAN bus transmit output | Port FCN2TX |
| **FCN3:** | | |
| CRXD3 | CAN bus receive input | Port FCN3RX[a] |
| CTXD3 | CAN bus transmit output | Port FCN3TX |
| **FCN4:** | | |
| CRXD4 | CAN bus receive input | Port FCN4RX[a] |
| CTXD4 | CAN bus transmit output | Port FCN4TX |
| **FCN5:** | | |
| CRXD5 | CAN bus receive input | Port FCN5RX[a] |
| CTXD5 | CAN bus transmit output | Port FCN5TX |

a) These signals can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

b) Refer to *17.7 "FCN0 and FCN1 connection" on page 1* for details.

**Time stamp** Following FCNn time stamp output signals can be internally connected to a capture input of the Timer Array Units B.

**Table 18-8 FCNn time stamp signals**

| FCNn signals | Function | Connected to |
|---|---|---|
| **FCN0:** | | |
| TSOUT | CAN time stamp output | TAUB0 TAUB0TTIN0, TAUB0TTIN10 |
| **FCN1:** | | |
| TSOUT | CAN time stamp output | TAUB0 TAUB0TTIN1, TAUB0TTIN11 |
| **FCN2:** | | |
| TSOUT | CAN time stamp output | TAUB0 TAUB0TTIN2, TAUB0TTIN12 |
| **FCN3:** | | |
| TSOUT | CAN time stamp output | TAUB0 TAUB0TTIN3, TAUB0TTIN13 |
| **FCN4:** | | |
| TSOUT | CAN time stamp output | TAUB0 TAUB0TTIN4, TAUB0TTIN14 |
| **FCN5:** | | |
| TSOUT | CAN time stamp output | TAUB0 TAUB0TTIN5, TAUB0TTIN15 |

Refer to section *"TAUB Input Selections"* in the *"Timer Array Unit B (TAUB)"* for details.

## 18.2 FCN0 and FCN1 connection

The FCN0 and FCN1 CAN Controllers have the option to be connected to the same CAN bus signals. This allow to operate the two CAN Controllers on the same CAN bus (FCN1 signals) thus allowing twice the number of message buffer support on this bus.

The following figure depicts the FCN0 and FCN1 connection scheme:

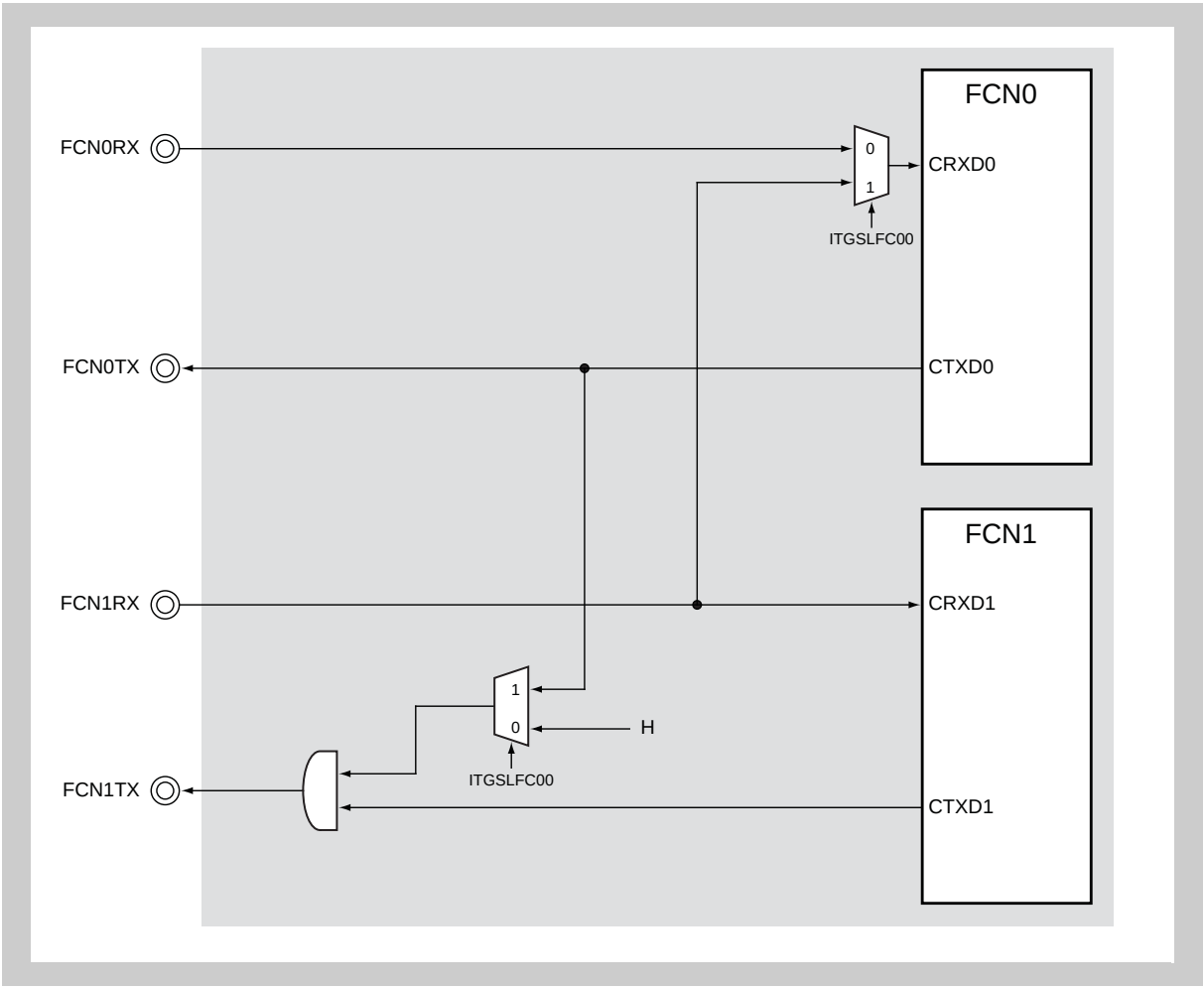



**Figure 18-2 FCN0 and FCN1 connection scheme**

### (1) ITGSLFC0 - FCN0 signal connection selection register

This register selects the signals of FCN0.

**Access** This register can be read/written in 8-bit units.

**Address** FF77 2008$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ITGSLFC00 |
| R | R | R | R | R | R | R | R/W |

**Table 18-9 ITGSLFC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | ITGSLFC00 | FCN0 signal seelction<br>0: use FCN0 bus signals (FCN0RX, FCN0TX)<br>1: use FNC1 bus signals (FCN1RX, FCN1TX) - combined operation |

## 18.3 CAN baudrate and time quanta

When configuring the CAN Controller (FCN) one of the following combinations for

- the CAN Controller clock $f_{CAN}$,
- the CAN baudrate and
- the number of time quantas (TQ) per data bit time (DBT)

must be used:

**Table 18-10 CAN baudrate settings**

| CAN baudrate [kbps] | CAN Controller clock $f_{CAN}$ [MHz] | TQs per DBT |
|---|---|---|
| 1000 | 16 | 8 |
| | 24 | 8 |
| | 32 | 8 |
| | 40 | 8, 10 |
| | 48 | 8, 12, 16 |
| 500 | 8 | 8, 16 |
| | 16 | 8, 16 |
| | 20 | 10 |
| | 24 | 8, 12 |
| | 32 | 8, 16 |
| | 40 | 8, 10, 20 |
| | 48 | 8, 12, 16, 24 |
| 250/125 | 8 | 8, 16 |
| | 16 | 8, 16 |
| | 20 | 10, 20 |
| | 24 | 8, 12, 24 |
| | 32 | 8, 16 |
| | 40 | 8, 10, 20 |
| | 48 | 8, 12, 16, 24 |

Thus make sure to set the

- global clock selection register FCNnGMCSPRE
- module bit rate prescaler register FCNnCMBRPRS
- module bit rate register FCNnCMBTCTL

accordingly.

**Note** The table describes common combinations. For other combinations please contact your local Renesas engineering representative.

## 18.4 Features

- Compliant with ISO 11898 and tested according to ISO/DIS 16845 (CAN conformance test)
- Standard frame and extended frame transmission/reception enabled
- Transfer rate: 1 Mbps max. (if CAN Controller clock input ≥ 16 MHz, for 64 or more message buffers)
- 32, 64 or 128 message buffers per channel
  For the number of available message buffers for each CAN Controller refer to the key word "Message buffers index m" in the first section of this chapter.
- Receive/transmit history list function, with enable flag for each message buffer individually
- Automatic block transmission function
- Multi-buffer receive block function
- Mask setting of 8 patterns is possible for each channel, applicable for data and remote frames
- Data bit time, communication baudrate and sample point can be controlled by CAN channel bit-rate prescaler register (FCNnCMBRPRS) and bit rate register (FCNnCMBTCTL)
  - As an example the following sample-point configurations can be configured:
  - 66.7%, 70.0%, 75.0%, 80.0%, 81.3%, 85.0%, 87.5%
  - Baudrates in the range of 10 kbps up to 1000 kbps can be configured
- Enhanced features:
  - Each message buffer can be configured to operate as a transmit or a receive message buffer
  - A transmission request can be aborted by clearing the dedicated Transmit-Request flag of the concerned message buffer. Supported by Transmission Abort Interrupt, on successful abortion.
  - Automatic block transmission operation mode (ABT)
  - Time stamp function in collaboration with timers capture channels
  - Centralized data new flag register, collecting all data new flags to be read from one location

### 18.4.1 Overview of functions

*Table 18-11 “Overview of functions”* presents an overview of the CAN Controller functions.

**Table 18-11 Overview of functions**

| Function | Details |
|---|---|
| Protocol | CAN protocol ISO 11898 (standard and extended frame transmission/reception) |
| Baudrate | Maximum 1 Mbps (minimum CAN Controller clock input = 16 MHz) |
| Data storage | Storing messages in the CAN RAM |
| Number of messages | • 32, 64 or 128 message buffers per channel<br>For the number of available message buffers for each CAN Controller refer to the key word “Message buffers index m” in the first section of this chapter.<br>• Each message buffer can be set to be either a transmit message buffer or a receive message buffer. |
| Message reception | • Unique ID can be set to each message buffer.<br>• Mask setting of 8 patterns is possible for each channel, applicable for data and remote frames<br>• A receive completion interrupt is generated each time a message is received and stored in a message buffer.<br>• Two or more receive message buffers can be used as a FIFO receive buffer (multi-buffer receive block function).<br>• Receive history list function, with enable flag for each message buffer individually<br>• Centralized Data New Flag register |
| Message transmission | • Unique ID can be set to each message buffer.<br>• Transmit completion interrupt for each message buffer<br>• Transmit Abort interrupt and -flag for each message buffer (only one transmission of any buffer can be aborted at a time)<br>• Message buffer number 0 to 7 (m = 32) / 15 (m = 64) / 31(m = 128) specified as the transmit message buffer can be set for automatic block transfer. Message transmission interval is programmable (automatic block transmission function (hereafter referred to as “ABT”)).<br>• Transmission history list function, with enable flag for each message buffer individually |
| Remote frame processing | • Remote frame processing by transmit message buffer<br>• Remote frame processing by receive message buffer, when applying one of the 8 masks |
| Time stamp function | • The time stamp function can be set for a message reception when a 16-bit timer is used in combination.<br>• Time stamp capture trigger can be selected (SOF or EOF in a CAN message frame can be detected.). |
| Diagnostic function | • Readable error counters<br>• “Valid protocol operation flag” for verification of bus connections<br>• Receive-only mode<br>• Single-shot mode<br>• CAN protocol error type decoding<br>• Self-test mode |
| Release from bus-off state | • Forced release from bus-off (by ignoring timing constraint) possible by software.<br>• No automatic release from bus-off (software must re-enable). |
| Power save mode | • Sleep mode (can be woken up by CAN bus)<br>• Stop mode (cannot be woken up by CAN bus) |

### 18.4.2 Configuration

The CAN Controller is composed of the following four blocks.

- PBUS interface
  This functional block provides an PBUS interface and means of transmitting and receiving messages between the CAN Controller and the host CPU.
- MAC (Memory Access Controller)
  This functional block controls access to the CAN protocol layer and to the CAN RAM within the CAN Controller.
- CAN protocol layer
  This functional block is involved in the operation of the CAN protocol and its related settings.
- CAN RAM
  The CAN RAM is used to store message IDs, message data, etc.

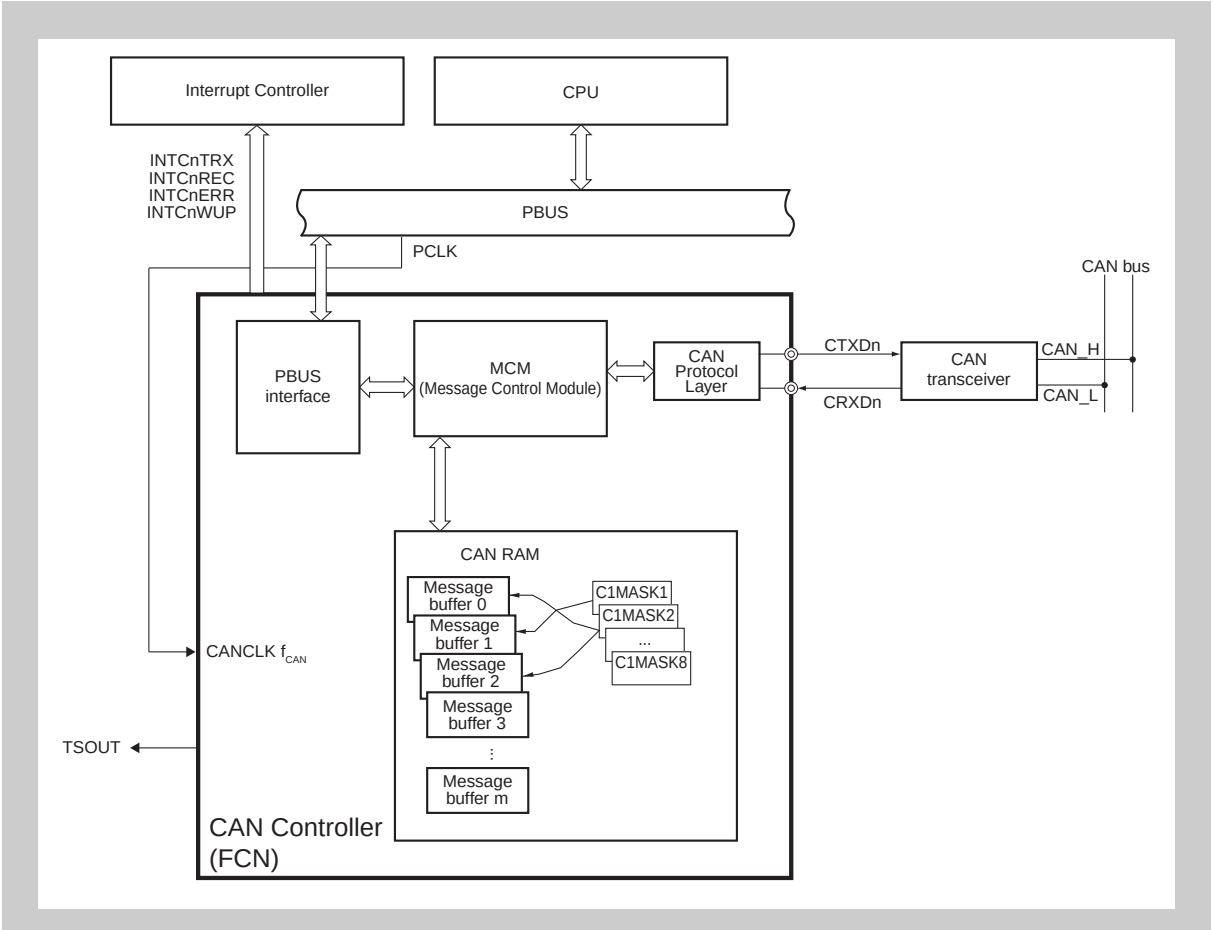



**Figure 18-3 Block diagram of the CAN Controller**

**Note** For the number m of available message buffers for each CAN Controller refer to the key word “Message buffers index m” in the first section of this chapter.

**CAN RAM** The CAN RAM is equipped with a memory checking module. If a CAN RAM error occurs during a software reset, the message buffer RAM read error detection bit FCNnGMCLCTL.FCNnGMCLECCF is set.
Check the CAN Controller functionality in such case.

# 18.5 Internal Registers of CAN Controller

## 18.5.1 CAN Controller configuration

**Table 18-12 List of CAN Controller registers (1/2)**

| Item | Register Name |
|---|---|
| CAN Controller global registers | FCNn global control register (FCNnGMCLCTL) |
| | FCNn global clock selection register (FCNnGMCSPRE) |
| | FCNn global automatic block transmission control register (FCNnGMABCTL) |
| | FCNn global automatic block transmission delay setting register (FCNnGMADCTL) |
| | FCNn global Data New bit monitor registers (FCNnDNBMRX0 - FCNnDNBMRX3) |
| CAN channel registers | FCNn channel mask 1 registers (FCNnCMMKCTL01H, FCNnCMMKCTL02H, FCNnCMMKCTL01W) |
| | FCNn channel mask 2 registers (FCNnCMMKCTL03H, FCNnCMMKCTL04H, FCNnCMMKCTL03W) |
| | FCNn channel mask3 registers (FCNnCMMKCTL05H, FCNnCMMKCTL06H, FCNnCMMKCTL05W) |
| | FCNn channel mask 4 registers (FCNnCMMKCTL07H, FCNnCMMKCTL08H, FCNnCMMKCTL07W) |
| | FCNn channel mask 5 registers (FCNnCMMKCTL09H, FCNnCMMKCTL10H, FCNnCMMKCTL09W) |
| | FCNn channel mask 6 registers (FCNnCMMKCTL11H, FCNnCMMKCTL12H, FCNnCMMKCTL11W) |
| | FCNn channel mask registers (FCNnCMMKCTL13H, FCNnCMMKCTL14H, FCNnCMMKCTL13W) |
| | FCNn channel mask 8 registers (FCNnCMMKCTL15H, FCNnCMMKCTL16H, FCNnCMMKCTL15W) |
| | FCNn channel control register (FCNnCMCLCTL) |
| | FCNn channel last error information register (FCNnCMLCSTR) |
| | FCNn channel information register (FCNnCMLCSTR) |
| | FCNn channel error counter register (FCNnCMERCNT) |
| | FCNn channel last error code register (FCNnCMLCSTR) |
| | FCNn channel interrupt enable register (FCNnCMIECTL) |
| | FCNn channel interrupt status register (FCNnCMISCTL) |
| | FCNn channel bit rate prescaler and clock selector register (FCNnCMBRPRS) |
| | FCNn channel bit rate register (FCNnCMBTCTL) |
| | FCNn channel last in-pointer register (FCNnCMLISTR) |
| | FCNn channel receive history list register (FCNnCMRGRX) |
| | FCNn channel last out-pointer register (FCNnCMLOSTR) |
| | FCNn channel transmit history list register (FCNnCMTGTX) |
| | FCNn channel time stamp register (FCNnCMTSCTL) |
| | FCNn Controller message data byte 0 to 3 registers m (FCNnMmDAT0W, FCNnMmDAT0H, FCNnMmDAT2H, FCNnMmDAT0B, FCNnMmDAT1B, FCNnMmDAT2B, FCNnMmDAT3B) |

**Table 18-12 List of CAN Controller registers (2/2)**

| Item | Register Name |
|---|---|
| CAN Controller message buffer registers | FCNn message data byte 4 to 7 registers m (FCNnMmDAT4W, FCNnMmDAT4H, FCNnMmDAT6H, FCNnMmDAT4B, FCNnMmDAT5B, FCNnMmDAT6B, FCNnMmDAT7B) |
| | FCNn message data length register m (FCNnMmDTLGB) |
| | FCNn message configuration register m (FCNnMmSTRB) |
| | FCNn message ID registers m (FCNnMmMID0H, FCNnMmMID1H, FCNnMmMID0W) |
| | FCNn message control register m (FCNnMmCTL) |

### 18.5.2 CAN Controller Registers Overview

**Address offset** All register addresses are given as offsets to the base address <FCNn_base>. The <FCNn_base> addresses of the registers are defined in the first section of this chapter under the keyword “Register addresses”.

#### (1) CAN Controller global and channel registers

Table 18-13 CAN Controller global and channel registers (1/2)

| Address offset | Register name | Symbol | R/W | Access bit | After reset |
|---|---|---|---|---|---|
| 0 0008$_H$ | FCNn global clock selection register | FCNnGMCSPRE | R/W | 8 | 0F$_H$ |
| 0 0020$_H$ | FCNn global automatic block transmission delay register | FCNnGMADCTL | R/W | 8 | 00$_H$ |
| 0 8000$_H$ | FCNn global control register | FCNnGMCLCTL | R/W | 16 | 00x0$_H$[a] |
| 0 8018$_H$ | FCNn global automatic block transmission register | FCNnGMABCTL | R/W | 16 | 0000$_H$ |
| 1 00C0$_H$ | FCNn global data new bit monitor register 0 | FCNnDNBMRX0 | R | 32 | 0000 0000$_H$ |
| 1 00D0$_H$ | FCNn global data new bit monitor register 1 | FCNnDNBMRX1 | R | 32 | 0000 0000$_H$ |
| 1 00E0$_H$ | FCNn global data new bit monitor register 2 | FCNnDNBMRX2 | R | 32 | 0000 0000$_H$ |
| 1 00F0$_H$ | FCNn global data new bit monitor register 3 | FCNnDNBMRX3 | R | 32 | 0000 0000$_H$ |
| 0 8300$_H$ | FCNn channel mask 1 register | FCNnCMMKCTL01H | R/W | 16 | 0000$_H$ |
| 0 8308$_H$ | | FCNnCMMKCTL02H | | | |
| 1 0300$_H$ | | FCNnCMMKCTL01W | | 32 | 0000 0000$_H$ |
| 0 8310$_H$ | FCNn channel mask 2 register | FCNnCMMKCTL03H | R/W | 16 | 0000$_H$ |
| 0 8318$_H$ | | FCNnCMMKCTL04H | | | |
| 1 0310$_H$ | | FCNnCMMKCTL03W | | 32 | 0000 0000$_H$ |
| 0 8320$_H$ | FCNn channel mask 3 register | FCNnCMMKCTL05H | R/W | 16 | 0000$_H$ |
| 0 8328$_H$ | | FCNnCMMKCTL06H | | | |
| 1 0320$_H$ | | FCNnCMMKCTL05W | | 32 | 0000 0000$_H$ |
| 0 8330$_H$ | FCNn channel mask 4 register | FCNnCMMKCTL07H | R/W | 16 | 0000$_H$ |
| 0 8338$_H$ | | FCNnCMMKCTL08H | | | |
| 1 0330$_H$ | | FCNnCMMKCTL07W | | 32 | 0000 0000$_H$ |
| 0 8340$_H$ | FCNn channel mask 5 register | FCNnCMMKCTL09H | R/W | 16 | 0000$_H$ |
| 0 8348$_H$ | | FCNnCMMKCTL10H | | | |
| 1 0340$_H$ | | FCNnCMMKCTL09W | | 32 | 0000 0000$_H$ |
| 0 8350$_H$ | FCNn channel mask 6 register | FCNnCMMKCTL11H | R/W | 16 | 0000$_H$ |
| 0 8358$_H$ | | FCNnCMMKCTL12H | | | |
| 1 0350$_H$ | | FCNnCMMKCTL11W | | 32 | 0000 0000$_H$ |

**Table 18-13 CAN Controller global and channel registers (2/2)**

| Address offset | Register name | Symbol | R/W | Access bit | After reset |
|---|---|---|---|---|---|
| 0 8360$_H$ | FCNn channel mask 7 register | FCNnCMMKCTL13H | R/W | 16 | 0000$_H$ |
| 0 8368$_H$ | | FCNnCMMKCTL14H | | | |
| 1 0360$_H$ | | FCNnCMMKCTL13W | | 32 | 0000 0000$_H$ |
| 0 8370$_H$ | FCNn channel mask 8 register | FCNnCMMKCTL15H | R/W | 16 | 0000$_H$ |
| 0 8378$_H$ | | FCNnCMMKCTL16H | | | |
| 1 0370$_H$ | | FCNnCMMKCTL15W | | 32 | 0000 0000$_H$ |
| 0 0248$_H$ | FCNn channel last error code register | FCNnCMLCSTR | R/W | 8 | 00$_H$ |
| 0 024C$_H$ | FCNn channel information register | FCNnCMINSTR | R | 8 | 00$_H$ |
| 0 0268$_H$ | FCNn channel bit-rate prescaler and clock selector register | FCNnCMBRPRS | R/W | 8 | FF$_H$ |
| 0 0278$_H$ | FCNn channel last in-pointer register | FCNnCMLISTR | R | 8 | Undefined |
| 0 0288$_H$ | FCNn channel last out-pointer register | FCNnCMLOSTR | R | 8 | Undefined |
| 0 8240$_H$ | FCNn channel control register | FCNnCMCLCTL | R/W | 16 | 0000$_H$ |
| 0 8250$_H$ | FCNn channel error counter register | FCNnCMERCNT | R | 16 | 0000$_H$ |
| 0 8258$_H$ | FCNn channel interrupt enable register | FCNnCMIECTL | R/W | 16 | 0000$_H$ |
| 0 8260$_H$ | FCNn channel interrupt status register | FCNnCMISCTL | R/W | 16 | 0000$_H$ |
| 0 8270$_H$ | FCNn channel bit-rate register | FCNnCMBTCTL | R/W | 16 | 370F$_H$ |
| 0 8280$_H$ | FCNn channel receive history list register | FCNnCMRGRX | R/W | 16 | xx02$_H$ |
| 0 8290$_H$ | FCNn channel transmit history list register | FCNnCMTGTX | R/W | 16 | xx02$_H$ |
| 0 8298$_H$ | FCNn channel time stamp register | FCNnCMTSCTL | R/W | 16 | 0000$_H$ |

a) Initial value depends on FCNnGMCLCTL.FCNnGMCLECCF, which indicates error detections when reading from message buffer RAM. Refer to the detailed description of the FCNnGMCLCTL register.

### 18.5.3 Register bit configuration

**Table 18-14 CAN Controller global register bit configuration**

| Address offset | Symbol | Bit 7/ 15/23/ 31 | Bit 6/ 14/22/ 30 | Bit 5/ 13/21/ 29 | Bit 4/ 12/20/ 28 | Bit 3/ 11/19/ 27 | Bit 2/ 10/18/ 26 | Bit 1/ 9/17/ 25 | Bit 0/ 8/16/ 24 |
|---|---|---|---|---|---|---|---|---|---|
| 0 8000$_H$ | FCNnGMCLCTL (W) | 0 | 0 | FCNnGMCLCLMB | | 0 | 0 | 0 | FCNnGMCLCLOM |
| | | 0 | 0 | 0 | FCNnGMCLSESR | 0 | 0 | FCNnGMCLSESD | FCNnGMCLSEOM |
| | FCNnGMCLCTL (R) | 0 | 0 | FCNnGMCLECCF | FCNnGMCLSORF | 0 | 0 | FCNnGMCLESDE | FCNnGMCLPWOM |
| | | FCN0GMCLSSMO | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 0008$_H$ | FCNnGMCSPRE | 0 | 0 | 0 | 0 | FCNnGMCSPRSC[3:0] | | | |
| 0 8018$_H$ | FCNnGMABCTL (W) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FCNnGMABCLAT |
| | | 0 | 0 | 0 | 0 | 0 | 0 | FCNnGMABSEAC | FCNnGMABSEAT |
| | FCNnGMABCTL (R) | 0 | 0 | 0 | 0 | 0 | 0 | FCNnGMABCLRF | FCNnGMABABTT |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 0020$_H$ | FCNnGMADCTL | 0 | 0 | 0 | 0 | FCNnGMADSSAD[3:0] | | | |
| 1 00C0$_H$ | FCNnDNBMRX0 (R) | FCNnDNBMSSDN[7:0] | | | | | | | |
| | | FCNnDNBMSSDN[15:8] | | | | | | | |
| | | FCNnDNBMSSDN[23:16] | | | | | | | |
| | | FCNnDNBMSSDN[31:24] | | | | | | | |
| 1 00D0$_H$ | FCNnDNBMRX1 (R)[a] | FCNnDNBMSSDN[39:32] | | | | | | | |
| | | FCNnDNBMSSDN[47:40] | | | | | | | |
| | | FCNnDNBMSSDN[55:48] | | | | | | | |
| | | FCNnDNBMSSDN[63:56] | | | | | | | |
| 1 00E0$_H$ | FCNnDNBMRX2 (R)[b] | FCNnDNBMSSDN[71:64] | | | | | | | |
| | | FCNnDNBMSSDN[79:72] | | | | | | | |
| | | FCNnDNBMSSDN[87:80] | | | | | | | |
| | | FCNnDNBMSSDN[95:88] | | | | | | | |
| 1 00F0$_H$ | FCNnDNBMRX3 (R)[b] | FCNnDNBMSSDN[103:96] | | | | | | | |
| | | FCNnDNBMSSDN[111:104] | | | | | | | |
| | | FCNnDNBMSSDN[119:112] | | | | | | | |
| | | FCNnDNBMSSDN[127:120] | | | | | | | |

a) Only available with 64 (m = 0 to 63) and 128 message buffers (m = 0 to 127)
b) Only available with 128 message buffers (m = 0 to 127)

**Table 18-15 CAN channel mask control 16-bit registers bit configuration**

| Address offset | Symbol | Bit 15 | Bit 14 | Bit 13 | Bit 12 to 0 |
|---|---|---|---|---|---|
| 0 8300$_H$ | FCNnCMMK CTL01H | FCNnCMMKSSID[15:0] | | | |
| 0 8308$_H$ | FCNnCMMK CTL02H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |
| 0 8310$_H$ | FCNnCMMK CTL03H | FCNnCMMKSSID[15:0] | | | |
| 0 8318$_H$ | FCNnCMMK CTL04H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |
| 0 8320$_H$ | FCNnCMMK CTL05H | FCNnCMMKSSID[15:0] | | | |
| 0 8328$_H$ | FCNnCMMK CTL06H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |
| 0 8330$_H$ | FCNnCMMK CTL07H | FCNnCMMKSSID[15:0] | | | |
| 0 8338$_H$ | FCNnCMMK CTL08H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |
| 0 8340$_H$ | FCNnCMMK CTL09H | FCNnCMMKSSID[15:0] | | | |
| 0 8348$_H$ | FCNnCMMK CTL10H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |
| 0 8350$_H$ | FCNnCMMK CTL11H | FCNnCMMKSSID[15:0] | | | |
| 0 8358$_H$ | FCNnCMMK CTL12H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |
| 0 8360$_H$ | FCNnCMMK CTL13H | FCNnCMMKSSID[15:0] | | | |
| 0 8368$_H$ | FCNnCMMK CTL14H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |
| 0 8370$_H$ | FCNnCMMK CTL15H | FCNnCMMKSSID[15:0] | | | |
| 0 8378$_H$ | FCNnCMMK CTL16H | 0 | 0 | 0 | FCNnCMMKSSID[28:16] |

**Table 18-16 CAN channel mask control 32-bit registers bit configuration (1/2)**

| Address offset | Symbol | Bit 31 | Bit 30 | Bit 29 | Bit 28 to 0 |
|---|---|---|---|---|---|
| 1 0300$_H$ | FCNnCMMK CTL01W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |
| 1 0310$_H$ | FCNnCMMK CTL03W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |
| 1 0320$_H$ | FCNnCMMK CTL05W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |
| 1 0330$_H$ | FCNnCMMK CTL07W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |
| 1 0340$_H$ | FCNnCMMK CTL09W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |

**Table 18-16 CAN channel mask control 32-bit registers bit configuration (2/2)**

| Address offset | Symbol | Bit 31 | Bit 30 | Bit 29 | Bit 28 to 0 |
|---|---|---|---|---|---|
| 1 0350$_H$ | FCNnCMMKCTL11W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |
| 1 0360$_H$ | FCNnCMMKCTL13W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |
| 1 0370$_H$ | FCNnCMMKCTL15W | 0 | 0 | 0 | FCNnCMMKSSID[28:0] |

**Table 18-17 CAN channel register bit configuration (1/2)**

| Address offset | Symbol | Bit 7/15 | Bit 6/14 | Bit 5/13 | Bit 4/12 | Bit 3/11 | Bit 2/10 | Bit 1/9 | Bit 0/8 |
|---|---|---|---|---|---|---|---|---|---|
| 0 8240$_H$ | FCNnCMCLCTL (W) | 0 | FCNnCMCLCLAL | FCNnCMCLCLVL | FCNnCMCLCLPS[1:0] | | FCNnCMCLCLOP[2:0] | | |
| | | FCNnCMCLSERC | FCNnCMCLCSEAL | 0 | FCNnCMSESEPS[1:0] | | FCNnCMCSELOP[2:0] | | |
| | FCNnCMCLCTL (R) | FCNnCMCLERCF | FCNnCMCLALBF | FCNnCMCLVALF | FCNnCMCLMDPF[1:0] | | FCNnCMCLMDOF[2:0] | | |
| | | 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMCLSSRS | FCNnCMCLSSTS |
| 0 00248$_H$ | FCNnCMLCSTR (W) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | FCNnCMLCSTR (R) | 0 | 0 | 0 | 0 | 0 | FCNnCMLCSSLC[2:0] | | |
| 0 024CH | FCNnCMINSTR | 0 | 0 | 0 | FCNnCMINBOFF | FCNnCMINSSTE[1:0] | | FCNnCMINSSRE[1:0] | |
| 0 8250$_H$ | FCNnCMERCNT | FCNnCMERTECF[7:0] | | | | | | | |
| | | FCNnCMERRPSF | FCNnCMERRECF[6:0] | | | | | | |
| 0 8258$_H$ | FCNnCMIECTL (W) | 0 | FCNnCMIECLIE[6:0] | | | | | | |
| | | 0 | FCNnCMIESEIE[6:0] | | | | | | |
| | FCNnCMIECTL (R) | 0 | FCNnCMIEINTF[6:0] | | | | | | |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 8260$_H$ | FCNnCMISCTL (W) | 0 | FCNnCMISCLTS[6:0] | | | | | | |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | FCNnCMISCTL (R) | 0 | FCNnCMISITSF[6:0] | | | | | | |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 0268$_H$ | FCNnCMBRPRS | FCNnCMBRPRS[7:0] | | | | | | | |
| 0 8270$_H$ | FCNnCMBTCTL | 0 | 0 | 0 | 0 | FCNnCMBTS1LG[3:0] | | | |
| | | 0 | 0 | FCNnCMBTJWLG[1:0] | | 0 | FCNnCMBTS2LG[2:0] | | |
| 0 0278$_H$ | FCNnCMLISTR | FCNnCMLISSLR[7:0] | | | | | | | |

**Table 18-17 CAN channel register bit configuration (2/2)**

| Address offset | Symbol | Bit 7/15 | Bit 6/14 | Bit 5/13 | Bit 4/12 | Bit 3/11 | Bit 2/10 | Bit 1/9 | Bit 0/8 |
|---|---|---|---|---|---|---|---|---|---|
| 0 8280$_H$ | FCNnCM RGRX (W) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMR GCLRV |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | FCNnCM RGRX (R) | 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMR GSSPM | FCNnCMR GRVFF |
| | | FCNnCMRGSSPT[7:0] | | | | | | | |
| 0 0288$_H$ | FCNnCM LOSTR | FCNnCMLOSTRSSLT[7:0] | | | | | | | |
| 0 8290$_H$ | FCNnCM TGTX (W) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMT GCLTV |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | FCNnCM TGTX (R) | 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMT GSSPM | FCNnCMT GTVFF |
| | | FCNnCMTGSSPT[7:0] | | | | | | | |
| 0 8298$_H$ | FCNnCM TSCTL (W) | 0 | 0 | 0 | 0 | 0 | FCNnCMT SCLLK | FCNnCMT SCLSL | FCNnCMT SCLTS |
| | | 0 | 0 | 0 | 0 | 0 | FCNnCMT SSELK | FCNnCMT SSESL | FCNnCMT SSETS |
| | FCNnCM TSCTL (R) | 0 | 0 | 0 | 0 | 0 | FCNnCMT SLOKE | FCNnCMT SSELE | FCNnCMT STSGE |
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 18-18 Message buffer register bit configuration (1/2)**

| Address offset | Symbol | Bit 7/15/ 23/31 | Bit 6/14/ 22/30 | Bit 5/13/ 21/29 | Bit 4/12/ 20/28 | Bit 3/11/ 19/27 | Bit 2/10/ 18/26 | Bit 1/9/ 17/25 | Bit 0/8/ 16/24 |
|---|---|---|---|---|---|---|---|---|---|
| 1 1000$_H$ + m × 40$_H$ | FCNnMm DAT0W | FCNnMmSSD0[7:0] | | | | | | | |
| | | FCNnMmSSD1[7:0] | | | | | | | |
| | | FCNnMmSSD2[7:0] | | | | | | | |
| | | FCNnMmSSD3[7:0] | | | | | | | |
| 0 9000$_H$ + m × 40$_H$ | FCNnMm DAT0H | FCNnMmSSD0[7:0] | | | | | | | |
| | | FCNnMmSSD1[7:0] | | | | | | | |
| 0 1000$_H$ + m × 40$_H$ | FCNnMm DAT0B | FCNnMmSSD0[7:0] | | | | | | | |
| 0 1004$_H$ + m × 40$_H$ | FCNnMm DAT1B | FCNnMmSSD1[7:0] | | | | | | | |
| 0 9008$_H$ + m × 40$_H$ | FCNnMm DAT2H | FCNnMmSSD2[7:0] | | | | | | | |
| | | FCNnMmSSD3[7:0] | | | | | | | |
| 0 1008$_H$ + m × 40$_H$ | FCNnMm DAT2B | FCNnMmSSD2[7:0] | | | | | | | |
| 0 100C$_H$ + m × 40$_H$ | FCNnMm DAT3B | FCNnMmSSD3[7:0] | | | | | | | |

**Table 18-18 Message buffer register bit configuration (2/2)**

| Address offset | Symbol | Bit 7/15/23/31 | Bit 6/14/22/30 | Bit 5/13/21/29 | Bit 4/12/20/28 | Bit 3/11/19/27 | Bit 2/10/18/26 | Bit 1/9/17/25 | Bit 0/8/16/24 |
|---|---|---|---|---|---|---|---|---|---|
| $1\ 1010_H$ + m × $40_H$ | FCNnMm DAT4W | FCNnMmSS4[7:0] | | | | | | | |
| | | FCNnMmSSD5[7:0] | | | | | | | |
| | | FCNnMmSSD6[7:0] | | | | | | | |
| | | FCNnMmSSD7[7:0] | | | | | | | |
| $0\ 9010_H$ + m × $40_H$ | FCNnMm DAT4H | FCNnMmSSD4[7:0] | | | | | | | |
| | | FCNnMmSSD5[7:0] | | | | | | | |
| $0\ 1010_H$ + m × $40_H$ | FCNnMm DAT4B | FCNnMmSSD4[7:0] | | | | | | | |
| $0\ 1014_H$ + m × $40_H$ | FCNnMm DAT5B | FCNnMmSSD5[7:0] | | | | | | | |
| $0\ 9018_H$ + m × $40_H$ | FCNnMm DAT6H | FCNnMmSSD6[7:0] | | | | | | | |
| | | FCNnMmSSD7[7:0] | | | | | | | |
| $0\ 1018_H$ + m × $40_H$ | FCNnMm DAT6B | FCNnMmSSD6[7:0] | | | | | | | |
| $0\ 101C_H$ + m × $40_H$ | FCNnMm DAT7B | FCNnMmSSD7[7:0] | | | | | | | |
| $0\ 1020_H$ + m × $40_H$ | FCNnMm DTLGB | 0 | | | | FCNnMmDTLG[3:0] | | | |
| $0\ 1024_H$ + m × $40_H$ | FCNnMm STRB | FCNnMm SSOW | FCNnMmSSMT[3:0] | | | | FCNnMm SSRT | 0 | FCNnMm SSAM |
| $0\ 9028_H$ + m × $40_H$ | FCNnMm MID0H | FCNnMmSSID[7:0] | | | | | | | |
| | | FCNnMmSSID[15:8] | | | | | | | |
| $0\ 9030_H$ + m × $40_H$ | FCNnMm MID1H | FCNnMmSSID[23:16] | | | | | | | |
| | | FCNnMm SSIE | 0 | 0 | FCNnMmSSID[28:24] | | | | |
| $1\ 1028_H$ + m × $40_H$ | FCNnMm MID0W | FCNnMmSSID[7:0] | | | | | | | |
| | | FCNnMmSSID[15:8] | | | | | | | |
| | | FCNnMmSSID[23:16] | | | | | | | |
| | | FCNnMm SSIE | 0 | 0 | FCNnMmSSID[28:24] | | | | |
| $0\ 9038_H$ + m × $40_H$ | FCNnMmCTL (W) | 0 | FCNnMm CLNH | 0 | FCNnMm CLMW | FCNnMm CLIE | FCNnMm CLDN | FCNnMm CLTR | FCNnMm CLRY |
| | | 0 | FCNnMm SENH | 0 | 0 | FCNnMm SEIE | 0 | FCNnMm CSETR | FCNnMm SERY |
| | FCNnMmCTL (R) | 0 | FCNnMm NHMF | 0 | FCNnMm MOWF | FCNnMm IENF | FCNnMm DTNF | FCNnMm TRQF | FCNnMm RDYF |
| | | 0 | 0 | FCNnMm MUCF | 0 | 0 | 0 | FCNnMm TCPF | 0 |

## 18.6 Bit Set/Clear Function

The CAN Controller control registers include registers whose bits can be set or cleared via the CPU and via the CAN Controller. These register bits can not be changed directly by the CPU by any bit manipulation instructions, such as SET1, CLR1, and NOT1. Instead a special bit-set/bit-clear mechanism is used.

All registers where bit manlpulation operations are prohibited are organised in such a way that all bits allowed for changing by the CPU are located in the lower byte (RWx in the register layout below), while in the upper byte either no or read-only information is located (ROx in the register layout below).

The registers can be read in the usual way getting all 16 data bits in their current setting and as described in the register description.
For setting or clearing any of the lower 8 bits the following mechanism is implemented:

When writing 16-bit data to the register address

**Bit clear**

- each of the lower 8 data bits (CLx in the register layout below) indicates whether the corresponding register bit RWx should be
  - cleared, i.e. set to 0: if CLx = 1, the corresponding RWx is cleared to 0
  - remain unchanged: if CLx = 0, the corresponding RWx does not change

**Bit set**

- each of the upper 8 data bits (SEx in the register layout below) indicate whether the corresponding register bit should be
  - set, i.e. set to 1: if SEx = 1, the corresponding RWx is set to 1
  - remain unchanged: if SEx = 0, the corresponding RWx does not change

Register layout for read access:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RO7 | RO6 | RO5 | RO4 | RO3 | RO2 | RO1 | RO0 | RW7 | RW6 | RW5 | RW4 | RW3 | RW2 | RW1 | RW0 |
| changing by the CPU not possible | | | | | | | | bits for CPU manipulation via SE7-SE0 and CL7-CL0 | | | | | | | |

Register layout for write access:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SE7 | SE6 | SE5 | SE4 | SE3 | SE2 | SE1 | SE0 | CL7 | CL6 | CL5 | CL4 | CL3 | CL2 | CL1 | CL0 |
| SEx = 1 sets the corresponding RW7-RW0 | | | | | | | | CLx = 1 clears the corresponding RW7-RW0 | | | | | | | |

The following table denotes the operations applied to the RWx bits:

**Table 18-19 Bit set/clear operation**

| CLx | SEx | Operation on RWx |
|---|---|---|
| 0 | 0 | no change of RWx |
| 0 | 1 | RWx set to 1 |
| 1 | 0 | RWx cleared to 0 |
| 1 | 1 | no change of RWx |

**Example** Following an example.

The register with the content $1883_H$ shall be changed so, that

- bit 3 shall be set to 1: SE3 = 1

- bit 1 shall be cleared to 0: CL1 = 1

Register read before bit manipulations:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | **0** | 0 | **1** | 1 |
| may hold any value, here $18_H$ | | | | | | | | RW7-RW0: $83_H$ | | | | | | | |

Register write access:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 |
| SE3 = 1: $08_H$ | | | | | | | | CL1 = 1: $02_H$ | | | | | | | |

Register read after bit manipulations:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | **1** | 0 | **0** | 1 |
| may hold any value, here $18_H$ | | | | | | | | RW7-RW0: $89_H$ | | | | | | | |

# 18.7 Control Registers

## 18.7.1 CAN Controller global registers

### (1) FCNnGMCLCTL - FCNn global control register

This register is used to control the operation of the CAN Controller.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8000_H$

**Initial Value** The initial value depends on the occurrence of errors in relation to a software reset:

- no CAN RAM error detection after CAN S/W reset: $0000_H$
- no CAN RAM error detection while CAN S/W reset is ongoing: $0010_H$
- CAN RAM error detection after CAN S/W reset: $0020_H$
- CAN RAM error detection while CAN S/W reset is ongoing: $0030_H$

#### (a) FCNnGMCLCTL read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnGMCLSSMO | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | FCNnGMCLECCF | FCNnGMCLSORF | 0 | 0 | FCNnGMCLESDE | FCNnGMCLPWOM |

| FCNnGMCLSSMO | Bit enabling access to message buffer register, transmit/receive history registers |
|---|---|
| 0 | Write access and read access to the message buffer register and the transmit/receive history list registers is disabled. |
| 1 | Write access and read access to the message buffer register and the transmit/receive history list registers is enabled. |

**Cautions**

1. While the FCNnGMCLCTL.FCNnGMCLSSMO is cleared (to 0), software access to all message buffers and message buffer registers (i.e. all registers with name prefix FCNnMm...), or registers related to transmit history or receive history (FCNnCMLOSTR, FCNnCMTGTX, FCNnCMLISTR, and FCNnCMRGRX) is disabled.
2. FCNnGMCLCTL.FCNnGMCLSSMO is read-only. Even if 1 is written while it is 0, its value does not change, and access to the message buffer registers, or registers related to transmit history or receive history remains disabled.

**Note** FCNnGMCLCTL.FCNnGMCLSSMO is cleared to 0 when the CAN Controller enters sleep/stop, or when the FCNnGMCLCTL.FCNnGMCLPWOM is cleared to 0.
FCNnGMCLSSMO is set to 1 when the CAN Controller sleep/stop mode is released, or when the FCNnGMCLCTL.FCNnGMCLPWOM is set to 1.

| FCNnGMCLECCF | Message buffer RAM read error detect bit |
|---|---|
| 0 | Not detect error for reading from message buffer RAM. |
| 1 | Detect error for reading from message buffer RAM. |

**Notes**

1. FCNnGMCLCTL.FCNnGMCLECCF is set (1) in case of detecting a memory error when reading from the message buffer RAM during the soft reset process. Once FCNnGMCLECCF is set (1), it keeps the level until it is cleared (0).
2. Although reading from message buffer RAM happens, when
   - reading a message buffer register,
   - reading a register that is related to receive and transmit history lists,
   - sending (transmitting) a message or
   - receiving a message,

   FCNnGMCLECCF is only evaluated during soft reset is ongoing. During any other operation, FCNnGMCLECCF is not updated.
3. It is impossible to clear FCNnGMCLECCF (0) during FCNnGMCLCTL.FCNnGMCLSORF is set (1) (soft reset is ongoing).

| FCNnGMCLSORF | Soft reset execution status bit |
|---|---|
| 0 | No soft reset |
| 1 | Soft reset is ongoing |

**Notes**

1. While a soft reset is ongoing (FCNnGMCLCTL.FCNnGMCLSORF is set (1)), it is impossible to set FCNnGMCLCTL.FCNnGMCLPWOM and FCNnGMCLCTL.FCNnGMCLESDE.
   It is possible to set start a software reset by FCNnGMCLCTL.FCNnGMCLSESR = 1 during FCNnGMCLCTL.FCNnGMCLPWOM bit is clear (0).
2. When FCNnGMCLCTL.FCNnGMCLSORF is set (1), the initialization of message buffer RAM starts. It is possible to detect error during initializing message buffer RAM, if FCNnGMCLCTL.FCNnGMCLECCF is cleared before setting FCNnGMCLSORF.
3. When FCNnGMCLCTL.FCNnGMCLSORF is set (1) again in the condition that is is already set (1), the soft reset procedure does not restart, but continues.
4. After release of the H/W reset FCNnGMCLCTL.FCNnGMCLSORF is set (1) automatically and initialization of message buffer RAM starts.
5. It is impossible that clearing FCNnGMCLCTL.FCNnGMCLPWOM (0) and setting FCNnGMCLCTL.FCNnGMCLSORF (1) are done at the same time.

6. If a hardware RESET occurs during FCNnGMCLCTL.FCNnGMCLSORF = 1, then the soft reset procedure is stopped (aborted), and the hardware RESET starts.

| FCNnGMCLESDE | Bit enabling forced shut down |
|---|---|
| 0 | Forced shut down disabled. |
| 1 | Forced shut down of FCNnGMCLCTL.FCNnGMCLPWOM bit = 0 enabled. |

**Caution** To request a forced shut down, FCNnGMCLCTL.FCNnGMCLPWOM must be cleared to 0 in a subsequent, immediately following access after FCNnGMCLCTL.FCNnGMCLESDE has been set to 1. If any access to another register (including reading the FCNnGMCLCTL register) is executed without clearing FCNnGMCLPWOM immediately after FCNnGMCLESDE has been set to 1, FCNnGMCLESDE is forcibly cleared to 0, and the forced shut down request is invalid.

| FCNnGMCLPWOM | Global operation mode bit |
|---|---|
| 0 | CAN Controller is disabled. |
| 1 | CAN Controller is enabled to operate. |

**Caution** FCNnGMCLCTL.FCNnGMCLPWOM can be cleared only in the initialization mode or immediately after FCNnGMCLCTL.FCNnGMCLESDE is set (forced shutdown).

**(b) FCNnGMCLCTL write**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | FCNnGMCLSESR | 0 | 0 | FCNnGMCLSESD | FCNnGMCLSEOM |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | FCNnGMCLCLMB | 0 | 0 | 0 | 0 | FCNnGMCLCLOM |

| FCNnGMCLSESR | Software reset start |
|---|---|
| 0 | No changes. |
| 1 | Start soft reset. |

| FCNnGMCLSESD | FCNnGMCLSESD bit setting |
|---|---|
| 0 | No change in FCNnGMCLSESD bit. |
| 1 | FCNnGMCLESDE bit set to 1. |

| FCNnGMCLSEOM | FCNnGMCLCLOM | FCNnGMCLPWOM bit setting |
|---|---|---|
| 0 | 1 | FCNnGMCLCTL.FCNnGMCLPWOM bit cleared to 0. |
| 1 | 0 | FCNnGMCLCTL.FCNnGMCLPWOM bit set to 1. |
| Other than above | | No change of FCNnGMCLCTL.FCNnGMCLPWOM bit. |

**Caution** Set FCNnGMCLCTL.FCNnGMCLPWOM and FCNnGMCLCTL.FCNnGMCLESDE bit always separately.

| FCNnGMCLCLMB | FCNnGMCLCTL.FCNnGMCLECCF bit clear |
|---|---|
| 0 | No change in FCNnGMCLCTL.FCNnGMCLECCF bit. |
| 1 | FCNnGMCLCTL.FCNnGMCLECCF bit cleared to 0. |

### (2) FCNnGMCSPRE - FCNn global clock selection register

This register is used to select the CAN channel clock.

**Access** This register can be read/written in 8-bit units.

**Address** <FCNn_base> + $0008_H$

**Initial Value** $0F_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | FCNnGMCSPRSC[3:0] | | | |

| FCNnGMCSPRSC[3:0] | CAN channel clock ($f_{CANCHN}$) |
|---|---|
| $0000_B$ | $f_{CAN}/1$ |
| $0001_B$ | $f_{CAN}/2$ |
| $0010_B$ | $f_{CAN}/3$ |
| $0011_B$ | $f_{CAN}/4$ |
| $0100_B$ | $f_{CAN}/5$ |
| $0101_B$ | $f_{CAN}/6$ |
| $0110_B$ | $f_{CAN}/7$ |
| $0111_B$ | $f_{CAN}/8$ |
| $1000_B$ | $f_{CAN}/9$ |
| $1001_B$ | $f_{CAN}/10$ |
| $1010_B$ | $f_{CAN}/11$ |
| $1011_B$ | $f_{CAN}/12$ |
| $1100_B$ | $f_{CAN}/13$ |
| $1101_B$ | $f_{CAN}/14$ |
| $1110_B$ | $f_{CAN}/15$ |
| $1111_B$ | $f_{CAN}/16$ (default value) |

**Note** $f_{CAN}$ = clock supplied to CAN Controller.

**Caution** Setting of the channel clock, CAN baudrate and time quantas (TQs) per data bit time (DBT) is allowed only in certain combinations.
Refer to the section *"CAN baudrate and time quanta"* above in this chapter.

### (3) FCNnGMABCTL - FCNn global automatic block transmission control register

This register is used to control the automatic block transmission (ABT) operation.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8018_H$

**Initial Value** $0000_H$

#### (a) FCNnGMABCTL read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | FCNnGMABCLRF | FCNnGMABABTT |

| FCNnGMABCLRF | Automatic block transmission engine clear status bit |
|---|---|
| 0 | Clearing the automatic transmission engine is completed. |
| 1 | The automatic transmission engine is being cleared. |

**Notes** **1.** Start automatic transmission engine clearance by FCNnGMABCTL.FCNnGMABCLAT = 1 while FCNnGMABCTL.FCNnGMABCLRF = 0.
The operation is not guaranteed if FCNnGMABCLRF is set to 1 while FCNnGMABCLRF = 1.

| FCNnGMABABTT | Automatic block transmission status bit |
|---|---|
| 0 | Automatic block transmission is stopped. |
| 1 | Automatic block transmission is under execution. |

**(b) FCNnGMABCTL write**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | FCNnGMABSEAC | FCNnGMABSEAT |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | FCNnGMABCLAT |

**Note** When the automatic block transmission engine is cleared by setting FCNnGMABCTL.FCNnGMABSEAC to 1, FCNnGMABCLRF is automatically set, and cleared to 0 as soon as the requested clearing processing is completed.

**Cautions**

1. Before changing the normal operation mode with ABT to the initialization mode, be sure to set the FCNnGMABCTL register to the default value ($0000_H$) and confirm the FCNnGMABCTL register is surely initialized to the default value ($0000_H$).
2. Do not start automatic block transmission in the initialization mode. If automatic block transmission is started in the initialization mode, the operation is not guaranteed after the CAN Controller has entered the normal operation mode with ABT.
3. Do not start automatic block transmission while FCNnCMCLCTL.FCNnCMCLSSTS is set to 1 (transmission in progress). Confirm FCNnCMCLSSTS = 0 directly in advance before starting automatic block transmission.

| FCNnGMABSEAC | Automatic block transmission engine clear request bit |
|---|---|
| 0 | The automatic block transmission engine is in idle status or under operation. |
| 1 | Request to clear the automatic block transmission engine. After the automatic block transmission engine has been cleared, automatic block transmission is started from message buffer 0 by setting the FCNnGMABCTL.FCNnGMABABTT = 1. |

| FCNnGMABSEAT | FCNnGMABCLAT | Automatic block transmission start bit |
|---|---|---|
| 0 | 1 | Request to stop automatic block transmission. |
| 1 | 0 | Request to start automatic block transmission. |
| Other than above | | No change of FCNnGMABCTL.FCNnGMABABTT. |

### (4) FCNnGMADCTL - FCNn global automatic block transmission delay register

This register is used to set the interval at which the data of the message buffer assigned to ABT is to be transmitted in the normal operation mode with ABT.

**Access** This register can be read/written in 8-bit units.

**Address** <FCNn_base> + $0020_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | FCNnGMADSSAD[3:0] | | | |

| FCNnGMADSSAD[3:0] | Data frame interval during automatic block transmission in DBT[a] |
|---|---|
| $0000_B$ | 0 DBT (default value) |
| $0001_B$ | $2^5$ DBT |
| $0010_B$ | $2^6$ DBT |
| $0011_B$ | $2^7$ DBT |
| $0100_B$ | $2^8$ DBT |
| $0101_B$ | $2^9$ DBT |
| $0110_B$ | $2^{10}$ DBT |
| $0111_B$ | $2^{11}$ DBT |
| $1000_B$ | $2^{12}$ DBT |
| Other than above | Setting prohibited |

a) Unit: Data bit time (DBT)

**Cautions**

1. Do not change the contents of the FCNnGMADCTL register while FCNnGMABCTL.FCNnGMABCLRF = 1 (clearing of ABT in progress).
2. The timing at which the ABT message is actually transmitted onto the CAN bus differs depending on the status of transmission from the other station or how a request to transmit a message other than an ABT message is made.

### (5) FCNnDNBMRXk - FCNn data new bit monitor registers (k = 0 to 3)

These registers are used to read Data New Flags globally for several message buffers at a time.

**Access** These registers can be read in 32-bit units.

**Address** FCNnDNBMRX0: <FCNn_base> + 1 $00C0_H$
Following register is available only with m = 64 or 128 message buffers
FCNnDNBMRX1: <FCNn_base> + 1 $00D0_H$
Following registers are available only with m = 128 message buffers:
FCNnDNBMRX2: <FCNn_base> + 1 $00E0_H$
FCNnDNBMRX3: <FCNn_base> + 1 $00F0_H$

**Initial Value** $0000\ 0000_H$

FCNnDNBMRX0:

| 31 ... 0 |
|---|
| FCNnDNBMSSDN[31:00] |

FCNnDNBMRX1 (for m = 64 or 128 message buffers only):

| 31 ... 0 |
|---|
| FCNnDNBMSSDN[63:32] |

FCNnDNBMRX2 (for m = 128 message buffers only):

| 31 ... 0 |
|---|
| FCNnDNBMSSDN[95:64] |

FCNnDNBMRX3 (for m = 128 message buffers only):

| 31 ... 0 |
|---|
| FCNnDNBMSSDN[127:96] |

| FCNnDNBMSSDN[31:0] | Message buffer data new bit |
|---|---|
| 0 | No remote or data frame has been stored into the message buffer. |
| 1 | A remote or data frame has been stored into the message buffer. |

## 18.7.2 CAN channel registers

### (1) FCNnCMMKCTLaH - FCNn channel mask control register

These registers are used to extend the number of receivable messages into the same message buffer by masking part of the identifier (ID) comparison of a message and invalidating the ID of the masked part.

Two 16-bit registers FCNnCMMKCTLaH (a = 01 to 16) can also be accessed via a single 32-bit access to the registers FCNnCMMKCTLaW (a = 01, 03, 05, 07, 09, 11, 13, 15).

**Access** The FCNnCMMKCTLaH registers can be read/written in 16-bit units.
The FCNnCMMKCTLaW registers can be read/written in 32-bit units.

**Address** FCnCMMKCTL01H: <FCNn_base> + $0\ 8300_H$
FCnCMMKCTL02H: <FCNn_base> + $0\ 8308_H$
FCnCMMKCTL03H: <FCNn_base> + $0\ 8310_H$
FCnCMMKCTL04H: <FCNn_base> + $0\ 8318_H$
FCnCMMKCTL05H: <FCNn_base> + $0\ 8320_H$
FCnCMMKCTL06H: <FCNn_base> + $0\ 8328_H$
FCnCMMKCTL07H: <FCNn_base> + $0\ 8330_H$
FCnCMMKCTL08H: <FCNn_base> + $0\ 8338_H$
FCnCMMKCTL09H: <FCNn_base> + $0\ 8340_H$
FCnCMMKCTL10H: <FCNn_base> + $0\ 8348_H$
FCnCMMKCTL11H: <FCNn_base> + $0\ 8350_H$
FCnCMMKCTL12H: <FCNn_base> + $0\ 8358_H$
FCnCMMKCTL13H: <FCNn_base> + $0\ 8360_H$
FCnCMMKCTL14H: <FCNn_base> + $0\ 8368_H$
FCnCMMKCTL15H: <FCNn_base> + $0\ 8370_H$
FCnCMMKCTL16H: <FCNn_base> + $0\ 8378_H$

FCnCMMKCTL01W: <FCNn_base> + $1\ 0300_H$
FCnCMMKCTL03W: <FCNn_base> + $1\ 0310_H$
FCnCMMKCTL05W: <FCNn_base> + $1\ 0320_H$
FCnCMMKCTL07W: <FCNn_base> + $1\ 0330_H$
FCnCMMKCTL09W: <FCNn_base> + $1\ 0340_H$
FCnCMMKCTL11W: <FCNn_base> + $1\ 0350_H$
FCnCMMKCTL13W: <FCNn_base> + $1\ 0360_H$
FCnCMMKCTL15W: <FCNn_base> + $1\ 0370_H$

**Initial Value** $0000_H$

**(a) FCNnCMMKCTLaH (a = 01, 03, 05, 07, 09, 11, 13, 15)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnCMMKSSID[15:8] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnCMMKSSID[7:0] | | | | | | | |

**(b) FCNnCMMKCTLaH (a = 02, 04, 06, 08, 10, 12, 14, 16)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | FCNnCMMKSSID[28:24] | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnCMMKSSID[23:16] | | | | | | | |

**(c) FCNnCMMKCTLaW (a = 01, 03, 05, 07, 09, 11, 13, 15)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | FCNnCMMKSSID[28:24] | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| FCNnCMMKSSID[23:16] | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnCMMKSSID[15:8] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnCMMKSSID[7:0] | | | | | | | |

| FCNnCMMKSSID[i] [a] | Mask pattern setting of ID bit |
|---|---|
| 0 | The ID bit **i** of the message buffer m set by FCNnMmSSID[**i**] are compared with the ID bits of the received message frame. |
| 1 | The ID bit **i** of the message buffer m set by FCNnMmSSID[**i**] are not compared with the ID bits of the received message frame (they are masked). |

a) i = [28:0]

**Note** Masking is always defined by an ID length of 29 bits. If a mask is assigned to a message with a standard ID, FCNnCMMKSSID[17:0] are ignored. Therefore, only FCNnCMMKSSID[28:18] of the received ID are masked. The same mask can be used for both the standard and extended IDs.

### (2) FCNnCMCLCTL - FCNn channel control register

This register is used to control the operation mode of the CAN Controller.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8240_H$

**Initial Value** $0000_H$

#### (a) FCNnCMCLCTL read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMCLSSRS | FCNnCMCLSSTS |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnCMCLERCF | FCNnCMCLALBF | FCNnCMCLVALF | FCNnCMCLMDPF[1:0] | | FCNnCMCLMDOF[2:0] | | |

| FCNnCMCLSSRS | Reception status bit |
|---|---|
| 0 | Reception is stopped. |
| 1 | Reception is in progress. |

**Notes**

1. FCNnCMCLSSRS is set to 1 under the following conditions (timing)
   - The SOF bit of a receive frame is detected
   - On occurrence of arbitration loss during a transmit frame
2. FCNnCMCLSSRS is cleared to 0 under the following conditions (timing)
   - When a recessive level is detected at the second bit of the interframe space
   - On transition to the initialization mode at the first bit of the interframe space

| FCNnCMCLSSTS | Transmission status bit |
|---|---|
| 0 | Transmission is stopped. |
| 1 | Transmission is in progress. |

**Notes**

1. FCNnCMCLSSTS is set to 1 under the following conditions (timing)
   - The SOF bit of a transmit frame is detected
2. FCNnCMCLSSTS is cleared to 0 under the following conditions (timing)
   - During transition to bus-off state
   - On occurrence of arbitration loss in transmit frame
   - On detection of recessive level at the second bit of the interframe space
   - On transition to the initialization mode at the first bit of the interframe space

| FCNnCMCLERCF | Error counter clear bit |
|---|---|
| 0 | The FCNnCMERCNT and FCNnCMLCSTR registers are not cleared in the initialization mode. |
| 1 | The FCNnCMERCNT and FCNnCMLCSTR registers are cleared in the initialization mode. |

**Caution** The error counter register FCNnCMERCNT and the information register FCNnCMINSTR are cleared by FCNnCMCLERCF bit only following conditions:

- During Bus-Off state, while in initialization mode.
- After starting the CAN Controller (FCNnGMCLPWOM is set at FCNnGMCLPWOM=0), while in initialization mode.
- After aborting all transmit request according to *Figure 18-27 “Transmission abort processing (except normal operation mode with ABT)” on page 1011* or *Figure 18-28 “Transmission abort processing (normal operation mode with ABT) - Repeat option for aborted message” on page 1012* , while in initialization mode.

**Notes**

1. When the FCNnCMERCNT and FCNnCMLCSTR registers have been cleared, FCNnCMCLERCF is also cleared to 0 automatically.
2. FCNnCMCLERCF can be set to 1 at the same time as a request to change the initialization mode to an operation mode is made.
3. FCNnCMCLERCF is read-only in the CAN Controller sleep or stop mode.
4. The error counters can also be cleared by shut down or forced shut down of the CAN controller.

| FCNnCMCLALBF | Bit to set operation in case of arbitration loss |
|---|---|
| 0 | Re-transmission is not executed in case of an arbitration loss in the single-shot mode. |
| 1 | Re-transmission is executed in case of an arbitration loss in the single-shot mode. |

**Note** FCNnCMCLALBF is valid only in the single-shot mode.

| FCNnCMCLVALF | Valid receive message frame detection bit |
|---|---|
| 0 | A valid message frame has not been received since FCNnCMCLVALF was last cleared to 0. |
| 1 | A valid message frame has been received since FCNnCMCLVALF was last cleared to 0. |

**Notes**

1. Detection of a valid receive message frame is not dependent upon storage in the receive message buffer (data frame) or transmit message buffer (remote frame).
2. Clear FCNnCMCLVALF (0) before changing the initialization mode to an operation mode.

3. If only two CAN nodes are connected to the CAN bus with one transmitting a message frame in the normal mode and the other in the receive-only mode, FCNnCMCLVALF is not set to 1 before the transmitting node enters the error passive state, because in receive-only mode no acknowledge is generated.
4. To clear FCNnCMCLVALF, set FCNnCMCLCLVL to 1 first and confirm that FCNnCMCLVALF is cleared. If it is not cleared, perform clearing processing again.

| FCNnCMCLMDPF[1:0] | Power save mode |
|---|---|
| $00_B$ | No power save mode is selected. |
| $01_B$ | CAN Controller sleep mode |
| $10_B$ | Setting prohibited |
| $11_B$ | CAN Controller stop mode |

**Cautions**

1. Transition to and from the CAN Controller stop mode must be made via sleep mode. A request for direct transition to and from the stop mode is ignored.
2. The FCNnGMCLSSMO flag of FCNnGMCLCTL must be checked after releasing a power save mode, prior to access the message buffers again.
3. CAN Controller sleep mode requests are kept pending, until cancelled by software or entered on appropriate bus condition (bus idle). Software can check the actual status by reading FCNnCMCLMDPF[1:0].

**Note** In case that the CAN bus is blocked on dominant level, so that the CAN Controller could not synchronize since initialization mode was left, the sleep mode can be reached nevertheless. In this case however, the wake up from sleep mode will happen after the first recessive to dominant edge, after a synchronization was successful. Waking up by software is possible in any case.

| FCNnCMCLMDOF[2:0] | Operation mode |
|---|---|
| $000_B$ | No operation mode is selected<br>(CAN Controller is in the initialization mode). |
| $001_B$ | Normal operation mode |
| $010_B$ | Normal operation mode with automatic block transmission function (normal operation mode with ABT) |
| $011_B$ | Receive-only mode |
| $100_B$ | Single-shot mode |
| $101_B$ | Self-test mode |
| Other than above | Setting prohibited |

**Caution** Transit to initialization mode or power saving modes may take some time. Be sure to verify the success of mode change by reading the values, before proceeding.

**Note** FCNnCMCLMDOF[2:0] are read-only in the CAN Controller sleep or stop mode.

**(b) FCNnCMCLCTL write**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnCMCLSERC | FCNnCMCLSEAL | 0 | FCNnCMSESEPS[1:0] | | FCNnCMCLSEOP[2:0] | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | FCNnCMCLCLAL | FCNnCMCLCLVL | FCNnCMSECLPS[1:0] | | FCNnCMCLCLOP[2:0] | | |

| FCNnCMCLSERC | Setting of FCNnCMCLERCF bit |
|---|---|
| 1 | FCNnCMCLERCF is set to 1. |
| Other than above | FCNnCMCLERCF is not changed. |

| FCNnCMCLSEAL | FCNnCMCLCLAL | Setting of FCNnCMCLALBF bit |
|---|---|---|
| 0 | 1 | FCNnCMCLALBF is cleared to 0. |
| 1 | 0 | FCNnCMCLALBF is set to 1. |
| Other than above | | FCNnCMCLALBF is not changed. |

| FCNnCMCLCLVL | Setting of FCNnCMCLVALF bit |
|---|---|
| 0 | FCNnCMCLVALF is not changed. |
| 1 | FCNnCMCLVALF is cleared to 0. |

| FCNnCMSESEPS0 | FCNnCMSECLPS0 | Setting of FCNnCMCLMDPF0 bit |
|---|---|---|
| 0 | 1 | FCNnCMCLMDPF0 is cleared to 0. |
| 1 | 0 | FCNnCMCLMDPF0 is set to 1. |
| Other than above | | FCNnCMCLMDPF0 is not changed. |

| FCNnCMSESEPS1 | FCNnCMSECLPS1 | Setting of FCNnCMCLMDPF1 bit |
|---|---|---|
| 0 | 1 | FCNnCMCLMDPF1 is cleared to 0. |
| 1 | 0 | FCNnCMCLMDPF1 is set to 1. |
| Other than above | | FCNnCMCLMDPF1 is not changed. |

| FCNnCMCLSEOP0 | FCNnCMCLCLOP0 | Setting of FCNnCMCLMDOF0 bit |
|---|---|---|
| 0 | 1 | FCNnCMCLMDOF0 is cleared to 0. |
| 1 | 0 | FCNnCMCLMDOF0 is set to 1. |
| Other than above | | FCNnCMCLMDOF0 is not changed. |

| FCNnCMCLSEOP1 | FCNnCMCLCLOP1 | Setting of FCNnCMCLMDOF1 bit |
|---|---|---|
| 0 | 1 | FCNnCMCLMDOF1 is cleared to 0. |
| 1 | 0 | FCNnCMCLMDOF1 is set to 1. |
| Other than above | | FCNnCMCLMDOF1 is not changed. |

| FCNnCMCLSEOP2 | FCNnCMCLCLOP2 | Setting of FCNnCMCLMDOF2 bit |
|---|---|---|
| 0 | 1 | FCNnCMCLMDOF2 is cleared to 0. |
| 1 | 0 | FCNnCMCLMDOF2 is set to 1. |
| Other than above | | FCNnCMCLMDOF2 is not changed. |

**Caution** When setting initialization mode while reception is ongoing in an operation mode, a last reception may occur, which sets the data new flag of a message box. However, the transition back to an operation mode also clears the Receive History List. Therefore, reaching the initialization mode must be confirmed by software by reading back the operation mode. Before restarting an operation mode, all set data new flags of all active receive message boxes must be cleared, before activating an operation mode again.

### (3) FCNnCMLCSTR - FCNn channel last error information register

This register provides the error information of the CAN protocol.

**Access** This register can be read/written in 8-bit units.

**Address** <FCNn_base> + $0248_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | FCN0CMLCSSLC[2:0] | | |

**Notes**

1. The contents of the FCNnCMLCSTR register are not cleared when the CAN Controller changes from an operation mode to the initialization mode.
2. If an attempt is made to write a value other than $00_H$ to the FCNnCMLCSTR register by software, the access is ignored.

| FCN0CMLCSSLC[2:0] | Last CAN protocol error information |
|---|---|
| $000_B$ | No error |
| $001_B$ | Stuff error |
| $010_B$ | Form error |
| $011_B$ | ACK error |
| $100_B$ | Bit error. (The CAN Controller tried to transmit a recessive-level bit as part of a transmit message (except the arbitration field), but the value on the CAN bus is a dominant-level bit.) |
| $101_B$ | Bit error. (The CAN Controller tried to transmit a dominant-level bit as part of a transmit message, ACK bit, error frame, or overload frame, but the value on the CAN bus is a recessive-level bit.) |
| $110_B$ | CRC error |
| $111_B$ | Undefined |

### (4) FCNnCMINSTR - FCNn channel information register

This register indicates the status of the CAN Controller.

**Access** This register is read-only in 8-bit units.

**Address** <FCNn_base> + 0 024C$_H$

**Initial Value** 00$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | FCNnCM INBOFF | FCNnCM INSSTE[1:0] | | FCNnCM INSSRE[1:0] | |

| FCNnCMINBOFF | Bus-off state bit |
|---|---|
| 0 | Not bus-off state (transmit error counter ≤ 255). (The value of the transmit counter is less than 256.) |
| 1 | Bus-off state (transmit error counter > 255). (The value of the transmit counter is 256 or more.) |

| FCNnCMINSSTE[1:0] | Transmission error counter status bit |
|---|---|
| 00$_B$ | The value of the transmission error counter is less than that of the warning level (< 96). |
| 01$_B$ | The value of the transmission error counter is in the range of the warning level (96 to 127). |
| 10$_B$ | Undefined |
| 11$_B$ | The value of the transmission error counter is in the range of the error passive or bus-off status (≥ 128). |

| FCNnCMINSSRE[1:0] | Reception error counter status bit |
|---|---|
| 00$_B$ | The value of the reception error counter is less than that of the warning level (< 96). |
| 01$_B$ | The value of the reception error counter is in the range of the warning level (96 to 127). |
| 10$_B$ | Undefined |
| 11$_B$ | The value of the reception error counter is in the error passive range (≥ 128). |

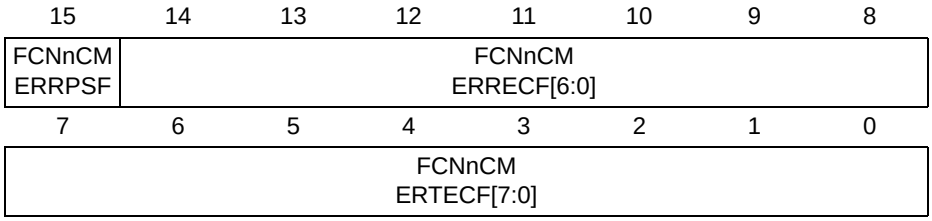
### (5) FCNnCMERCNT - FCNn channel error counter register

This register indicates the count value of the transmission/reception error counter.

**Access** This register is read-only in 16-bit units.

**Address** <FCNn_base> + $0\ 8250_H$

**Initial Value** $0000_H$




| FCNnCMERRPSF | Reception error passive status bit |
|---|---|
| 0 | The reception error counter is not in the error passive range (< 128) |
| 1 | The reception error counter is in the error passive range (≥ 128) |

| FCNnCMERRECF[6:0] | Reception error counter bit |
|---|---|
| 0 to 127 | Number of reception errors. These bits reflect the status of the reception error counter. The number of errors is defined by the CAN protocol. |

**Note** FCNnCMERRECF[6:0] are invalid in the reception error passive state (FCNnCMINSTR.FCNnCMINSSRE[1:0] = $11_B$).

| FCNnCMERTECF[7:0] | Transmission error counter bit |
|---|---|
| 0 to 255 | Number of transmission errors. These bits reflect the status of the transmission error counter. The number of errors is defined by the CAN protocol. |

**Note** FCNnCMERTECF[7:0] are invalid in the bus-off state (FCNnCMINSTR.FCNnCMINBOFF = 1).

### (6) FCNnCMIECTL - FCNn channel interrupt enable register

This register is used to enable or disable the interrupts of the CAN Controller.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8258_H$

**Initial Value** $0000_H$

#### (a) FCNnCMIECTL read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | FCNnCMIEINTF[6:0] | | | | | | |

| FCNnCMIEINTF[6:0] | CAN Controller interrupt enable bit |
|---|---|
| 0 | Output of the interrupt corresponding to interrupt status register FCNnCMISCTL is disabled. |
| 1 | Output of the interrupt corresponding to interrupt status register FCNnCMISCTL is enabled. |

#### (b) FCNnCMIECTL write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | FCNnCMIESEIE[6:0] | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | FCNnCMIECLIE[6:0] | | | | | | |

| FCNnCMIESEIE[6:0] | FCNnCMIECLIE[6:0] | Setting of FCNnCMIEINTF[6:0] bit |
|---|---|---|
| 0 | 1 | FCNnCMIEINTF[6:0] bit is cleared to 0. |
| 1 | 0 | FCNnCMIEINTF[6:0] bit is set to 1. |
| Other than above | | FCNnCMIEINTF[6:0] bit is not changed. |

### (7) FCNnCMISCTL - FCNn channel interrupt status register

This register indicates the interrupt status of the CAN Controller.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8260_H$

**Initial Value** $0000_H$

#### (a) FCNnCMISCTL read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | FCNnCMISITSF[6:0] | | | | | | |

| FCNnCMISITSF[6:0] | CAN Controller interrupt status bit |
|---|---|
| 0 | No related interrupt source event is pending |
| 1 | A related interrupt source event is pending |

| Interrupt status bit | Related interrupt source event |
|---|---|
| FCNnCMISITSF6 | CAN Controller transmission abort interrupt status bit |
| FCNnCMISITSF5 | Wakeup interrupt from CAN Controller sleep mode [a] |
| FCNnCMISITSF4 | Arbitration loss interrupt |
| FCNnCMISITSF3 | CAN protocol error interrupt |
| FCNnCMISITSF2 | CAN error status interrupt |
| FCNnCMISITSF1 | Interrupt on completion of reception of valid message frame to message buffer m |
| FCNnCMISITSF0 | Interrupt on normal completion of transmission of message frame from message buffer m |

a) FCNnCMISITSF5 is set only when the CAN Controller is woken up from the CAN Controller sleep mode by a CAN bus operation. It is not set when the sleep mode has been released by software.

#### (b) FCNnCMISCTL write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | FCNnCMISITSF[6:0] | | | | | | |

| FCNnCMISITSF[6:0] | Clearing of FCNnCMISITSF[6:0] |
|---|---|
| 0 | FCNnCMISITSF[6:0] bits are not changed |
| 1 | FCNnCMISITSF[6:0] bits are cleared to 0 |

**Caution** Clear the status bit of this register by software, when the confirmation of each status is necessary in the interrupt processing, because these bits are not cleared automatically.

### (8) FCNnCMBRPRS - FCNn channel bit rate prescaler register

This register is used to select the CAN protocol layer clock ($f_{TQ}$). The communication baudrate is set to the FCNnCMBTCTL register.

**Access** This register can be read/written in 8-bit units.

**Address** <FCNn_base> + $0\ 0268_H$

**Initial Value** $FF_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnCMBRPRS[7:0] | | | | | | | |

| FCNnCMBRPRS | CAN protocol layer clock ($f_{TQ}$) |
|---|---|
| 0 | $f_{CANCHN}/1$ |
| 1 | $f_{CANCHN}/2$ |
| n | $f_{CANCHN}/(n+1)$ |
| : | : |
| 255 | $f_{CANCHN}/256$ (default value) |

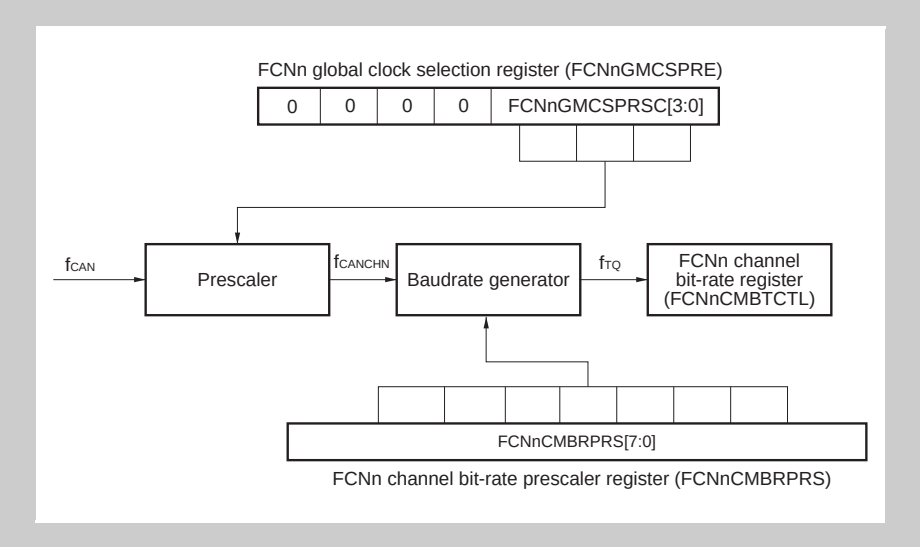



**Figure 18-4 CAN Controller clocks**

**Note** $f_{CAN}$: clock supplied to CAN Controller
$f_{CANCHN}$: CAN channel clock
$f_{TQ}$: CAN protocol layer clock

**Cautions**

1. FCNnCMBRPRS can be write-accessed only in the initialization mode.
2. Setting of the channel clock, CAN baudrate and time quantas (TQs) per data bit time (DBT) is allowed only in certain combinations.
   Refer to the section *"CAN baudrate and time quanta"* above in this chapter.

### (9) FCNnCMBTCTL - FCNn channel bit rate register

This register is used to control the data bit time of the communication baudrate.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8270_H$

**Initial Value** $0370F_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | FCNnCM BTJWLG[1:0] | | 0 | FCNnCM BTS2LG[2:0] | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | FCNnCMBTS1LG[3:0] | | | |

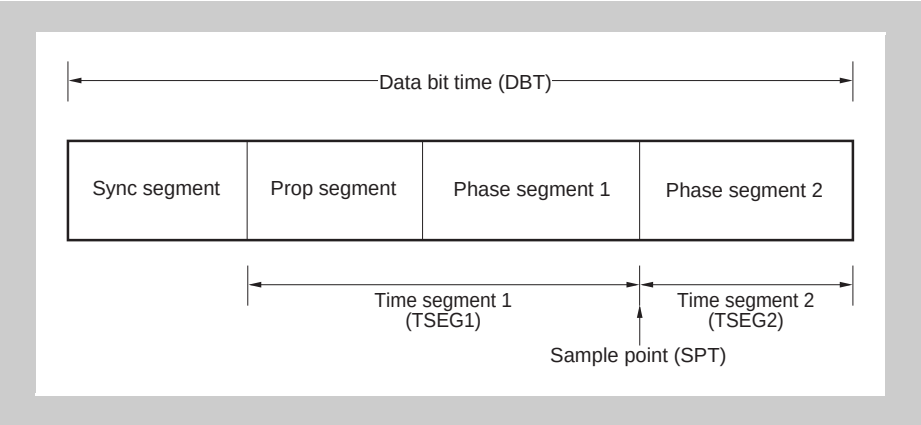



**Figure 18-5 Data bit time**

| FCNnCMBTJWLG[1:0] | Length of synchronization jump width (SJW) |
|---|---|
| $00_B$ | $1T_Q$ |
| $01_B$ | $2T_Q$ |
| $10_B$ | $3T_Q$ |
| $11_B$ | $4T_Q$ (default value) |

| FCNnCMBTS2LG[2:0] | Length of time segment 2 (TSEG2) |
|---|---|
| $000_B$ | $1T_Q$ |
| $001_B$ | $2T_Q$ |
| $010_B$ | $3T_Q$ |
| $011_B$ | $4T_Q$ |
| $100_B$ | $5T_Q$ |
| $101_B$ | $6T_Q$ |
| $110_B$ | $7T_Q$ |
| $111_B$ | $8T_Q$ (default value) |

| FCNnCMBTS1LG[3:0] | Length of time segment 1(TSEG1) |
|---|---|
| $0000_B$ | Setting prohibited |
| $0001_B$ | 2 $T_Q$[a] |
| $0010_B$ | 3 $T_Q$[a] |
| $0011_B$ | 4 $T_Q$ |
| $0100_B$ | 5 $T_Q$ |
| $0101_B$ | 6 $T_Q$ |
| $0110_B$ | 7 $T_Q$ |
| $0111_B$ | 8 $T_Q$ |
| $1000_B$ | 9 $T_Q$ |
| $1001_B$ | 10 $T_Q$ |
| $1010_B$ | 11 $T_Q$ |
| $1011_B$ | 12 $T_Q$ |
| $1100_B$ | 13 $T_Q$ |
| $1101_B$ | 14 $T_Q$ |
| $110_B$ | 15 $T_Q$ |
| $1111_B$ | 16 $T_Q$ (default value) |

a) This setting must not be made when the FCNnCMBRPRS register = $00_H$

**Note** $T_Q = 1/f_{TQ}$ ($f_{TQ}$: CAN protocol layer clock)

**Caution** Setting of the CAN channel clock, CAN baudrate and time quantas (TQs) per data bit time (DBT) is allowed only in certain combinations.
Refer to the section *"CAN baudrate and time quanta"* above in this chapter.

### (10) FCNnCMLISTR - FCNn channel last in-pointer register

This register indicates the number of the message buffer in which a data frame or a remote frame was last stored.

**Access** This register is read-only in 8-bit units.

**Address** <FCNn_base> + $0\ 0278_H$

**Initial Value** Undefined.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnCMLISSLT[7:0] | | | | | | | |

| FCNnCMLISSLT[7:0] | Last in-pointer register |
|---|---|
| 0 to 31 [a]<br>0 to 63 [b]<br>0 to 127 [c] | When the FCNnCMLISTR register is read, the contents of the element indexed by the last in-pointer (FCNnCMLISSLT[7:0]) of the receive history list are read. These contents indicate the number of the message buffer in which a data frame or a remote frame was last stored. |

a) On 32 message buffer CAN Controller.
b) On 64 message buffer CAN Controller.
c) On 128 message buffer CAN Controller.

**Note** The read value of FCNnCMLISTR is undefined if a data frame or a remote frame has never been stored in the message buffer. If FCNnCMRGRX.FCNnCMRGSSPM is set to 1 after the CAN Controller has changed from the initialization mode to an operation mode, therefore, the read value of FCNnCMLISTR is undefined.

### (11) FCNnCMRGRX - FCNn channel receive history list register

This register is used to read the receive history list (RHL).

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8280_H$

**Initial Value** $xx02_H$

#### (a) FCNnCMRGRX read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnCMRGSSPT[7:0] | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMRGSSPM | FCNnCMRGRVFF |

| FCNnCMRGSSPT[7:0] | Receive history list read pointer |
|---|---|
| 0 to 31 [a]<br>0 to 63 [b]<br>0 to 127 [c] | When FCNnCMRGRX is read, the contents of the element indexed by the receive history list get pointer (FCNnCMRGRX.FCNnCMRGSSPT) of the receive history list are read. These contents indicate the number of the message buffer in which a data frame or a remote frame has been stored. |

a) On 32 message buffer CAN Controller.
b) On 64 message buffer CAN Controller.
c) On 128 message buffer CAN Controller.

| FCNnCMRGSSPM[a] | Receive history list pointer match |
|---|---|
| 0 | The receive history list has at least one message buffer number that has not been read. |
| 1 | The receive history list has no message buffer numbers that have not been read. |

a) The read value of FCNnCMRGSSPT[7:0] is invalid when FCNnCMRGSSPM = 1.

| FCNnCMRGRVFF[a] | Receive history list overflow bit [b] |
|---|---|
| 0 | All the message buffer numbers that have not been read are preserved. All the numbers of the message buffers in which a new data frame or remote frame has been received and stored are recorded to the receive history list (the receive history list has a vacant element). |
| 1 | At least **(i)** entries have been stored since the host processor has serviced the RHL last time (i.e. read FCNnCMRGRX). The first **(i-1)** entries are sequentially stored while the last entry can have been overwritten whenever newly received message is stored, because all buffer numbers are stored at position **(i)**, when FCNnCMRGRVFF is set. Thus the sequence of receptions can not be recovered completely now. |

a) If FCNnCMRGRVFF is set, FCNnCMRGSSPM is no longer cleared on message storage, but FCNnCMRGSSPM is still set, if all entries of FCNnCMRGRX have been read by software.
b) i = 23 on 32 message buffer CAN Controller;
i = 47 on 64 message buffer CAN Controller;
i = 95 on 128 message buffer CAN Controller.

### (b) FCNnCMRGRX write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMRGCLRV |

| FCNnCMRGCLRV | Clearing of FCNnCMRGRVFF bit |
|---|---|
| 0 | FCNnCMRGRVFF bit is not changed. |
| 1 | FCNnCMRGRVFF bit is cleared to 0. |

### (12) FCNnCMLOSTR - FCNn channel last out-pointer register

This register indicates the number of the message buffer, from which a data frame or a remote frame was transmitted last.

**Access** This register is read-only in 8-bit units.

**Address** <FCNn_base> + $0288_H$

**Initial Value** Undefined

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnCMLOSTR[7:0] | | | | | | | |

| FCNnCMLOSTR[7:0] | Last out-pointer of transmit history list |
|---|---|
| 0 to 31 [a]<br>0 to 63 [b]<br>0 to 127 [c] | When the FCNnCMLOSTR register is read, the contents of the element indexed by the last out-pointer (FCNnCMLOSTR[7:0]) of the receive history list are read. These contents indicate the number of the message buffer to which a data frame or a remote frame was transmitted last. |

a) On 32 message buffer CAN Controller.
b) On 64 message buffer CAN Controller.
c) On 128 message buffer CAN Controller.

**Note** The value read from the FCNnCMLOSTR register is undefined if a data frame or remote frame has never been transmitted from a message buffer. If FCNnCMTGTX.FCNnCMTGSSPM is set to 1 after the CAN Controller has changed from the initialization mode to an operation mode, therefore, the read value of the FCNnCMLOSTR register is undefined.

### (13) FCNnCMTGTX - FCNn channel transmit history list register

This register is used to read the transmit history list (THL).

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8290_H$

**Initial Value** $xx02_H$

#### (a) FCNnCMTGTX read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnCMTGSSPT[7:0] | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMTGSSPM | FCNnCMTGTVFF |

| FCNnCMTGSSPT[7:0] | Transmit history list read pointer |
|---|---|
| 0 to 31 [a]<br>0 to 63 [b]<br>0 to 127 [c] | When the FCNnCMTGTX register is read, the contents of the element indexed by the read pointer (FCNnCMTGSSPT[7:0]) of the transmit history list are read. These contents indicate the number of the message buffer to which a data frame or a remote frame was transmitted last. |

a) On 32 message buffer CAN Controller.
b) On 64 message buffer CAN Controller.
c) On 128 message buffer CAN Controller.

| FCNnCMTGSSPM[a] | Transmit history pointer match |
|---|---|
| 0 | The transmit history list has at least one message buffer number that has not been read. |
| 1 | The transmit history list has no message buffer numbers that have not been read. |

a) The read value of FCNnCMTGSSPT[7:0] is invalid when the FCNnCMTGSSPM = 1.

| FCNnCMTGTVFF[a] | Transmit history list overflow bit [b] |
|---|---|
| 0 | All the message buffer numbers that have not been read are preserved. All the numbers of the message buffers to which a new data frame or remote frame has been transmitted are recorded to the transmit history list (the transmit history list has a vacant element). |
| 1 | At least **(i)** entries have been stored since the host processor has serviced the THL last time (i.e. read FCNnCMTGTX). The first **(i-1)** entries are sequentially stored while the last entry can have been overwritten whenever newly received message is stored, because all buffer numbers are stored at position **(i)**, when FCNnCMTGTVFF is set. Thus the sequence of receptions can not be recovered completely now. |

a) If FCNnCMTGTVFF is set, FCNnCMTGSSPM is no longer cleared on message transmission, but FCNnCMTGSSPM is still set, if all entries of FCNnCMTGTX are read by software.

b) i = 7 on 32 message buffer CAN Controller
i = 15 on 64 message buffer CAN Controller;
i = 31 on 128 message buffer CAN Controller.

**Note** Transmission from message buffers ...

- 0 to 8 (for 32 message buffer CAN Controller)
- 0 to 16 (for 64 message buffer CAN Controller)
- 0 to 32 (for 128 message buffer CAN Controller)

... is not recorded to the transmit history list in the normal operation mode with ABT.

### (b) FCNnCMTGTX write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | FCNnCMTGCLTV |

| FCNnCMTGCLTV | Setting of FCNnCMTGTVFF bit |
|---|---|
| 0 | FCNnCMTGTVFF bit is not changed |
| 1 | FCNnCMTGTVFF bit is cleared to 0 |

### (14) FCNnCMTSCTL - FCNn channel time stamp register

This register is used to control the time stamp function.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + $0\ 8298_H$

**Initial Value** $0000_H$

#### (a) FCNnCMTSCTL read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | FCNnCMTSLOKE | FCNnCMTSSELE | FCNnCMTSTSGE |

**Note** The lock function of the time stamp function must not be used when the CAN Controller is in the normal operation mode with ABT.

| FCNnCMTSLOKE | Time stamp lock function enable bit |
|---|---|
| 0 | Time stamp lock function stopped.<br>The TSOUT signal is toggled each time the selected time stamp capture event occurs. |
| 1 | Time stamp lock function enabled.<br>The TSOUT signal is toggled each time the selected time stamp capture event occurs.<br>However, the TSOUT output signal is locked when a data frame has been correctly received to message buffer 0 [a]. |

a) FCNnCMTSTSGE is automatically cleared to 0.

| FCNnCMTSSELE | Time stamp capture event selection bit |
|---|---|
| 0 | The time capture event is SOF. |
| 1 | The time stamp capture event is the last bit of EOF. |

| FCNnCMTSTSGE | TSOUT operation setting bit |
|---|---|
| 0 | TSOUT toggle operation is disabled. |
| 1 | TSOUT toggle operation is enabled. |

**(b) FCNnCMTSCTL write**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | FCNnCMTSSELK | FCNnCMTSSES | FCNnCMTSSETS |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | FCNnCMTSCLLK | FCNnCMTSCLSL | FCNnCMTSCLTS |

| FCNnCMTSSELK | FCNnCMTSCLLK | Setting of FCNnCMTSLOKE bit |
|---|---|---|
| 0 | 1 | FCNnCMTSLOKE is cleared to 0. |
| 1 | 0 | FCNnCMTSLOKE is set to 1. |
| Other than above | | FCNnCMTSLOKE is not changed. |

| FCNnCMTSSES | FCNnCMTSCLSL | Setting of FCNnCMTSSELE bit |
|---|---|---|
| 0 | 1 | FCNnCMTSSELE is cleared to 0. |
| 1 | 0 | FCNnCMTSSELE is set to 1. |
| Other than above | | FCNnCMTSSELE is not changed. |

| FCNnCMTSSETS | FCNnCMTSCLTS | Setting of FCNnCMTSTSGE bit |
|---|---|---|
| 0 | 1 | FCNnCMTSTSGE is cleared to 0. |
| 1 | 0 | FCNnCMTSTSGE is set to 1. |
| Other than above | | FCNnCMTSTSGE is not changed. |

### 18.7.3 Message buffer registers

#### (1) FCNnMmDATxB/H/W, FCNn message data byte registers

These registers are used to store the data of a transmit/receive message.

**Access** The FCNnMmDATxW registers can be read/written in 32-bit units.
The FCNnMmDATxH registers can be read/written in 16-bit units.
The FCNnMmDATxB registers can be read/written in 8-bit units.

**Address** FCNnMmDAT0B: <FCNn_base> + 0 $1000_H$ + m × $40_H$
FCNnMmDAT1B: <FCNn_base> + 0 $1004_H$ + m × $40_H$
FCNnMmDAT2B: <FCNn_base> + 0 $1008_H$ + m × $40_H$
FCNnMmDAT3B: <FCNn_base> + 0 $100C_H$ + m × $40_H$
FCNnMmDAT4B: <FCNn_base> + 0 $1010_H$ + m × $40_H$
FCNnMmDAT5B: <FCNn_base> + 0 $1014_H$ + m × $40_H$
FCNnMmDAT6B: <FCNn_base> + 0 $1018_H$ + m × $40_H$
FCNnMmDAT7B: <FCNn_base> + 0 $101C_H$ + m × $40_H$

FCNnMmDAT0H: <FCNn_base> + 0 $9000_H$ + m × $40_H$
FCNnMmDAT2H: <FCNn_base> + 0 $9008_H$ + m × $40_H$
FCNnMmDAT4H: <FCNn_base> + 0 $9010_H$ + m × $40_H$
FCNnMmDAT6H: <FCNn_base> + 0 $9018_H$ + m × $40_H$

FCNnMmDAT0W: <FCNn_base> + 1 $1000_H$ + m × $40_H$
FCNnMmDAT4W: <FCNn_base> + 1 $1010_H$ + m × $40_H$

**Initial Value** Undefined.

**(a) FCNnCMmDATxB (x = 0 to 7)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSD0[7:0], FCNnMmSSD1[7:0], FCNnMmSSD2[7:0], FCNnMmSSD3[7:0], FCNnMmSSD4[7:0], FCNnMmSSD5[7:0], FCNnMmSSD6[7:0], FCNnMmSSD7[7:0] | | | | | | | |

**(b) FCNnCMmDATyH (y = 0, 2, 4, 6)**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSD0[7:0], FCNnMmSSD2[7:0], FCNnMmSSD4[7:0], FCNnMmSSD6[7:0] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSD1[7:0], FCNnMmSSD3[7:0], FCNnMmSSD5[7:0], FCNnMmSSD7[7:0] | | | | | | | |

**(c) FCNnCMmDATzW (z = 0, 4)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSD0[7:0], FCNnMmSSD4[7:0] | | | | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSD1[7:0], FCNnMmSSD5[7:0] | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSD2[7:0], FCNnMmSSD6[7:0] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSD3[7:0], FCNnMmSSD7[7:0] | | | | | | | |

### (2) FCNnMmDTLGB - FCNn message data length register m

This register is used to set the number of bytes of the data field of a message buffer.

**Access** This register can be read/written in 8-bit units.

**Address** <FCNn_base> + 0 $1020_H$ + m × $40_H$

**Initial Value** Undefined.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | FCNnMmDTLG[3:0] | | | |

| FCNnMmDTLG[3:0] | Data length of transmit/receive message |
|---|---|
| $0000_B$ | 0 bytes |
| $0001_B$ | 1 byte |
| $0010_B$ | 2 bytes |
| $0001_B$ | 3 bytes |
| $0100_B$ | 4 bytes |
| $0101_B$ | 5 bytes |
| $0110_B$ | 6 bytes |
| $0111_B$ | 7 bytes |
| $1000_B$ | 8 bytes |
| $1001_B$ | Setting prohibited<br>(If these bits are set during transmission, 8-byte data is transmitted regardless of the set DLC value when a data frame is transmitted. However, the DLC actually transmitted to the CAN bus is the DLC value set to this register.)Note |
| $1010_B$ | |
| $1001_B$ | |
| $1100_B$ | |
| $1101_B$ | |
| $1110_B$ | |
| $1111_B$ | |

**Note** The data and DLC value actually transmitted to CAN bus are as follows.

| Type of transmit frame | Length of transmit data | DLC transmitted |
|---|---|---|
| Data frame | Number of bytes specified by DLC (However, 8 bytes if DLC ≥ 8) | FCNnMmDTLGB.FCNnMmDTLG[3:0] bits |
| Remote frame | 0 bytes | |

**Cautions**

1. Be sure to set bits 7 to 4 to $0000_B$.
2. Receive data is stored in as many FCNnMmDATxB register as the number of bytes (however, the upper limit is 8) corresponding to DLC of the received frame. The FCNnMmDATxB register in which no data is stored is undefined.
3. On reception, FCNnMmDTLGB is updated according to the received frame.

### (3) FCNnMmSTRB - FCNn message configuration register m

This register is used to specify the type of the message buffer and to set a mask.

**Access** This register can be read/written in 8-bit units.

**Address** <FCNn_base> + 0 $1024_H$ + m × $40_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnMm SSOW | FCNnMm SSMT[3:0] | | | | FCNnMm SSRT | 0 | FCNnMm SSAM |

| FCNnMmSSOW | Overwrite control bit |
|---|---|
| 0 | The message buffer that has already received a data frame[a] is not overwritten by a newly received data frame. The newly received data frame is discarded. |
| 1 | The message buffer that has already received a data frame[a] is overwritten by a newly received data frame. |

a) The “message buffer that has already received a data frame” is a receive message buffer whose FCNnMmCTL.FCNnMmDTNF bit has been set to 1.

**Note** A remote frame is received and stored, regardless of the setting of FCNnMmCTL.FCNnMmSSOW and FCNnMmCTL.FCNnMmDTNF. A remote frame that satisfies the other conditions

- either:
  ID matches exactly, FCNnMmSTRB.FCNnMmSSRT = 0, FCNnMmCTL.FCNnMmTRQF = 0, FCNnMmSSMT = $0000_B$
- or:
  ID matches with or without mask, FCNnMmSTRB.FCNnMmSSRT = 1, FCNnMmSSMT <> $0000_B$

is always received and stored in the corresponding message buffer (interrupt generated, FCNnMmDTNF flag set, FCNnMmDTLGB.FCNnMmDTLG[3:0] updated, and recorded to the receive history list).

| FCNnMmSSRT | Remote frame request bit |
|---|---|
| 0 | Transmit / receive a data frame. |
| 1 | Transmit / receive a remote frame. |

FCNnMmSTRB.FCNnMmSSRT specifies the type of message frame that is transmitted or received from/to a message buffer.

**Notes**

1. If the message buffer is defined as a transmit message buffer, and a remote frame shall be received into it, the FCNnMmSSRT bit must be cleared.
2. Even if a valid remote frame has been received in a transmit message buffer, the FCNnMmSSRT bit of the transmit message buffer that has received the frame remains cleared to 0.

3. Even if a remote frame whose ID matches has been received from the CAN bus, if the FCNnMmSSRT bit of a transmit message buffer is set to 1 (to transmit a remote frame), that remote frame is not stored in this transmit message buffer.
4. If the message buffer is defined as a receive message buffer, the FCNnMmSSRT bit must be set, in order to receive remote frames instead of data frames.

| FCNnMmSSMT[3:0] | Message buffer type setting bit |
|---|---|
| $0000_B$ | Transmit message buffer |
| $0001_B$ | Receive message buffer (no mask setting) |
| $0010_B$ | Receive message buffer (mask 1 set) |
| $0011_B$ | Receive message buffer (mask 2 set) |
| $0100_B$ | Receive message buffer (mask 3 set) |
| $0101_B$ | Receive message buffer (mask 4 set) |
| $0110_B$ | Receive message buffer (mask 5 set) |
| $0111_B$ | Receive message buffer (mask 6 set) |
| $1000_B$ | Receive message buffer (mask 7 set) |
| $1001_B$ | Receive message buffer (mask 8 set) |
| Other than above | Setting prohibited |

**Note** The setting of FCNnMmSSMT is also valid to select masks in conjunction with remote frame reception. To receive remote frames in receive message buffers, the flag FCNnMmSSRT of the message buffer must be set.

| FCNnMmSSAM | Message buffer assignment bit |
|---|---|
| 0 | Message buffer not used. |
| 1 | Message buffer used. |

**Caution** Be sure to write 0 to bits 2 and 1.

### (4) FCNnMmMID0H, FCNnMmMID1H, FCNnMmMID0W - FCNn message ID register m

These registers are used to set an identifier (ID).

**Access** FCNnMmMID0H, FCNnMmMID1H can be read/written in 16-bit units.
FCNnMmMID0W can be read/written in 32-bit units.

**Address** FCNnMmMID0H: <FCNn_base> + $0\ 9028_H$ + m × $40_H$
FCNnMmMID1H: <FCNn_base> + $0\ 9030_H$ + m × $40_H$

FCNnMmMID0W: <FCNn_base> + $1\ 1028_H$ + m × $40_H$

**Initial Value** Undefined.

#### (a) FCNnMmMID0H

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSID[15:8] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSID[7:0] | | | | | | | |

#### (b) FCNnMmMID1H

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSIE | 0 | 0 | FCNnMmSSID[28:24] | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSID[23:16] | | | | | | | |

#### (c) FCNnCMmMID0W

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSIE | 0 | 0 | FCNnMmSSID[28:24] | | | | |

| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSID[23:16] | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSID[15:8] | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FCNnMmSSID[7:0] | | | | | | | |

| FCNnMmSSIE | Format mode specification bit |
|---|---|
| 0 | Standard format mode (FCNnMmSSID[28:18]: 11 bits, FCNnMmSSID[17:0] are not used) |
| 1 | Extended format mode (FCNnMmSSID[28:0]: 29 bits) |

| FCNnMmSSID[28:0] | Message ID |
|---|---|
| FCNnMmSSID[28:18] | Standard ID value of 11 bits (when FCNnMmSSIE = 0) |
| FCNnMmSSID[28:0] | Extended ID value of 29 bits (when FCNnMmSSIE = 1) |

**Cautions**

1. Be sure to write 0 to bits 14 and 13 of FCNnMmMID1H, respectively bits 30 and 29 of FCNnMmMID0W register.
2. Be sure to align the ID value according to the given bit positions into this registers. Note that for standard ID, the ID value must be shifted to fit into FCNnMmSSID[28:18] bit positions.

### (5) FCNnMmCTL - FCNn message control register m

This register is used to control the operation of the message buffer.

**Access** This register can be read/written in 16-bit units.

**Address** <FCNn_base> + 0 $9038_H$ + m × $40_H$

**Initial Value** $0000_H$

#### (a) FCNnMmCTL read

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | FCNnMmMUCF | 0 | 0 | 0 | FCNnMmTCPF | 0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | FCNnMmNHMF | 0 | FCNnMmMOWF | FCNnMmIENF | FCNnMmDTNF | FCNnMmTRQF | FCNnMmRDYF |

| FCNnMmNHMF | History Mask flag [a] |
|---|---|
| 0 | Updates of receive/transmit history list registers FCNnCMRGRX/FCNnCMRGTX are not masked. |
| 1 | Updates of receive/transmit history list registers FCNnCMRGRX/FCNnCMRGTX are masked. |

a) When masked, the history lists are not updated upon reception or transmission completion activity on this message buffer.

| FCNnMmMUCF[a] | Bit indicating that message buffer data is being updated |
|---|---|
| 0 | The CAN Controller is not updating the message buffer (reception and storage). |
| 1 | The CAN Controller is updating the message buffer (reception and storage). |

a) FCNnMmMUCF is undefined until the first reception and storage is performed.

| FCNnMmTCPF[a] | Transmission completely finished flag |
|---|---|
| 0 | Transmission failed.[b c] |
| 1 | Transmission (including ABT) completely finished. |

a) FCNnMmTCPF is cleared if FCNnMmRDYF is changed or FCNnMmTRQF is set.
b) This indicates a successful transmission abort, if this was requested by the application by clearing the FCNnMmTRQF flag.
c) FCNnMmTCPF is not cleared, if FCNnMmTRQF is set by the ABT operation.

| FCNnMmMOWF | Message buffer overwrite status bit |
|---|---|
| 0 | The message buffer is not overwritten by a newly received data or remote frame. |
| 1 | The message buffer is overwritten by a newly received data or remote frame. |

| FCNnMmIENF | Message buffer interrupt request enable bit |
|---|---|
| 0 | Receive message buffer: Valid message reception completion interrupt disabled.<br>Transmit message buffer: Normal message transmission completion interrupt disabled. |
| 1 | Receive message buffer: Valid message reception completion interrupt enabled.<br>Transmit message buffer: Normal message transmission completion interrupt enabled. |

| FCNnMmDTNF | Message buffer data update bit |
|---|---|
| 0 | No new data frame or remote frame has been stored in the message buffer. |
| 1 | A new data frame or remote frame has been stored in the message buffer. |

| FCNnMmTRQF | Message buffer transmission request bit |
|---|---|
| 0 | No message frame transmitting request that is pending or being transmitted is in the message buffer. |
| 1 | The message buffer is holding transmission of a message frame pending or is transmitting a message frame. |

| FCNnMmRDYF | Message buffer ready bit |
|---|---|
| 0 | The message buffer can be written by software. The CAN Controller cannot write to the message buffer. |
| 1 | Writing the message buffer by software is ignored (except a write access to the FCNnMmRDYF, FCNnMmTRQF, FCNnMmDTNF, and FCNnMmMOWF). The CAN Controller can write to the message buffer. |

### (b) FCNnMmCTL write

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | FCNnMmSENH | 0 | 0 | FCNnMmSEIE | FCNnMmSEDN | FCNnMmSETR | FCNnMmSERY |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | FCNnMmCLNH | 0 | FCNnMmCLMW | FCNnMmCLIE | FCNnMmCLDN | FCNnMmCLTR | FCNnMmCLRY |

| FCNnMmSENH | FCNnMmCLNH | Setting of FCNnMmNHMF bit |
|---|---|---|
| 0 | 1 | FCNnMmNHMF is cleared to 0. |
| 1 | 0 | FCNnMmNHMF is set to 1. |
| Other than above | | FCNnMmNHMF is not changed. |

| FCNnMmCLMW | Setting of FCNnMmMOWF bit |
|---|---|
| 0 | FCNnMmMOWF is not changed. |
| 1 | FCNnMmMOWF is cleared to 0. |

| FCNnMmSEIE | FCNnMmCLIE | Setting of FCNnMmIENF bit |
|---|---|---|
| 0 | 1 | IE is cleared to 0. |
| 1 | 0 | IE is set to 1. |
| Other than above | | IE is not changed. |

| FCNnMmSEDN | FCNnMmCLDN | Setting of FCNnMmDTNF bit |
|---|---|---|
| 0 | 1 | FCNnMmDTNF is cleared to 0. |
| 1 | 0 | FCNnMmDTNF is set to 1. |
| Other than above | | FCNnMmDTNF is not changed. |

**Note** If FCNnMmDTNF is cleared by the finish of ID field receiving, the message buffer participates in the search to store the receiving frame.

| FCNnMmSETR | FCNnMmCLTR | Setting of FCNnMmTRQF bit |
|---|---|---|
| 0 | 1 | FCNnMmTRQF is cleared to 0. |
| 1 | 0 | FCNnMmTRQF is set to 1. |
| Other than above | | FCNnMmTRQF is not changed. |

| FCNnMmSERY | FCNnMmCLRY | Setting of FCNnMmRDYF bit |
|---|---|---|
| 0 | 1 | FCNnMmRDYF is cleared to 0. |
| 1 | 0 | FCNnMmRDYF is set to 1. |
| Other than above | | FCNnMmRDYF is not changed. |

**Cautions**

1. Set FCNnMmIENF and FCNnMmRDYF always separately.
2. Do not set FCNnMmDTNF to 1 by software. Be sure to write 0 to bit 10.
3. Do not set FCNnMmTRQF and FCNnMmRDYF to 1 at the same time. Set FCNnMmRDYF = 1 before setting FCNnMmTRQF = 1.
4. Do not clear FCNnMmRDYF to “0” during message transmission. Follow the transmission abort process about clearing FCNnMmRDYF for redefinition of the message buffer.
5. Clearing of FCNnMmRDYF may take some time, depending on activity of the CAN Controller. Repeat the clearing access, until reading of FCNnMmRDYF confirms that the bit is cleared.
6. Be sure that FCNnMmRDYF is cleared before writing to the other message buffer registers, by checking the status of FCNnMmRDYF.

# 18.8 CAN Controller Initialization

## 18.8.1 Initialization of CAN Controller

Before the CAN Controller operation is enabled, the CAN channel clock $T_{CANCH}$ needs to be determined by setting FCNnGMCSPRE.FCNnGMCSPRSC[3:0] by software.
Do not change the setting of the CAN channel clock $T_{CANCH}$ after CAN Controller operation is enabled.

The CAN Controller is enabled by setting FCNnGMCLCTL.FCNnGMCLPWOM.

For the procedure of initializing the CAN Controller, refer to *18.16 “Operation of the CAN Controller” on page 999* .

## 18.8.2 Initialization of message buffer

After the CAN Controller is enabled, the message buffers contain undefined values. A minimum initialization for all the message buffers, even for those not used in the application, is necessary before switching the CAN Controller from the initialization mode to one of the operation modes.

- Clear FCNnMmRDYF, FCNnMmTRQF, and FCNnMmDTNF of the FCNnMmCTL registers to 0.
- Clear all FCNnMmSTRB.FCNnMmSSAM to 0.

## 18.8.3 Redefinition of message buffer

Redefining a message buffer means changing the ID and control information of the message buffer while a message is being received or transmitted, without affecting other transmission/reception operations.

**(1) To redefine message buffer in initialization mode**

Place the CAN Controller in the initialization mode once and then change the ID and control information of the message buffer in the initialization mode. After changing the ID and control information, set the CAN Controller to an operation mode.

**(2) To redefine message buffer during reception**

Perform redefinition as shown in *Figure 18-20 “Message buffer redefinition during reception”*.

**(3) To redefine message buffer during transmission**

To rewrite the contents of a transmit message buffer to which a transmission request has been set, perform transmission abort processing (see and *2 “Transmission abort process except for ABT transmission in normal operation mode with automatic block transmission (ABT)” on page 980* ). Confirm that transmission has been aborted or completed, and then redefine the message buffer. After redefining the transmit message buffer, set a transmission request using the procedure described below. When setting a transmission request to a message buffer that has been redefined without aborting the transmission in progress, however, the 1-bit wait time is not necessary.

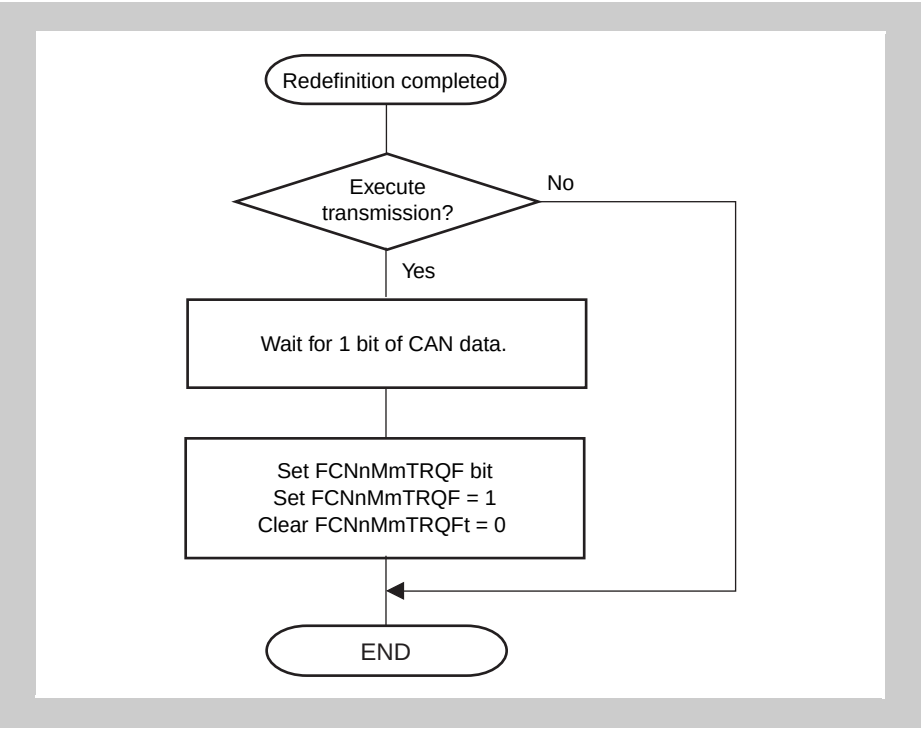



**Figure 18-6 Setting transmission request (FCNnMmCTL.FCNnMmTRQF) to transmit message buffer after redefinition**

**Cautions**

1. When a message is received, reception filtering is performed in accordance with the ID and mask set to each receive message buffer. If the procedure in *Figure 18-20 “Message buffer redefinition during reception” on page 1003* is not observed, the contents of the message buffer after it has been redefined may contradict the result of reception (result of reception filtering). If this happens, check that the ID and IDE received first and stored in the message buffer following redefinition are those stored after the message buffer has been redefined. If no ID and IDE are stored after redefinition, redefine the message buffer again.
2. When a message is transmitted, the transmission priority is checked in accordance with the ID, IDE, and FCNnMmSTRB.FCNnMmSSRT set to each transmit message buffer to which a transmission request was set. The transmit message buffer having the highest priority is selected for transmission. If the procedure in *Figure 18-6 “Setting transmission request (FCNnMmCTL.FCNnMmTRQF) to transmit message buffer after redefinition” on page 963* is not observed, a message with an ID not having the highest priority may be transmitted after redefinition.

### 18.8.4 Transition from initialization mode to operation mode

The CAN Controller can be switched to the following operation modes.

- Normal operation mode
- Normal operation mode with ABT
- Receive-only mode
- Single-shot mode
- Self-test mode

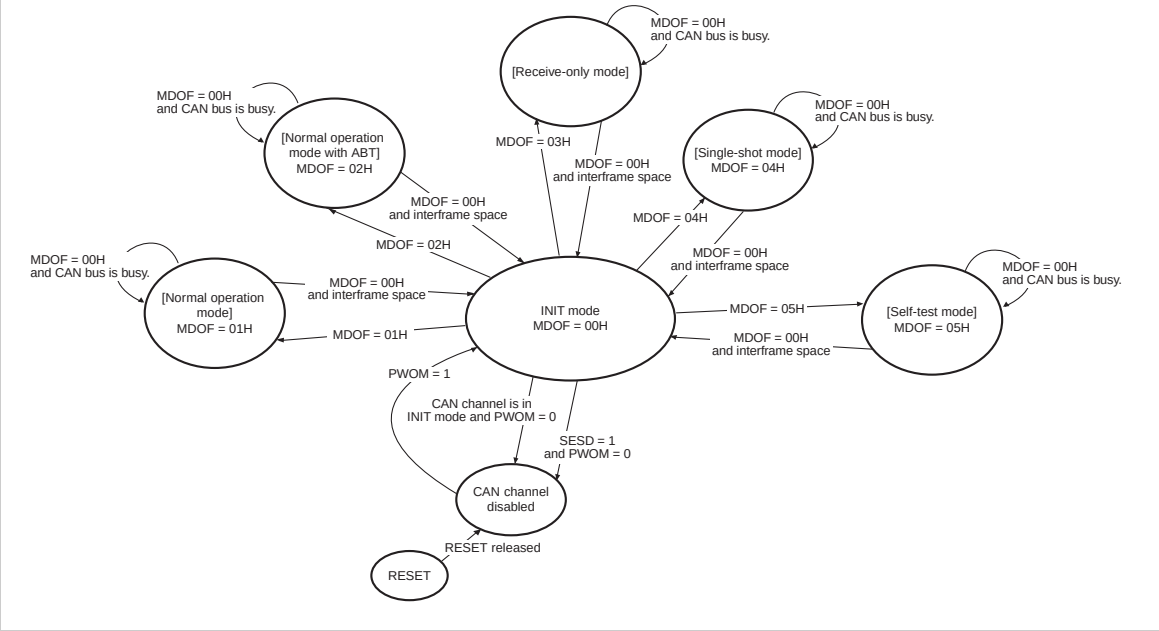



**Figure 18-7 Transition to operation modes**

**Note** In the figure above following abbreviations are used:

- MDOF = FCNnCMCLCTL.FCNnCMCLMDOF[2:0]
- PWOM = FCNnGMCLCTL.FCNnGMCLPWOM
- SESD = FCNnGMCLCTL.FCNnGMCLSESD

The transition from the initialization mode to an operation mode is controlled by the bit string FCNnCM.FCNnCMCLMDOF[2:0].

Changing from one operation mode into another requires shifting to the initialization mode in between. Do not change one operation mode to another directly; otherwise the operation will not be guaranteed.

Requests for transition from an operation mode to the initialization mode are held pending when the CAN bus is not in the interframe space (i.e., frame reception or transmission is in progress), and the CAN Controller enters the initialization mode at the first bit in the interframe space (the values of the FCNnCMCLCTL.FCNnCMCLMDOF[2:0] are changed to $000_{B)}$. After issuing a request to change the mode to the initialization mode, read FCNnCMCLCTL.FCNnCMCLMDOF[2:0] until their value becomes $000_B$ to confirm that the channel has entered the initialization mode (see *Figure 18-17 “Re-initialization without Software Reset function” on page 1000* ).

## 18.9 Message Reception

### 18.9.1 Message reception

In all the operation modes, the complete message buffer area is analyzed to find a suitable buffer to store a newly received message. All message buffers satisfying the following conditions are included in that evaluation (RX-search process).

- Used as a message buffer
  (FCNnMmSTRB.FCNnMmSSAM = 1.)
- Set as a receive message buffer
  (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0001_B$ to $1001_B$.)
- Ready for reception
  (FCNnMmCTL.FCNnMmRDYF = 1.)

When two or more message buffers of the CAN Controller are found to be able to receive a message, the message is stored according to the priority explained below. The message is always stored in the message buffer with the highest priority, not in a message buffer with a low priority. For example, when an unmasked receive message buffer and a receive message buffer linked to mask 1 have the same ID, the received message is not stored in the message buffer linked to mask 1, even if that message buffer has not received a message and a message has already been received in the unmasked receive message buffer. In other words, when a condition has been set in two or more message buffers with different priorities, the message buffer with the highest priority always stores the message; the message is not stored in message buffers with a lower priority. This also applies when the message buffer with the highest priority is unable to store a message (i.e., when FCNnMmCTL.FCNnMmDTNF = 1 indicating that a message has already been received, but rewriting is disabled because FCNnMmSTRB.FCNnMmSSOW = 0). In this case, the message is not actually stored in the candidate message buffer with the highest priority, but neither is it stored in a message buffer with a lower priority.

**Table 18-20 MBRB priorities**

| Priority | Storing condition if same ID is set | |
|---|---|---|
| 1 (high) | Unmasked message buffer | FCNnMmDTNF = 0 |
| | | FCNnMmDTNF = 1 and FCNnMmSSOW = 1 |
| 2 | Message buffer linked to mask 1 | FCNnMmDTNF = 0 |
| | | FCNnMmDTNF = 1 and FCNnMmSSOW = 1 |
| 3 | Message buffer linked to mask 2 | FCNnMmDTNF = 0 |
| | | FCNnMmDTNF = 1 and FCNnMmSSOW = 1 |
| ... | ... | |
| 9 (low) | Message buffer linked to mask 8 | FCNnMmDTNF = 0 |
| | | FCNnMmDTNF = 1 and FCNnMmSSOWt = 1 |

### 18.9.2 Receive data read

To keep data consistency when reading CAN Controller message buffers, perform the data reading according to *Figure 18-34 “Reception via interrupt (using FCNnCMLISTR register)” on page 1019* to *Figure 18-37 “Reception via software polling” on page 1023* .

During message reception, the CAN Controller sets FCNnMmCTL.FCNnMmDTNF two times: at the beginning of the storage process of data to the message buffer, and again at the end of this storage process. During this storage process, FCNnMmCTL.FCNnMmMUCF of the message buffer is set (refer to *Figure 18-8 “FCNnMmCTL.FCNnMmDTNF and FCNnMmCTL.FCNnMmMUCF bit setting period (for standard ID format)”*).

The receive history list is also updated just before the storage process. In addition, during storage process (FCNnMmCTL.FCNnMmMUCF = 1), FCNnMmCTL.FCNnMmRDYF of the message buffer is locked to avoid any coincidental data write by CPU. Note that the storage process may be disturbed (delayed) when the CPU accesses the message buffer.

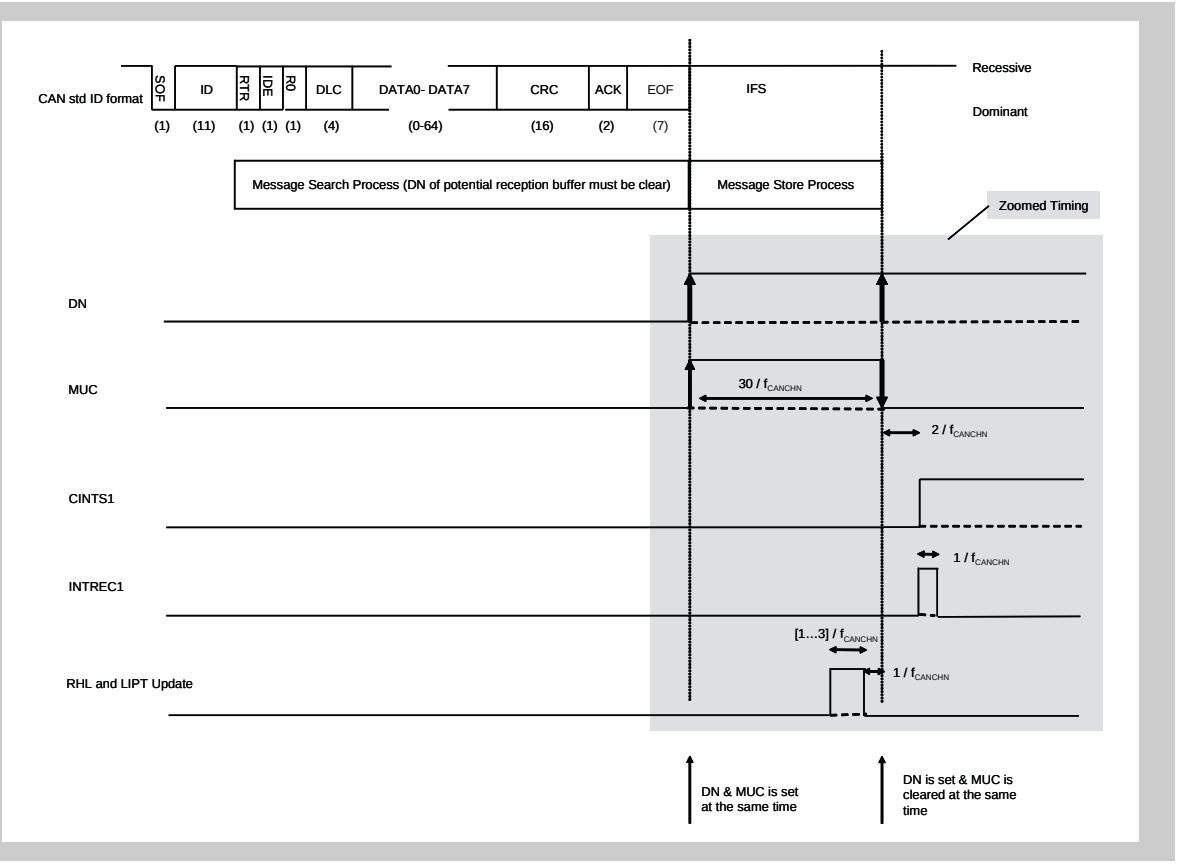



**Figure 18-8 FCNnMmCTL.FCNnMmDTNF and FCNnMmCTL.FCNnMmMUCF bit setting period (for standard ID format)**

**Note** If a message shall be stored in a message buffer, the FCNnMmDTNF bit of this buffer must be cleared before the Message Search Process is started, i.e., right after the ID of the frame is on the bus. In worst case, this happens 15 CAN bits after EOF of the previous frame. Consider to use more than one Message Buffer for reception of a frame, if CAN frames are appearing back-to-back on the bus and none shall be lost.

### 18.9.3 Receive history list function

The receive history list (RHL) function records in the receive history list the number of the receive message buffer in which each data frame or remote frame was received and stored. The RHL consists of storage elements equivalent to up to 23 messages (on 32 message buffer CAN Controller), 47 messages (on 64 message buffer CAN Controller) or up to 95 messages (on 128 message buffer CAN Controller), the last in-message pointer FCNnCMLISSLR[7:0] with the corresponding FCNnCMLISTR register and the receive history list get pointer FCNnCMRGSSPT with the corresponding FCNnCMRGRX register.

The RHL is undefined immediately after the transition of the CAN Controller from the initialization mode to one of the operation modes.

The FCNnCMLISTR register holds the contents of the RHL element indicated by the value of the FCNnCMLISTR.FCNnCMLISSLR[7:0] pointer minus 1. By reading the FCNnCMLISTR register, therefore, the number of the message buffer that received and stored a data frame or remote frame first can be checked. The FCNnCMLISSLR[7:0] pointer is utilized as a write pointer that indicates to what part of the RHL a message buffer number is recorded. Any time a data frame or remote frame is received and stored, the corresponding message buffer number is recorded to the RHL element indicated by the FCNnCMLISSLR[7:0] pointer. Each time recording to the RHL has been completed, the FCNnCMLISSLR[7:0] pointer is automatically incremented. In this way, the number of the message buffer that has received and stored a frame will be recorded chronologically.

For message buffers, where the flag FCNnMmCTL.FCNnMmNHMF is set, no entry in the history lists is recorded.

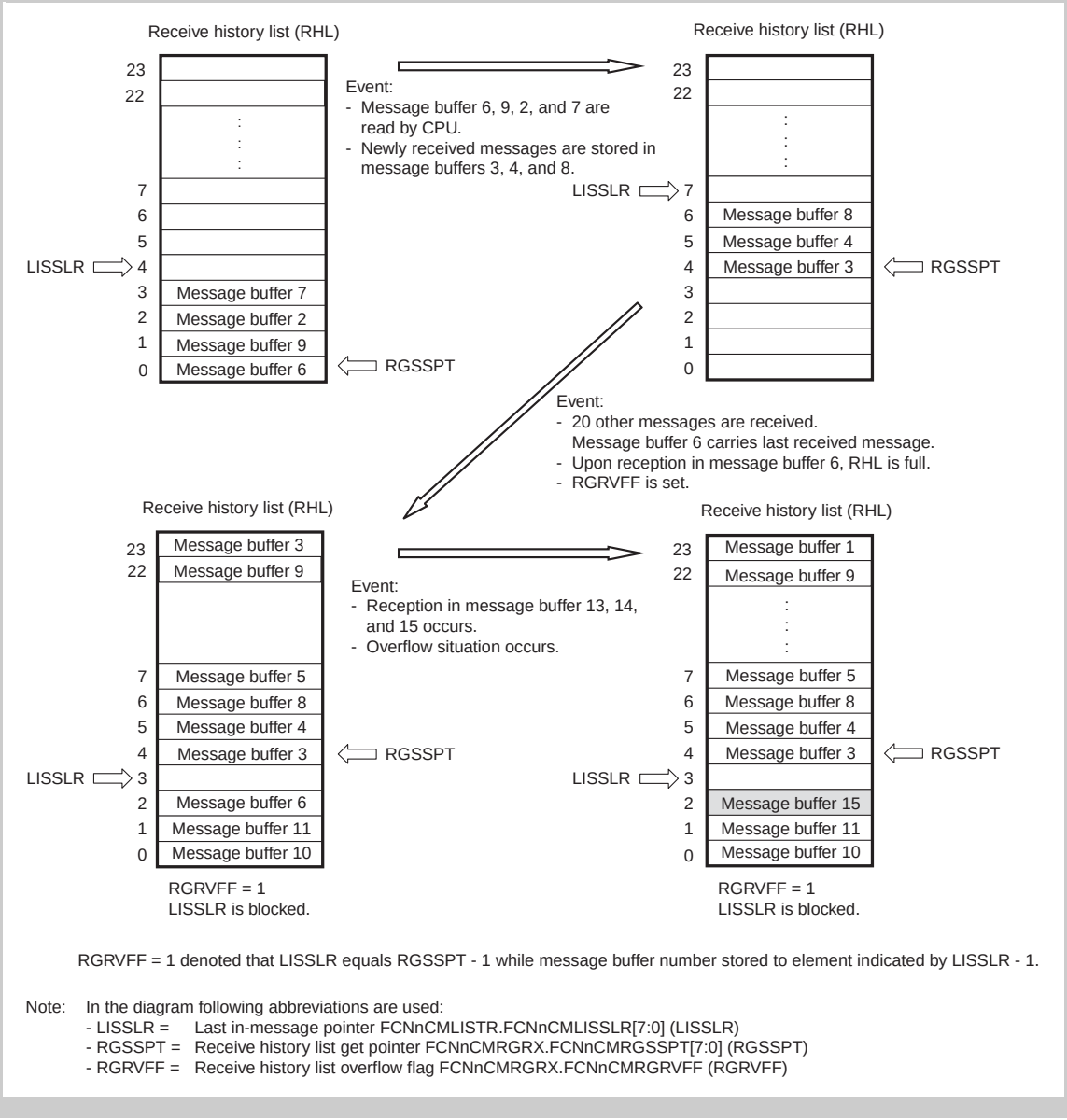
The FCNnCMRGRX.FCNnCMRGSSPT pointer is utilized as a read pointer that reads a recorded message buffer number from the RHL. This pointer indicates the first RHL element that the CPU has not read yet. By reading the FCNnCMRGRX register by software, the number of a message buffer that has received and stored a data frame or remote frame can be read. Each time a message buffer number is read from the FCNnCMRGRX register, the FCNnCMRGSSPT pointer is automatically incremented.

If the value of the FCNnCMRGRX.FCNnCMRGSSPT pointer matches the value of the FCNnCMLISTR.FCNnCMLISSLR[7:0] pointer, FCNnCMRGRX.FCNnCMRGSSPM (receive history list pointer match) is set to 1. This indicates that no message buffer number that has not been read remains in the RHL. If a new message buffer number is recorded, the FCNnCMLISSLR[7:0] pointer is incremented and because its value no longer matches the value of the FCNnCMRGSSPT pointer, FCNnCMRGSSPM is cleared. In other words, the numbers of the unread message buffers exist in the RHL.

If the FCNnCMLISTR.FCNnCMLISSLR[7:0] pointer is incremented and matches the value of the FCNnCMRGRX.FCNnCMRGSSPT pointer minus 1, FCNnCMRGRX.FCNnCMRGRVFF (receive history list overflow) is set to 1. This indicates that the RHL is full of numbers of message buffers that have not been read. When further message reception and storing occur, the last recorded message buffer number is overwritten by the number of the message buffer that received and stored the newly received message. In this case, after FCNnCMRGRVFF has been set (1), the recorded message buffer numbers in the RHL do not completely reflect the chronological order. However messages itself are not lost and can be located by CPU search in message buffer memory with the help of FCNnMmCTL.FCNnMmDTNF, or by reading the global registers FCNnDNBMRX[3:0].

**Caution** If the history list is in the overflow condition (FCNnCMRGRX.FCNnCMRGRVFF is set), reading the history list contents is still possible, until the history list is empty (indicated by FCNnCMRGRX.FCNnCMRGSSPM flag set). Nevertheless, the history list remains in the overflow condition, until FCNnCMRGRVFF is cleared by software. If FCNnCMRGRVFF is not cleared, the FCNnCMRGSSPM flag will also not be updated (cleared) upon a message storage of newly received frame. This may lead to the situation, that FCNnCMRGSSPM indicates an empty history list, although a reception has taken place, while the history list is in the overflow state (FCNnCMRGRVFF and FCNnCMRGSSPM are set).

As long as the RHL still has free entries, the sequence of occurrence is maintained. If more receptions occur without reading the RHL by the host processor, complete sequence of receptions can not be recovered.




RGRVFF = 1 denoted that LISSLR equals RGSSPT - 1 while message buffer number stored to element indicated by LISSLR - 1.

Note: In the diagram following abbreviations are used:
- LISSLR = Last in-message pointer FCNnCMLISTR.FCNnCMLISSLR[7:0] (LISSLR)
- RGSSPT = Receive history list get pointer FCNnCMRGRX.FCNnCMRGSSPT[7:0] (RGSSPT)
- RGRVFF = Receive history list overflow flag FCNnCMRGRX.FCNnCMRGRVFF (RGRVFF)

**Figure 18-9 Receive history list**

### 18.9.4 Mask function

For any message buffer, which is used for reception, the assignment to one of eight global reception masks (or no mask) can be selected.

By using the mask function, the message ID comparison can be reduced by masked bits, herewith allowing the reception of several different IDs into one buffer.

While the mask function is in effect, an identifier bit that is defined to be 1 by a mask in the received message is not compared with the corresponding identifier bit in the message buffer.

However, this comparison is performed for any bit whose value is defined as 0 by the mask.

For example, let us assume that all messages that have a standard-format ID, in which bits ID27 to ID25 are 0 and bits ID24 and ID22 are 1, are to be stored in message buffer 14. The procedure for this example is shown below.

**(1) Identifier to be stored in message buffer**

| ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 |
|---|---|---|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 0 | 1 | x | 1 | x | x | x | x |

| ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 |
|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x |

| ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |
|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x |

**(2) Identifier to be configured in message buffer 14 (example) (using FCNnCM14MID0W register)**

| ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 |
|---|---|---|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 0 | 1 | x | 1 | x | x | x | x |

| ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 |
|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x |

| ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |
|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x |

**Notes**

1. ID with the ID27 to ID25 bits cleared to 0 and the ID24 and ID22 bits set to 1 is registered (initialized) to message buffer 14. Other bits of ID can be set to any value (x), because they are going to be masked.
2. Message buffer 14 is set as a standard format identifier that is linked to mask 1 (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0010_B$).

**Mask setting for CAN Controller 1 (mask 1) (example)**
**(using CAN1 address mask 1 register FCNnCMMKCTL01W)**

FNCnCMMKSSID[..]

| ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |

1: Not compared (masked)
0: Compared

FCNnCMMKSSID[27:24] and FCNnCMMKSSID[22] are cleared to 0, and FCNnCMMKSSID[28], FCNnCMMKSSID[23], and FCNnCMMKSSID[21:0] are set to 1.

### 18.9.5 Multi buffer receive block function

The multi buffer receive block (MBRB) function is used to store a block of data in two or more message buffers sequentially with no CPU interaction, by setting the same ID to two or more message buffers with the same message buffer type. These message buffers can be allocated anywhere in the message buffer memory, they do not even have to follow each other adjacently.

Suppose, for example, the same message buffer type is set to 10 message buffers, message buffers 10 to 19, and the same ID is set to each message buffer. If the first message whose ID matches an ID of the message buffers is received, it is stored in message buffer 10. At this point, FCNnMmCTL.FCNnMmDTNF of message buffer 10 is set, prohibiting overwriting the message buffer when subsequent messages are received.

When the next message with a matching ID is received, it is received and stored in message buffer 11. Each time a message with a matching ID is received, it is sequentially (in the ascending order) stored in message buffers 12, 13, and so on. Even when a data block consisting of multiple messages is received, the messages can be stored and received without overwriting the previously received matching-ID data.

Whether a data block has been received and stored can be checked by setting FCNnMmCTL.FCNnMmIENF of each message buffer. For example, if a data block consists of k messages, k message buffers are initialized for reception of the data block. FCNnMmIENF in message buffers 0 to (k-2) is cleared to 0 (interrupts disabled), and FCNnMmIENF in message buffer k-1 is set to 1 (interrupts enabled). In this case, a reception completion interrupt occurs when a message has been received and stored in message buffer k-1, indicating that MBRB has become full. Alternatively, by clearing FCNnMmIENF of message buffers 0 to (k-3) and setting FCNnMmIENF of message buffer k-2, a warning that MBRB is about to overflow can be issued.

The basic conditions of storing receive data in each message buffer for the MBRB are the same as the conditions of storing data in a single message buffer.

**Cautions**

1. MBRB can be configured for each of the same message buffer types. Therefore, even if a message buffer of another MBRB whose ID matches but whose message buffer type is different has a vacancy, the received message is not stored in that message buffer, but instead discarded.
2. MBRB does not have a ring buffer structure. Therefore, after a message is stored in the message buffer having the highest number in the MBRB configuration, a newly received message will not be stored in the message buffer having the lowest message buffer number.
3. MBRB operates based on the reception and storage conditions; there are no settings dedicated to MBRB, such as function enable bits. By setting the same message buffer type and ID to two or more message buffers, MBRB is automatically configured.
4. With MBRB, “matching ID” means “matching ID after mask”. Even if the ID set to each message buffer is not the same, if the ID that is masked by the mask register matches, it is considered a matching ID and the buffer that has this ID is treated as the storage destination of a message.
5. The priority between MBRBs is mentioned in the table *Table 18-20 “MBRB priorities”*.

### 18.9.6 Remote frame reception

In all the operation modes, when a remote frame is received, the message buffer that is to store the remote frame is searched from all the message buffers satisfying the following conditions (1 and 2, condition 1 has priority on reception acceptance). If condition 1 is not fulfilled, the remaining message buffers are scanned, whether condition 2 could be fulfilled.

- Condition 1:
  Set as a transmit message buffer
  (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0000_B$)
  - Used as a message buffer
    (FCNnMmSTRB.FCNnMmSSAM = 1.)
  - Ready for reception
    (FCNnMmCTL.FCNnMmRDYF = 1.)
  - Set to data frame message type
    (FCNnMmSTRB.FCNnMmSSRT = 0.)
  - Transmission request is not set.
    (FCNnMmCTL.FCNnMmTRQF = 0.)
- Condition 2:
  Set as a receive message buffer
  (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0001_B$ ... $1001_B$)
  - Used as a message buffer
    (FCNnMmSTRB.FCNnMmSSAM = 1.)
  - Ready for reception
    (FCNnMmCTL.FCNnMmRDYF = 1.)
  - Set to remote frame message type
    (FCNnMmSTRB.FCNnMmSSRT = 1.)
  - Buffer is ready to store a message
    (FCNnMmCTL.FCNnMmDTNF = 0, or FCNnMmSTRB.FCNnMmSSOW = 1 with FCNnMmCTL.FCNnMmDTNF = 1).

Upon acceptance of a remote frame, the following actions are executed if the ID of the received remote frame matches the ID of a message buffer that satisfies the above conditions.

- The FCNnMmDTLG[3:0] bit string in the FCNnMmDTLGB register store the received DLC value.
- The FCNnMmDAT0B to FCNnMmDAT7B registers in the data area are not updated (data before reception is saved).
- FCNnMmCTL.FCNnMmDTNF is set to 1.
- FCNnCMISCTL.FCNnCMISITSF1 is set to 1 (if FCNnMmCTL.FCNnMmIENF of the message buffer that receives and stores the frame is set to 1).
- The receive completion interrupt (INTCnREC) is output (if FCNnMmCTL. FCNnMmIENF of the message buffer that receives and stores the frame is set to 1 and if FCNnCMIECTL.FCNnCMIESEIE1 is set to 1).
- The message buffer number is recorded in the receive history list, if the flag FCNnMmCTL.FCNnMmNHMF is not set.

**Caution** When a transmit message buffer is found for receiving and storing a remote frame, overwrite control by FCNnMmSTRB.FCNnMmSSOW of the message buffer and FCNnMmCTL.FCNnMmDTNF are not checked. The setting of FCNnMmSSOW is ignored, and FCNnMmDTNF is set in any case.

**Notes**

1. If more than one transmit message buffer has the same ID and the ID of the received remote frame matches that ID, the remote frame is stored in the transmit message buffer with the lowest message buffer number.
2. If transmit and receive message buffers are found, which could receive a remote frame matching with its ID, either masked or unmasked, the remote frame is stored in the transmit message buffer.
3. If several receive message buffers would match for reception for a remote frame, the reception priority is identical as for a data frame.
4. If a receive message buffer is found to match for a remote frame reception, and selected for storage, but this receive message buffer does not allow the storage, because FCNnMmDTNF is set, and FCNnMmSSOW is not set, the remote frame is not stored at all.

# 18.10 Message Transmission

## 18.10.1 Message transmission

A message buffer with its FCNnMmCTL.FCNnMmTRQF bit set to 1 participates in the search for the most high-prioritized message when the following conditions are fulfilled. This behavior is valid for all operational modes.

- Used as a message buffer (FCNnMmSTRB.FCNnMmSSAM = 1.)
- Set as a transmit message buffer (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0000_B$.)
- Ready for transmission (FCNnMmCTL.FCNnMmRDYF = 1.)

The CAN bus is a multi-master communication system. In a system like this, the priority of message transmission is determined based on message identifiers (IDs). To facilitate transmission processing by software when there are several messages awaiting transmission, the CAN Controller uses hardware to check the ID of the message with the highest priority and automatically identifies that message. This eliminates the need for software-based priority control.

Transmission priority is controlled by the identifier (ID).

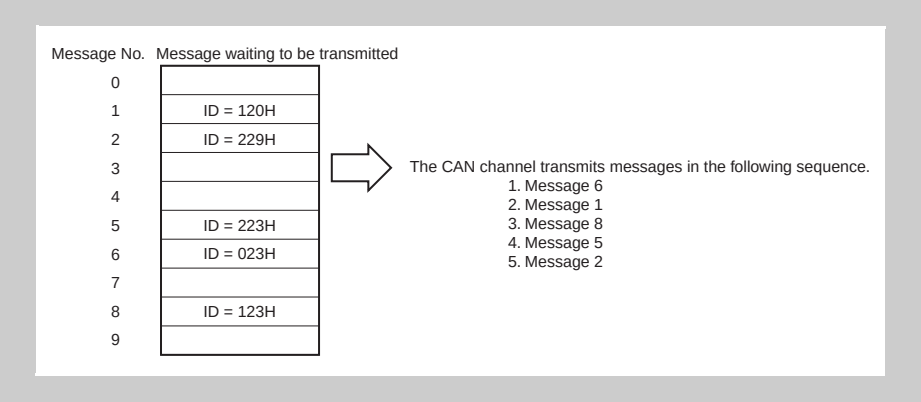



**Figure 18-10 Message processing example**

After the transmit message search, the transmit message with the highest priority of the transmit message buffers that have a pending transmission request (message buffers with the FCNnMmCTL.FCNnMmTRQF bit set to 1 in advance) is transmitted.

If a new transmission request is set, the transmit message buffer with the new transmission request is compared with the transmit message buffer with a pending transmission request. If the new transmission request has a higher priority, it is transmitted, unless transmission of a message with a low priority has already started. If transmission of a message with a low priority has already started, however, the new transmission request is transmitted later. To solve this priority inversion effect, the software can perform a transmission abort request for the lower priority message. The highest priority is determined according to the following rules.

| Priority | Conditions | Description |
|---|---|---|
| 1 (high) | Value of first 11 bits of ID [ID28 to ID18]: | The message frame with the lowest value represented by the first 11 bits of the ID is transmitted first. If the value of an 11-bit standard ID is equal to or smaller than the first 11 bits of a 29-bit extended ID, the 11-bit standard ID has a higher priority than a message frame with a 29-bit extended ID. |
| 2 | Frame type | A data frame with an 11-bit standard ID (FCNnMmSTRB.FCNnMmSSRT cleared to 0) has a higher priority than a remote frame with a standard ID and a message frame with an extended ID. |
| 3 | ID type | A message frame with a standard ID (message buffer identifier register FCNnMmMID... bit FCNnMmSSIE is cleared to 0) has a higher priority than a message frame with an extended ID. |
| 4 | Value of lower 18 bits of ID [ID17 to ID0]: | If one or more transmission-pending extended ID message frame has equal values in the first 11 bits of the ID and the same frame type (equal FCNnMmSTRB.FCNnMmSSRT bit values), the message frame with the lowest value in the lower 18 bits of its extended ID is transmitted first. |
| 5 (low) | Message buffer number | If two or more message buffers request transmission of message frames with the same ID, the message from the message buffer with the lowest message buffer number is transmitted first. |

**Notes 1.** If the automatic block transmission request bit FCNnGMABCTL.FCNnGMABABTT is set to 1 in the normal operation mode with ABT, FCNnMmCTL.FCNnMmTRQF is set to 1 only for one message buffer in the ABT message buffer group.

If the ABT mode was triggered by FCNnGMABCTL.FCNnGMABSEAT = 1, one FCNnMmCTL.FCNnMmTRQF is set to 1 in the ABT area (buffer 0 through 7). Beyond this transmit request, the application can request transmissions (set FCNnMmTRQF to 1) for other TX-message buffers that do not belong to the ABT area. In that case an interval arbitration process (TX-search) evaluates all TX-message buffers with FCNnMmTRQF set to 1 and chooses the message buffer that contains the highest prioritized identifier for the next transmission. If there are 2 or more identifiers that have the highest priority (i.e. identical identifiers), the message located at the lowest message buffer number is transmitted at first.

Upon successful transmission of a message frame, the following operations are performed.

- The FCNnMmCTL.FCNnMmTRQF flag of the corresponding transmit message buffer is automatically cleared to 0.
- The transmission completion status bit FCNnCMISCTL.FCNnCMISITSF0 is set to 1 (if the interrupt enable bit FCNnMmIENF of the corresponding transmit message buffer is set to 1).
- An interrupt request signal INTCnTRX is output (if FCNnCMIECTL.FCNnCMIESEIE0 is set to 1 and if the interrupt enable bit FCNnMmIENF of the corresponding transmit message buffer is set to 1).

**2.** When changing the contents of a transmit buffer, the FCNnMmCTL.FCNnMmRDYF flag of this buffer must be cleared before updating the buffer contents. As during internal transfer actions, the FCNnMmRDYF flag may be locked temporarily, the status of FCNnMmRDYF must be checked by software, after changing it.

### 18.10.2 Transmit history list function

The transmit history list (THL) function records in the transmit history list the number of the transmit message buffer from which data or remote frames have been were sent. The THL consists of storage elements equivalent to up to 7 messages (on 32 message buffer CAN Controller), 15 messages (on 64 message buffer CAN Controller) or up to 31 messages (on 128 message buffer CAN Controller), the last out-message pointer FCNnCMLOSTR[7:0] with the corresponding FCNnCMLOSTR register, and the transmit history list get pointer FCNnCMTGSSPT[7:0] with the corresponding FCNnCMTGTX register.

The THL is undefined immediately after the transition of the CAN Controller from the initialization mode to one of the operation modes.

The FCNnCMLOSTR register holds the contents of the THL element indicated by the value of the FCNnCMLOSTR.FCNnCMLOSTR[7:0] pointer minus 1. By reading the FCNnCMLOSTR register, therefore, the number of the message buffer that transmitted a data frame or remote frame first can be checked. The FCNnCMLOSTR[7:0] pointer is utilized as a write pointer that indicates to what part of the THL a message buffer number is recorded. Any time a data frame or remote frame is transmitted, the corresponding message buffer number is recorded to the THL element indicated by the FCNnCMLOSTR[7:0] pointer. Each time recording to the THL has been completed, the FCNnCMLOSTR[7:0] pointer is automatically incremented. In this way, the number of the message buffer that has received and stored a frame will be recorded chronologically.

For message buffers, where the flag FCNnMmCTL.FCNnMmNHMF is set, no entry in the history lists is recorded.

The FCNnCMTGTX.FCNnCMTGSSPT[7:0] pointer is utilized as a read pointer that reads a recorded message buffer number from the THL. This pointer indicates the first THL element that the CPU has not yet read. By reading the FCNnCMTGTX register by software, the number of a message buffer that has completed transmission can be read. Each time a message buffer number is read from the FCNnCMTGTX register, the FCNnCMTGSSPT[7:0] pointer is automatically incremented.

If the value of the FCNnCMTGTX.FCNnCMTGSSPT[7:0] pointer matches the value of the FCNnCMLOSTR.FCNnCMLOSTR[7:0] pointer, FCNnCMTGTX.FCNnCMTGSSPM (transmit history list pointer match) is set to 1. This indicates that no message buffer numbers that have not been read remain in the THL. If a new message buffer number is recorded, the FCNnCMLOSTR[7:0] pointer is incremented and because its value no longer matches the value of the FCNnCMTGSSPT[7:0] pointer, FCNnCMTGSSPM is cleared. In other words, the numbers of the unread message buffers exist in the THL.

If the FCNnCMLOSTR.FCNnCMLOSTR[7:0] pointer is incremented and matches the value of the FCNnCMTGTX.FCNnCMTGSSPT[7:0] pointer minus 1, FCNnCMTGTX.FCNnCMTGTVFF (transmit history list overflow) is set to 1. This indicates that the THL is full of message buffer numbers that have not been read. If a new message is received and stored, the message buffer number recorded last is overwritten by the message buffer number that transmitted its message afterwards. In this case, after FCNnCMTGTVFF has been set (1), therefore, the recorded message buffer numbers in the THL do not completely reflect the chronological order. However the other transmitted messages can be found by a CPU search applied to all transmit message buffers unless the CPU has not overwritten a transmit object in one of these buffers beforehand. In total up to six transmission completions can occur without overflowing the THL.

**Caution** If the history list is in the overflow condition (FCNnCMTGTX.FCNnCMTGTVFF is set), reading the history list contents is still possible, until the history list is empty (indicated by FCNnCMTGTX.FCNnCMTGSSPM flag set). Nevertheless, the history list remains in the overflow condition, until FCNnCMTGTVFF is cleared by software. If FCNnCMTGTVFF is not cleared, the FCNnCMTGTX.FCNnCMTGSSPM flag will also not be updated (cleared) upon successful transmission of a new message. This may lead to the situation, that FCNnCMTGSSPM indicates an empty history list, although a successful transmission has taken place, while the history list is in the overflow state (FCNnCMTGTVFF and FCNnCMTGSSPM are set).

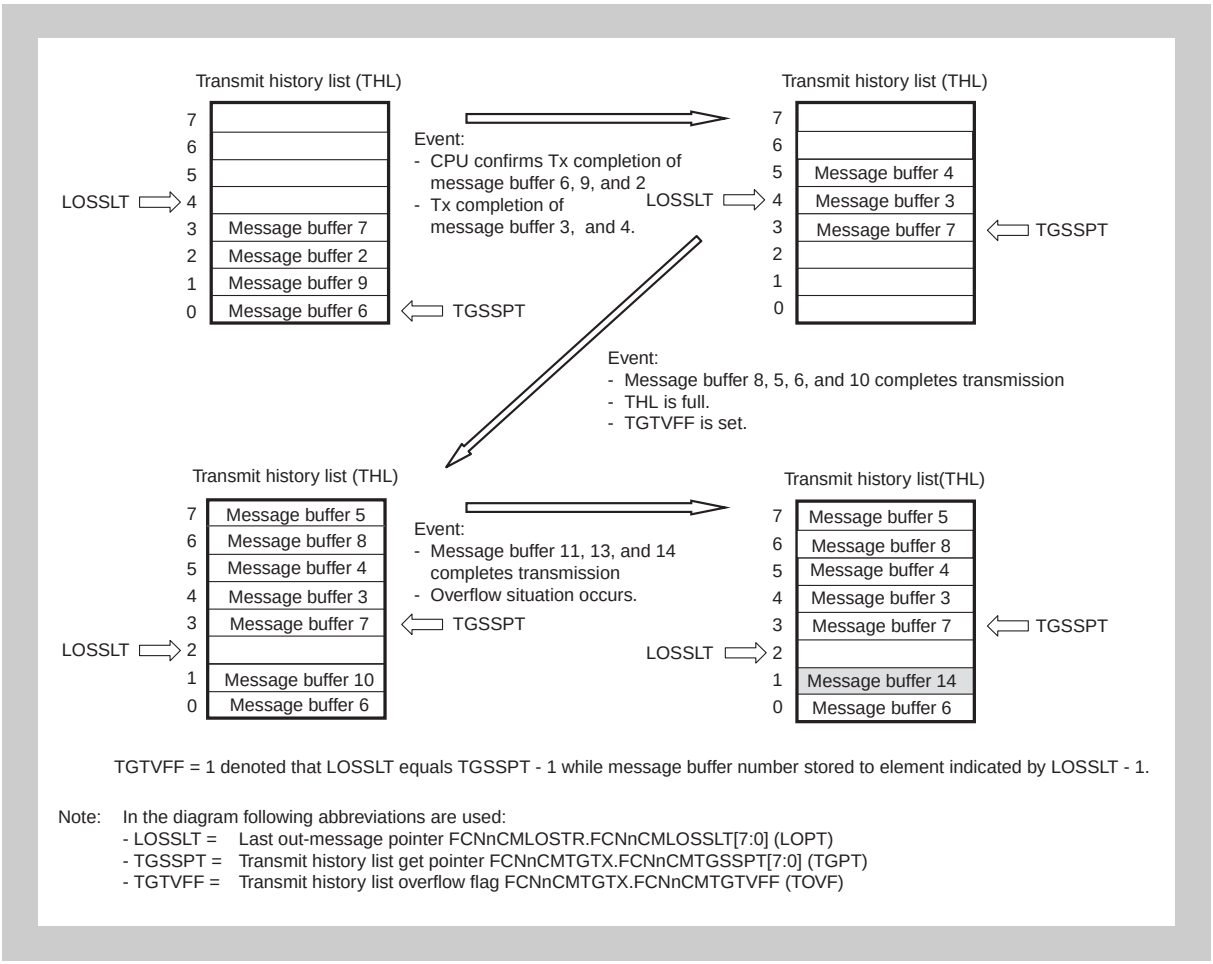



**Figure 18-11 Transmit history list**

### 18.10.3 Automatic block transmission (ABT)

The automatic block transmission (ABT) function is used to transmit two or more data frames successively with no CPU interaction. The maximum number of transmit message buffers assigned to the ABT function is 8 (for 32 message buffer CAN Controller), 16 (for 64 message buffer CAN Controller) or 32 (for 128 message buffer CAN Controller), always located in the lowest message buffers.

By setting FCNnCM.FCNnCMCLMDOF[2:0] to $010_B$, “normal operation mode with automatic block transmission function” (hereafter referred to as ABT mode) can be selected.

To issue an ABT transmission request, define the message buffers by software first. Set FCNnMmSTRB.FCNnMmSSAM = 1 in all the message buffers used for ABT, and define all the buffers as transmit message buffers by setting the FCNnMmSTRB.FCNnMmSSMT[3:0] bits to $0000_B$. Be sure to set the same ID for the message buffers for ABT even when that ID is being used for all the message buffers. To use two or more IDs, set the ID of each message buffer by using the FCNnMmMID0H and FCNnMmMID1H or FCNnMmMID0W registers. Set the CAN Controller message buffer data bytes registers before issuing a transmission request for the ABT function.

After initialization of message buffers for ABT is finished, FCNnMmCTL.FCNnMmRDYF needs to be set to 1. In the ABT mode, FCNnMmCTL.FCNnMmTRQF does not have to be manipulated by software.

After the data for the ABT message buffers has been prepared, set FCNnGMABCTL.FCNnGMABSEAT = 1. Automatic block transmission is then started. When ABT is started, FCNnMmCTL.FCNnMmTRQF in the first message buffer (message buffer 0) is automatically set to 1. After transmission of the data of message buffer 0 is finished, the FCNnMmTRQF of the next message buffer, message buffer 1, is set automatically. In this way, transmission is executed successively.

A delay time can be inserted by program in the interval in which the transmission request FCNnMmCTL.FCNnMmTRQF is automatically set while successive transmission is being executed. The delay time to be inserted is defined by the FCNnGMADCTL register. The unit of the delay time is DBT (data bit time). DBT depends on the setting of the FCNnCMBRPRS and FCNnCMBTCTL registers.

Among transmit objects within the ABT-area, the priority of the transmission ID is not evaluated. The data of message buffers 0 to 7 are sequentially transmitted. When transmission of the data frame from message buffer 7 has been completed, FCNnGMABCTL.FCNnGMABABTT is automatically cleared to 0 and the ABT operation is finished.

If FCNnMmCTL.FCNnMmRDYF of an ABT message buffer is cleared during ABT, no data frame is transmitted from that buffer, ABT is stopped, and FCNnGMABCTL.FCNnGMABABTT is cleared. After that, transmission can be resumed from the message buffer where ABT stopped, by setting FCNnMmRDYF and FCNnGMABABTT to 1 by software. To not resume transmission from the message buffer where ABT stopped, the internal ABT engine can be reset by setting the FCNnGMABCTL.FCNnGMABCLRF bit to 1 while ABT mode is stopped and FCNnGMABABTT is cleared to 0. In this case, transmission is started from message buffer 0 if FCNnGMABCTL.FCNnGMABSEAC is cleared to 0 and then FFCNnGMABABTT is set to 1.

An interrupt can be used to check if data frames have been transmitted from all the message buffers for ABT. To do so, FCNnMmCTL.FCNnMmIENF of each message buffer except the last message buffer needs to be cleared (0).

If a transmit message buffer other than those used by the ABT function is assigned to a transmit message buffer, the message to be transmitted next is determined by the priority of the transmission ID of the ABT message buffer whose transmission is currently held pending and the transmission ID of the message buffers other than those used by the ABT function.

Transmission of a data frame from an ABT message buffer is not recorded in the transmit history list (THL).

**Cautions**

1. Set FCNnGMABCTL.FCNnGMABSEAC = 1 while FCNnGMABCTL.FCNnGMABABTT is cleared to 0 in order to resume ABT operation at buffer No. 0. If FCNnGMABSEAC is set to 1 while FCNnGMABABTT is set to 1, the subsequent operation is not guaranteed.
2. If the automatic block transmission engine is cleared by setting FCNnGMABCTL.FCNnGMABSEAC = 1, FCNnGMABSEAC is automatically cleared immediately after the processing of the clearing request is completed.
3. Do not trigger automatic block transmission in the initialization mode. If FCNnGMABCTL.FCNnGMABSEAT is set in the initialization mode, the proper operation is not guaranteed after the mode is changed from the initialization mode to the ABT mode.
4. Do not set FCNnMmCTL.FCNnMmTRQF of the ABT message buffers to 1 by software in the normal operation mode with ABT. Otherwise, the operation is not guaranteed.
5. The FCNnGMADCTL register is used to set the delay time that is inserted in the period from completion of the preceding ABT message to setting of FCNnMmCTL.FCNnMmTRQF for the next ABT message when the transmission requests are set in the order of message numbers for each message for ABT that is successively transmitted in the ABT mode. The timing at which the messages are actually transmitted onto the CAN bus varies depending on the status of transmission from other stations and the status of the setting of the transmission request for messages other than the ABT messages.
6. If a transmission request is made for a message other than an ABT message and if no delay time is inserted in the interval in which transmission requests for ABT are automatically set (FCNnGMADCTL = $00_H$), messages other than ABT messages may be transmitted not depending on their priority compared to the priority of the ABT message.
7. Do not clear FCNnMmCTL.FCNnMmRDYF to 0 when FCNnGMABCTL.FCNnGMABABTT = 1.
8. If a message is received from another node while normal operation mode with ABT is active, the TX-message from the ABT-area may be transmitted with delay of one frame although FCNnGMADCTL register was set up with $00_H$.

### 18.10.4 Transmission abort process

**(1) Transmission abort process except for in normal operation mode with automatic block transmission (ABT)**

The user can clear FCNnMmCTL.FCNnMmTRQF to 0 to abort a transmission request. FCNnMmTRQF will be cleared immediately if the abort was successful. Whether the transmission was successfully aborted or not can be checked using FCNnCMCLCTL.FCNnCMCLSSTS and the FCNnCMTGTX register, or the FCNnMmCTL.FCNnMmTCPF flag, which indicate the transmission status on the CAN bus (for details, refer to the processing in *Figure 18-27 “Transmission abort processing (except normal operation mode with ABT)” on page 1011* ).

**(2) Transmission abort process except for ABT transmission in normal operation mode with automatic block transmission (ABT)**

The user can clear FCNnGMABCTL.FCNnGMABABTT to 0 to abort a transmission request. After checking FCNnGMABABTT = 0, clear FCNnMmCTL.FCNnMmTRQF to 0. FCNnMmTRQF will be cleared immediately if the abort was successful. Whether the transmission was successfully aborted or not can be checked using FCNnCMCLCTL.FCNnCMCLSSTS and the FCNnCMTGTX register, or the FCNnMmCTL.FCNnMmTCPF flag, which indicate the transmission status on the CAN bus (for details, refer to the processing in *Figure 18-28 “Transmission abort processing (normal operation mode with ABT) - Repeat option for aborted message” on page 1012* ).

**(3) Transmission abort process for ABT transmission in normal operation mode with automatic block transmission (ABT)**

To abort ABT that is already started, clear FCNnGMABCTL.FCNnGMABCLAT to 0. In this case, FCNnGMABCTL.FCNnGMABABTT remains 1 if an ABT message is currently being transmitted and until the transmission is completed (successfully or not), and is cleared to 0 as soon as transmission is finished. This aborts ABT.

If the last transmission (before ABT) was successful, the normal operation mode with ABT is left with the internal ABT pointer pointing to the next message buffer to be transmitted.

In the case of an erroneous transmission, the position of the internal ABT pointer depends on the status of FCNnMmCTL.FCNnMmTRQF in the last transmitted message buffer. If FCNnMmTRQF is set to 1 when clearing FCNnGMABCTL.FCNnGMABABTT is requested, the internal ABT pointer points to the last transmitted message buffer (for details, refer to the process in *Figure 18-29 “Transmission abort processing (normal operation mode with ABT) - No repetition option for aborted message” on page 1013* ). If FCNnMmTRQF is cleared to 0 when clearing FCNnGMABABTT is requested, the internal ABT pointer is incremented (+1) and points to the next message buffer in the ABT area (for details, refer to the process in *Figure 18-30 “ABT transmission request abort processing (normal operation mode with ABT)” on page 1014* ).

**Caution** Be sure to abort ABT by clearing FCNnGMABCTL.FCNnGMABCLAT to 0. The operation is not guaranteed if aborting transmission is requested by clearing FCNnMmCTL.FCNnMmRDYF.

When the normal operation mode with ABT is resumed after ABT has been aborted and FCNnGMABCTL.FCNnGMABSEAT is set to 1, the next ABT message buffer to be transmitted can be determined from the following table.

| Status of FCNnMmCTL.FCNnMmTRQF of ABT message buffer | Abort after successful transmission | Abort after erroneous transmission |
|---|---|---|
| Set (1) | Next message buffer in the ABT area[a] | Same message buffer in the ABT area |
| Cleared (0) | Next message buffer in the ABT area[a] | Next message buffer in the ABT area[a] |

a) The above resumption operation can be performed only if a message buffer ready for ABT exists in the ABT area. For example, an abort request that is issued while ABT of highest ABT message buffer is in progress is regarded as completion of ABT, rather than abort, if transmission of this message buffer has been successfully completed, even if FCNnGMABCTL.FCNnGMABABTT is cleared to 0. If FCNnMmCTL.FCNnMmRDYF in the next message buffer in the ABT area is cleared to 0, the internal ABT pointer is retained, but the resumption operation is not performed even if FCNnGMABABTT is set to 1, and ABT ends immediately.

### 18.10.5 Remote frame transmission

Remote frames can be transmitted only from transmit message buffers. Set whether a data frame or remote frame is transmitted via FCNnMmSTRB.FCNnMmSSRT. Setting FCNnMmSSRT = 1 sets remote frame transmission.

## 18.11 Power Saving Modes

### 18.11.1 CAN Controller sleep mode

The CAN Controller sleep mode can be used to set the CAN Controller to stand-by mode in order to reduce power consumption. The CAN Controller can enter the sleep mode from all operation modes. Release of the sleep mode returns the CAN Controller to exactly the same operation mode from which the sleep mode was entered.

In the sleep mode, the CAN Controller does not transmit messages, even when transmission requests are issued or pending.

**(1) Entering CAN Controller sleep mode**

The CPU issues a CAN Controller sleep mode transition request by setting FCNnCMCLCTL.FCNnCMCLMDPF[1:0] = $01_B$.

This transition request is acknowledged only under the following conditions.

1. The CAN Controller is already in one of the following operation modes
   - Normal operation mode
   - Normal operation mode with ABT
   - Receive-only mode
   - Single-shot mode
   - Self-test mode
   - CAN Controller stop mode in all the above operation modes
2. The CAN bus state is bus idle (the 4th bit in the interframe space is recessive).
   If the CAN bus is fixed to dominant, the request for transition to the sleep mode is held pending. Also the transition from stop mode to sleep mode is independent of the CAN bus state.
3. No transmission request is pending.

**Note** If a sleep mode request is pending, and at the same time a message is received in a message box, the sleep mode request is not cancelled, but is executed right after message storage has been finished. This may result in CAN Controller being in sleep mode, while the CPU would execute the RX interrupt routine. Therefore, the interrupt routine must check the access to the message buffers as well as reception history list registers by using the FCNnGMCLSSMO flag, if sleep mode is used.

If any one of the conditions mentioned above is not met, the CAN Controller will operate as follows.

- If the sleep mode is requested from the initialization mode, the sleep mode transition request is ignored and the CAN Controller remains in the initialization mode.
- If the CAN bus state is not bus idle (i.e., the CAN bus state is either transmitting or receiving) when the sleep mode is requested in one of the operation modes, immediate transition to the sleep mode is not possible. In this case, the sleep mode transition request is held pending until the CAN bus state becomes bus idle (the 4th bit in the interframe space is recessive). In the time from the sleep mode request to successful transition, FCNnCMCLCTL.FCNnCMCLMDPF[1:0] remain $00_B$. When the channel has entered the sleep mode, the FCNnCMCLMDPF[1:0] bits are set to $01_B$.

- If a request for transition to the initialization mode and a request for transition to the sleep mode are made at the same time while the CAN Controller is in one of the operation modes, the request for the initialization mode is enabled. The CAN Controller enters the initialization mode at a predetermined timing. At this time, the sleep mode request is not held pending and is ignored.
- Even when initialization mode and sleep mode are not requested simultaneously (i.e the first request has not been granted while the second request is made), the request for initialization has priority over the sleep mode request. The sleep mode request is cancelled when the initialization mode is requested. When a pending request for initialization mode is present, a subsequent request for Sleep mode request is cancelled right at the point in time where it was submitted.

**(2) Status in CAN Controller sleep mode**

The CAN Controller is in the following state after it enters the sleep mode:

- The CAN channel clock $T_{CANCH}$ is stopped and the power consumption is minimized.
- The function to detect the falling edge of the CAN Controller reception pin (CRXDn) remains in effect to wake up the CAN Controller from the CAN bus.
- To wake up the CAN Controller from the CPU, data can be set to FCNnCMCLCTL.FCNnCMCLMDPF[1:0], but nothing can be written to other CAN Controller registers or bits.
- The CAN Controller registers can be read, except for the FCNnCMLISTR, FCNnCMRGRX, FCNnCMLOSTR, and FCNnCMTGTX registers.
- The CAN Controller message buffer registers cannot be written or read.
- FCNnGMCLCTL.FCNnGMCLSSMO is cleared.
- The registers FCNnDNBMRX cannot be read.
- A request for transition to the initialization mode is not acknowledged and is ignored.

**(3) Releasing CAN Controller sleep mode**

The CAN Controller sleep mode is released by the following events:

- When the CPU sets $00_B$ to FCNnCMCLCTL.FCNnCMCLMDPF[1:0]
- A falling edge at the CAN Controller reception pin CRXDn (i.e. the CAN bus level shifts from recessive to dominant)

**Caution** Even if the falling edge belongs to the SOF of a receive message, this message will not be received and stored. If the CPU has turned off the clock supply to the CAN Controller while the CAN Controller was in sleep mode, even subsequently the sleep mode will not be released and FCNnCMCLMDPF[1:0] will remain $01_B$ unless the clock to the CAN Controller is supplied again. In addition to this, the receive message will not be received after that.

After releasing the sleep mode, the CAN Controller returns to the operation mode from which the sleep mode was requested and FCNnCMCLCTL.FCNnCMCLCTL[1:0] must be reset by software to $00_B$. If the CAN Controller sleep mode is released by a change in the CAN bus state, FCNnCMISCTL.FCNnCMISITSF5 is set to 1, regardless of FCNnCMIECTL.FCNnCMIEINTF[6:0]. After the CAN Controller is released from the sleep mode, it participates in the CAN bus again by automatically detecting 11 consecutive recessive-level bits on the CAN bus. The user application has to wait until FCNnGMCLCTL.FCNnGMCLSSMO = 1, before accessing message buffers again.

When a request for transition to the initialization mode is made while the CAN Controller is in the sleep mode, that request is ignored; the CAN Controller has to be released from sleep mode by software first before entering the initialization mode.

**Cautions**

1. Be aware that the release of sleep mode by CAN bus event, and thus the wake up interrupt may happen at any time, even right after requesting sleep mode, if a CAN bus event occurs.
2. Always reset the FCNnCMCLCTL.FCNnCMCLMDPF[1:0] bits to $00_B$, when waking up from sleep mode, before accessing any other registers of the CAN Controller.
3. Always clear the interrupt flag FCNnCMISCTL.FCNnCMISITSF5, when waking up from sleep mode.

### 18.11.2 CAN Controller stop mode

The CAN Controller stop mode can be used to set the CAN Controller to stand-by mode to reduce power consumption. The CAN Controller can enter the stop mode only from the sleep mode. Release of the stop mode puts the CAN Controller in the sleep mode.

The stop mode can only be released (entering sleep mode) by setting $01_B$ to FCNnCMCLCTL.FCNnCMCLMDPF[1:0] and not by a change in the CAN bus state. No message is transmitted even when transmission requests are issued or pending.

**(1) Entering CAN Controller stop mode**

A stop mode transition request is issued by setting $11_B$ to FCNnCMCLCTL.FCNnCMCLMDPF[1:0].

A stop mode request is only acknowledged when the CAN Controller is in the sleep mode. In all other modes, the request is ignored.

**Caution** To set the CAN Controller to the stop mode, it must be in the sleep mode first. To confirm that the CAN Controller is in the sleep mode, check that the FCNnCMCLCTL.FCNnCMCLMDPF[1:0] = $01_B$, and then request the stop mode. If a bus change occurs at the CAN Controller reception pin CRXDn while this process is being performed, the sleep mode is automatically released. In this case, the stop mode transition request cannot be acknowledged.

**(2) Status in CAN Controller stop mode**

The CAN Controller is in the following state after it enters the stop mode.

- The CAN channel clock $T_{CANCH}$ is stopped and the power consumption is minimized.
- To wake up the CAN Controller from the CPU, data can be set to FCNnCMCLCTL.FCNnCMCLMDPF[1:0], but nothing can be written to other CAN Controller registers or bits.
- The CAN Controller registers can be read, except for the FCNnCMLISTR, FCNnCMRGRX, FCNnCMLOSTR, and FCNnCMTGTX registers.
- The CAN Controller message buffer registers cannot be written or read.
- FCNnGMCLCTL.FCNnGMCLSSMO is cleared.
- The registers FCNnDNBMRX cannot be read.
- An initialization mode transition request is not acknowledged and is ignored.

**(3) Releasing CAN Controller stop mode**

The stop mode can only be released by writing $01_B$ to FCNnCMCLCTL.FCNnCMCLMDPF[1:0]. After releasing the stop mode, the CAN Controller enters the sleep mode.

When the initialization mode is requested while the CAN Controller is in the stop mode, that request is ignored; the CPU has to release the stop mode and subsequently sleep mode before entering the initialization mode. It is impossible to enter the other operation mode directly from the stop mode not entering the sleep mode, that request is ignored.

### 18.11.3 Example of using power saving modes

In some application systems, it may be necessary to place the CPU in a power saving mode to reduce the power consumption. By using the power saving mode specific to the CAN Controller and the power saving mode specific to the CPU in combination, the CPU can be woken up from the power saving status by the CAN bus.

Here is an example for using the power saving modes.

- First, put the CAN Controller in the sleep mode (FCNnCMCLCTL.FCNnCMCLMDPF[1:0] = $01_B$).
  After successfully confirming this state by reading back the sleep mode status, put the CPU in the power saving mode. Disable interrupts for the CPU, while processing additional tasks after the CAN Controller is in sleep mode, to avoid that the CAN Controller wakeup interrupt is acknowledged. If an edge transition from recessive to dominant is detected at the CAN Controller reception pin CRXDn in this status, FCNnCMISCTL.FCNnCMISITSF5 in the CAN Controller is set to 1. If FCNnCNIECTL.FCNnCMIEINT5 is set to 1, a wakeup interrupt (INTCnWUP) is generated.
  The CAN Controller is automatically released from sleep mode (FCNnCMCLMDPF[1:0] = $00_B$) and returns to normal operation mode.
- The CPU, in response to INTCnWUP, can release its own power saving mode and return to normal operation mode.

  To further reduce the power consumption of the CPU, the internal clock - including that of the CAN Controller - may be stopped. In this case, the operating clock supplied to the CAN Controller is stopped after the CAN Controller has been put in sleep mode. Then the CPU enters a power saving mode in which the clock supplied to the CPU is stopped.
- If an edge transition from recessive to dominant is detected at the CAN Controller reception pin CRXDn in this status, the CAN Controller can set FCNnCMISCTL.FCNnCMISITSF5 to 1 and generate the wakeup interrupt INTCnWUP even if it is not supplied with the clock.
- The other functions, however, do not operate, because clock supply to the CAN Controller is stopped, which remains in sleep mode.
- The CPU, in response to INTCnWUP,
  - releases its power saving mode,
  - resumes supply of the internal clocks - including the clock to the CAN Controller - after the oscillation stabilization time has elapsed, and
  - starts instruction execution.
- The CAN Controller is immediately released from the sleep mode when clock supply is resumed, and returns to the normal operation mode (FCNnCMCLCTL.FCNnCMCLMDPF[1:0] = $00_B$).

## 18.12 Interrupt Function

The CAN Controller provides 6 different interrupt sources.

The occurrence of these interrupt sources is stored in interrupt status registers. Four separate interrupt request signals are generated from the six interrupt sources. When an interrupt request signal that corresponds to two or more interrupt sources is generated, the interrupt sources can be identified by using an interrupt status register. After an interrupt source has occurred, the corresponding interrupt status bit must be cleared to 0 by software.

**Table 18-21 List of CAN Controller interrupt sources**

| No. | Interrupt status bit FCNnCMISCTL. | Interrupt enable bit FCNnCMIECTL. [a] | Interrupt request signal | Interrupt source description |
|---|---|---|---|---|
| 1 | FCNnCMISITSF0 | FCNnCMIESEIE0 | INTCnTRX | Message frame successfully transmitted from message buffer m |
| 2 | FCNnCMISITSF1 | FCNnCMIESEIE1 | INTCnREC | Valid message frame reception in message buffer m |
| 3 | FCNnCMISITSF2 | FCNnCMIESEIE2 | INTCnERR | CAN Controller error state interrupt<br>• This interrupt is generated when the transmission/reception error counter is at the warning level, or in the error passive or bus-off state. |
| 4 | FCNnCMISITSF3 | FCNnCMIESEIE3 | | CAN Controller protocol error interrupt<br>• This interrupt is generated when a stuff error, form error, ACK error, bit error, or CRC error occurs. |
| 5 | FCNnCMISITSF4 | FCNnCMIESEIE4 | | CAN Controller arbitration loss interrupt |
| 6 | FCNnCMISITSF5 | FCNnCMIESEIE5 | INTCnWUP | CAN Controller wakeup interrupt from CAN Controller sleep mode<br>• This interrupt is generated when the CAN Controller is woken up from the sleep mode because a falling edge is detected at the CAN Controller reception pin (CAN bus transition from recessive to dominant). |
| 7 | FCNnCMISITSF6 | FCNnCMIESEIE6 | | CAN Controller transmit abort interrupt status<br>• This interrupt is generated when the abortion of a transmission was successful (aborted message was not sent). |

a) The message buffer interrupt enable bit FCNnMmCTL.FCNnMmIENF of the corresponding message buffer has to be set to 1 for that message buffer to participate in the interrupt generation process.

## 18.13 Diagnosis Functions and Special Operational Modes

The CAN Controller provides a receive-only mode, single-shot mode, and self-test mode to support CAN bus diagnosis functions or the operation of special CAN communication methods.

### 18.13.1 Receive-only mode

The receive-only mode is used to monitor receive messages without causing any interference on the CAN bus and can be used for CAN bus analysis nodes.

For example, this mode can be used for automatic baud-rate detection. The baudrate in the CAN Controller is changed until “valid reception” is detected, so that the baudrates are matching (“valid reception” means a message frame has been received in the CAN protocol layer without occurrence of an error and with an appropriate ACK between nodes connected to the CAN bus). A valid reception does not require message frames to be stored in a receive message buffer (data frames) or transmit message buffer (remote frames). The event of valid reception is indicated by setting FCNnCMCLCTL.FCNnCMCLVALF = 1.

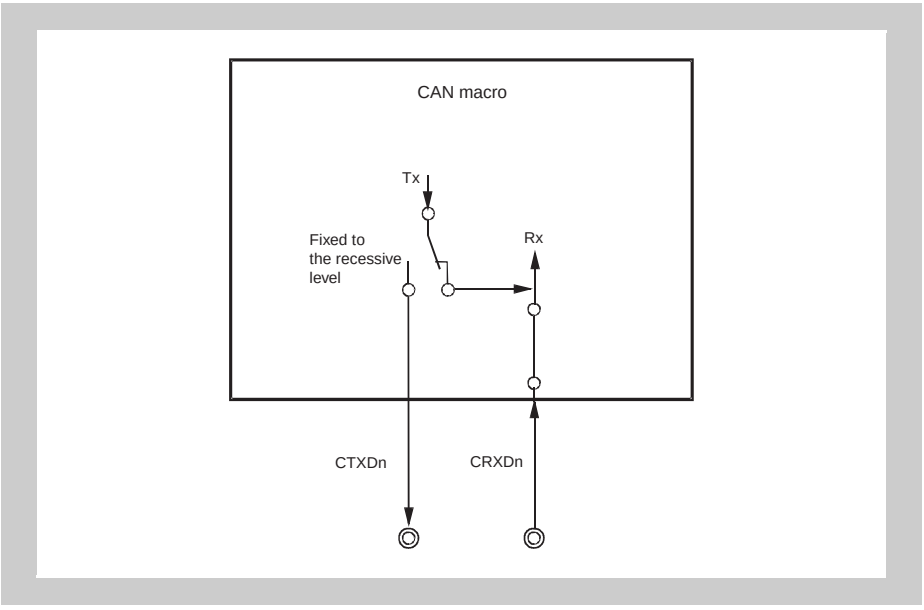



**Figure 18-12 CAN Controller terminal connection in receive-only mode**

In the receive-only mode, no message frames can be transmitted from the CAN Controller to the CAN bus. Transmit requests issued for message buffers defined as transmit message buffers are held pending.

In the receive-only mode, the CAN Controller transmission pin CTXDn in the CAN Controller is fixed to the recessive level. Therefore, no active error flag can be transmitted from the CAN Controller to the CAN bus even when a CAN bus error is detected while receiving a message frame. Since no transmission can be issued from the CAN Controller, the transmission error counter the FCNnCMERCNT.TEC7 to FCNnCMERCNT.TEC0 bits are never updated. Therefore, a CAN Controller in the receive-only mode does not enter the bus-off state.

Furthermore, in the receive-only mode ACK is not returned to the CAN bus in this mode upon the valid reception of a message frame. Internally, the local node recognizes that it has transmitted ACK. An overload frame cannot be transmitted to the CAN bus.

**Caution** If only two CAN nodes are connected to the CAN bus and one of them is operating in the receive-only mode, there is no ACK on the CAN bus. Due to the missing ACK, the transmitting node will transmit an active error flag, and repeat transmitting a message frame. The transmitting node becomes error passive after transmitting the message frame 16 times (assuming that the error counter was 0 in the beginning and no other errors have occurred). After the message frame for the 17th time is transmitted, the transmitting node generates a passive error flag. The receiving node in the receive-only mode detects the first valid message frame at this point, and the FCNnCMCLCTL.FCNnCMCLVALF bit is set to 1 for the first time.

### 18.13.2 Single-shot mode

In the single-shot mode, automatic re-transmission as defined in the CAN protocol is switched off. (According to the CAN protocol, a message frame transmission that has been aborted by either arbitration loss or error occurrence has to be repeated without control by software.) All other behavior of single shot mode is identical to normal operation mode. Features of single shot mode can not be used in combination with normal mode with ABT.

The single-shot mode disables the re-transmission of an aborted message frame transmission according to the setting of FCNnCMCLCTL.FCNnCMCLALBF. When FCNnCMCLALBF is cleared to 0, re-transmission upon arbitration loss and upon error occurrence is disabled. If FCNnCMCLALBF is set to 1, re-transmission upon error occurrence is disabled, but re-transmission upon arbitration loss is enabled. As a consequence, FCNnMmCTL.FCNnMmTRQF in a message buffer defined as a transmit message buffer is cleared to 0 by the following events:

- Successful transmission of the message frame
- Arbitration loss while sending the message frame
- Error occurrence while sending the message frame

The events arbitration loss and error occurrence can be distinguished by checking FCNnCMISCTL.FCNnCMISITSF4 and FCNnCMISCTL.FCNnCMISITSF3 respectively, and the type of the error can be identified by reading FCNnCMLCSTR.FCN0CMLCSSLC[2:0].

Upon successful transmission of the message frame, the transmit completion interrupt bit FCNnCMISCTL.FCNnCMISITSF0 is set to 1. If FCNnCMIECTL.FCNnCMIEINTF0 is set to 1 at this time, an interrupt request signal is output.

The single-shot mode can be used when emulating time-triggered communication methods (e.g., TTCAN level 1).

**Caution** FCNnCMCLCTL.FCNnCMCLALBF is only valid in single-shot mode. It does not influence the operation of re-transmission upon arbitration loss in the other operation modes.

### 18.13.3 Self-test mode

In the self-test mode, message frame transmission and message frame reception can be tested without connecting the CAN node to the CAN bus or without affecting the CAN bus.

In the self-test mode, the CAN Controller is completely disconnected from the CAN bus, but transmission and reception are internally looped back. The CAN Controller transmission pin CTXDn is fixed to the recessive level.

If the falling edge on the CAN Controller reception pin CRXDn is detected after the CAN Controller has entered the CAN Controller sleep mode from the self-test mode, however, the CAN Controller is released from sleep mode in the same manner as the other operation modes. To keep the CAN Controller in the sleep mode, use the CAN Controller reception pin CRXDn as a port pin.

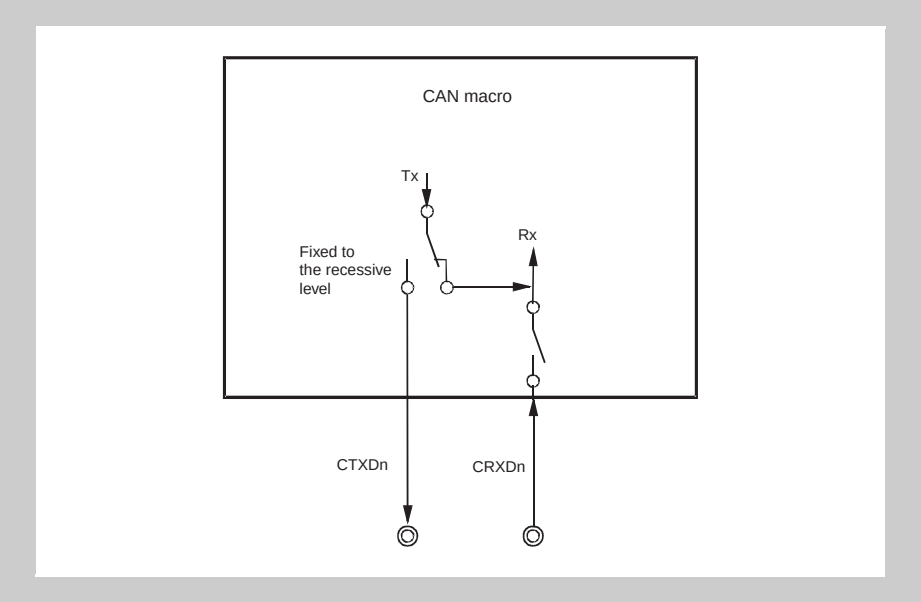



**Figure 18-13 CAN Controller terminal connection in self-test mode**

### 18.13.4 Receive/transmit operation in each operation mode

The following table shows outline of the receive/transmit operation in each operation mode.

**Table 18-22 Outline of the receive/transmit in each operation mode**

| Operation mode | Transmission of data/ remote frame | Transmission of ACK | Transmission of error/ overload frame | Transmission retry | Automatic block transmission (ABT) | Set of FCNnCMCLVALF bit | Store data to message buffer |
|---|---|---|---|---|---|---|---|
| Initialization mode | No | No | No | No | No | No | No |
| Normal operation mode | Yes | Yes | Yes | Yes | No | Yes | Yes |
| Normal operation mode with ABT | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Receive only mode | No | No | No | No | No | Yes | Yes |
| Single-shot mode | Yes | Yes | Yes | No[a] | No | Yes | Yes |
| Self-test mode | Yes[b] | Yes[b] | Yes[b] | Yes[b] | No | Yes[b] | Yes[b] |

a) When the arbitration lost occurs, control of re-transmission is possible by FCNnCMCLCTL.FCNnCMCLALBF.
b) Each signals are not generated to outside, but generated into the CAN Controller.

## 18.14 Time Stamp Function

CAN is an asynchronous, serial protocol. All nodes connected to the CAN bus have a local, autonomous clock. As a consequence, the clocks of the nodes have no relation (i.e., the clocks are asynchronous and may have different frequencies).

In some applications, however, a common time base over the network (= global time base) is needed. In order to build up a global time base, a time stamp function is used. The essential mechanism of a time stamp function is the capture of timer values triggered by signals on the CAN bus.

### 18.14.1 Time stamp function

The CAN Controller supports the capturing of timer values triggered by a specific frame. An on-chip 16-bit capture timer unit of the microcontroller is used in addition to the CAN Controller. The 16-bit capture timer unit captures the timer value according to a trigger signal (TSOUT) for capturing that is output when a data frame is received from the CAN Controller. The CPU can retrieve the time of occurrence of the capture event, i.e., the time stamp of the message received from the CAN bus, by reading the captured value. The TSOUT signal can be selected from the following two event sources and is specified by FCNnCMTSCTL.FCNnCMTSSELE.

- SOF event (start of frame)
  (FCNnCMTSCTL.FCNnCMTSSELE = 0)
- EOF event (last bit of end of frame)
  (FCNnCMTSCTL.FCNnCMTSSELE = 1)

The TSOUT signal is enabled by setting FCNnCMTSCTL.FCNnCMTSTSGE = 1.

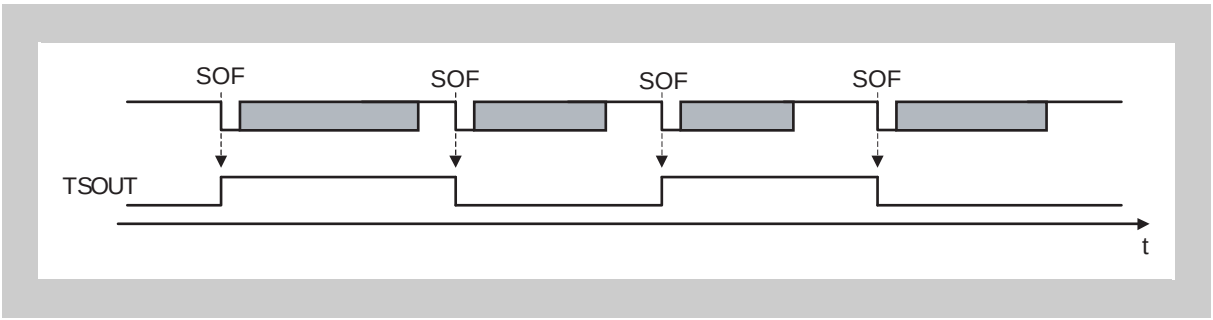



**Figure 18-14 Timing diagram of capture signal TSOUT**

The TSOUT signal toggles its level upon occurrence of the selected event during data frame reception (in *Figure 18-14 “Timing diagram of capture signal TSOUT”*, the SOF is used as the trigger event source). To capture a timer value by using the TSOUT signal, the capture timer unit must detect the capture signal at both the rising edge and falling edge.

This time stamp function is controlled by the FCNnCMTSLOKE bit of the FCNnCMTSCTL register. When FCNnCMTSLOKE is cleared to 0, the TSOUT signal toggles upon occurrence of the selected event. If FCNnCMTSLOKE is set to 1, the TSOUT signal toggles upon occurrence of the selected event, but the toggle is stopped as FCNnCMTSCTL.FCNnCMTSTSGE is automatically cleared to 0 as soon as the message storing to the message buffer 0 starts. This suppresses the subsequent toggle occurrence by the TSOUT signal, so

that the time stamp value toggled last (= captured last) can be saved as the time stamp value of the time at which the data frame was received in message buffer 0.

**Caution** The time stamp function using the FCNnCMTSLOKE bit stops toggle of the TSOUT signal by receiving a data frame in message buffer 0. Toggle of the TSOUT signal does not stop when a data frame is received in a message buffer other than message buffer 0.
A data frame cannot be received in message buffer 0 when the CAN Controller is in the normal operation mode with ABT, because message buffer 0 must be set as a transmit message buffer.
In this operation mode, therefore, the function to stop toggle of the TSOUT signal by the FCNnCMTSLOKE bit cannot be used.

## 18.15 Baudrate Settings

### 18.15.1 Baudrate setting conditions

Make sure that the settings are within the range of limit values for ensuring correct operation of the CAN Controller, as follows.

- 5 TQ ≤ SPT (sampling point) ≤ 17 TQ

  SPT = TSEG1 + 1
- 8 TQ ≤ DBT (data bit time) ≤ 25 TQ

  DBT = TSEG1 + TSEG2 + 1 TQ = TSEG2 + SPT
- 1 TQ ≤ SJW (synchronization jump width) ≤ 4 TQ

  SJW ≤ DBT – SPT
- 4 ≤ TSEG1 ≤ 16
- 1 ≤ TSEG2 ≤ 8

**Notes**
1. TQ = $1/f_{TQ}$ ($f_{TQ}$: CAN protocol layer clock)
2. The values TSEG1, TSEG2 and SJW are defined by following register bits:
   TSEG1 = FCNnCMBTCTL.FCNnCMBTS1LG[3:0] + 1
   TSEG2 = FCNnCMBTCTL.FCNnCMBTS2LG[2:0] + 1
   SJW = FCNnCMBTCTL.FCNnCMBTJWLG[1:0] + 1

*Table 18-23 “Settable bit rate combinations”* shows the combinations of bit rates that satisfy the above conditions.

**Table 18-23 Settable bit rate combinations (1/3)**

| Valid bit rate setting | | | | | FCNnCMBTCTL register setting value | | Sampling point (unit %) |
|---|---|---|---|---|---|---|---|
| DBT length | SYNC SEGMENT | PROP SEGMENT | PHASE SEGMENT1 | PHASE SEGMENT2 | FCNnCMBTS1LG[3:0] | FCNnCMBTS2LG[2:0] | |
| 25 | 1 | 8 | 8 | 8 | 1111 | 111 | 68.0 |
| 24 | 1 | 7 | 8 | 8 | 1110 | 111 | 66.7 |
| 24 | 1 | 9 | 7 | 7 | 1111 | 110 | 70.8 |
| 23 | 1 | 6 | 8 | 8 | 1101 | 111 | 65.2 |
| 23 | 1 | 8 | 7 | 7 | 1110 | 110 | 69.6 |
| 23 | 1 | 10 | 6 | 6 | 1111 | 101 | 73.9 |
| 22 | 1 | 5 | 8 | 8 | 1100 | 111 | 63.6 |
| 22 | 1 | 7 | 7 | 7 | 1101 | 110 | 68.2 |
| 22 | 1 | 9 | 6 | 6 | 1110 | 101 | 72.7 |
| 22 | 1 | 11 | 5 | 5 | 1111 | 100 | 77.3 |
| 21 | 1 | 4 | 8 | 8 | 1011 | 111 | 61.9 |
| 21 | 1 | 6 | 7 | 7 | 1100 | 110 | 66.7 |
| 21 | 1 | 8 | 6 | 6 | 1101 | 101 | 71.4 |
| 21 | 1 | 10 | 5 | 5 | 1110 | 100 | 76.2 |
| 21 | 1 | 12 | 4 | 4 | 1111 | 011 | 81.0 |
| 20 | 1 | 3 | 8 | 8 | 1010 | 111 | 60.0 |
| 20 | 1 | 5 | 7 | 7 | 1011 | 110 | 65.0 |
| 20 | 1 | 7 | 6 | 6 | 1100 | 101 | 70.0 |
| 20 | 1 | 9 | 5 | 5 | 1101 | 100 | 75.0 |
| 20 | 1 | 11 | 4 | 4 | 1110 | 011 | 80.0 |
| 20 | 1 | 13 | 3 | 3 | 1111 | 010 | 85.0 |
| 19 | 1 | 2 | 8 | 8 | 1001 | 111 | 57.9 |
| 19 | 1 | 4 | 7 | 7 | 1010 | 110 | 63.2 |
| 19 | 1 | 6 | 6 | 6 | 1011 | 101 | 68.4 |
| 19 | 1 | 8 | 5 | 5 | 1100 | 100 | 73.7 |
| 19 | 1 | 10 | 4 | 4 | 1101 | 011 | 78.9 |
| 19 | 1 | 12 | 3 | 3 | 1110 | 010 | 84.2 |
| 19 | 1 | 14 | 2 | 2 | 1111 | 001 | 89.5 |
| 18 | 1 | 1 | 8 | 8 | 1000 | 111 | 55.6 |
| 18 | 1 | 3 | 7 | 7 | 1001 | 110 | 61.1 |
| 18 | 1 | 5 | 6 | 6 | 1010 | 101 | 66.7 |
| 18 | 1 | 7 | 5 | 5 | 1011 | 100 | 72.2 |
| 18 | 1 | 9 | 4 | 4 | 1100 | 011 | 77.8 |
| 18 | 1 | 11 | 3 | 3 | 1101 | 010 | 83.3 |
| 18 | 1 | 13 | 2 | 2 | 1110 | 001 | 88.9 |
| 18 | 1 | 15 | 1 | 1 | 1111 | 000 | 94.4 |
| 17 | 1 | 2 | 7 | 7 | 1000 | 110 | 58.8 |

Table 18-23 Settable bit rate combinations (2/3)

| Valid bit rate setting | | | | | FCNnCMBTCTL register setting value | | Sampling point (unit %) |
|---|---|---|---|---|---|---|---|
| DBT length | SYNC SEGMENT | PROP SEGMENT | PHASE SEGMENT1 | PHASE SEGMENT2 | FCNnCMBTS1LG[3:0] | FCNnCMBTS2LG[2:0] | |
| 17 | 1 | 4 | 6 | 6 | 1001 | 101 | 64.7 |
| 17 | 1 | 6 | 5 | 5 | 1010 | 100 | 70.6 |
| 17 | 1 | 8 | 4 | 4 | 1011 | 011 | 76.5 |
| 17 | 1 | 10 | 3 | 3 | 1100 | 010 | 82.4 |
| 17 | 1 | 12 | 2 | 2 | 1101 | 001 | 88.2 |
| 17 | 1 | 14 | 1 | 1 | 1110 | 000 | 94.1 |
| 16 | 1 | 1 | 7 | 7 | 0111 | 110 | 56.3 |
| 16 | 1 | 3 | 6 | 6 | 1000 | 101 | 62.5 |
| 16 | 1 | 5 | 5 | 5 | 1001 | 100 | 68.8 |
| 16 | 1 | 7 | 4 | 4 | 1010 | 011 | 75.0 |
| 16 | 1 | 9 | 3 | 3 | 1011 | 010 | 81.3 |
| 16 | 1 | 11 | 2 | 2 | 1100 | 001 | 87.5 |
| 16 | 1 | 13 | 1 | 1 | 1101 | 000 | 93.8 |
| 15 | 1 | 2 | 6 | 6 | 0111 | 101 | 60.0 |
| 15 | 1 | 4 | 5 | 5 | 1000 | 100 | 66.7 |
| 15 | 1 | 6 | 4 | 4 | 1001 | 011 | 73.3 |
| 15 | 1 | 8 | 3 | 3 | 1010 | 010 | 80.0 |
| 15 | 1 | 10 | 2 | 2 | 1011 | 001 | 86.7 |
| 15 | 1 | 12 | 1 | 1 | 1100 | 000 | 93.3 |
| 14 | 1 | 1 | 6 | 6 | 0110 | 101 | 57.1 |
| 14 | 1 | 3 | 5 | 5 | 0111 | 100 | 64.3 |
| 14 | 1 | 5 | 4 | 4 | 1000 | 011 | 71.4 |
| 14 | 1 | 7 | 3 | 3 | 1001 | 010 | 78.6 |
| 14 | 1 | 9 | 2 | 2 | 1010 | 001 | 85.7 |
| 14 | 1 | 11 | 1 | 1 | 1011 | 000 | 92.9 |
| 13 | 1 | 2 | 5 | 5 | 0110 | 100 | 61.5 |
| 13 | 1 | 4 | 4 | 4 | 0111 | 011 | 69.2 |
| 13 | 1 | 6 | 3 | 3 | 1000 | 010 | 76.9 |
| 13 | 1 | 8 | 2 | 2 | 1001 | 001 | 84.6 |
| 13 | 1 | 10 | 1 | 1 | 1010 | 000 | 92.3 |
| 12 | 1 | 1 | 5 | 5 | 0101 | 100 | 58.3 |
| 12 | 1 | 3 | 4 | 4 | 0110 | 011 | 66.7 |
| 12 | 1 | 5 | 3 | 3 | 0111 | 010 | 75.0 |

**Table 18-23 Settable bit rate combinations (3/3)**

| Valid bit rate setting | | | | | FCNnCMBTCTL register setting value | | Sampling point (unit %) |
|---|---|---|---|---|---|---|---|
| DBT length | SYNC SEGMENT | PROP SEGMENT | PHASE SEGMENT1 | PHASE SEGMENT2 | FCNnCMBTS1LG[3:0] | FCNnCMBTS2LG[2:0] | |
| 12 | 1 | 7 | 2 | 2 | 1000 | 001 | 83.3 |
| 12 | 1 | 9 | 1 | 1 | 1001 | 000 | 91.7 |
| 11 | 1 | 2 | 4 | 4 | 0101 | 011 | 63.6 |
| 11 | 1 | 4 | 3 | 3 | 0110 | 010 | 72.7 |
| 11 | 1 | 6 | 2 | 2 | 0111 | 001 | 81.8 |
| 11 | 1 | 8 | 1 | 1 | 1000 | 000 | 90.9 |
| 10 | 1 | 1 | 4 | 4 | 0100 | 011 | 60.0 |
| 10 | 1 | 3 | 3 | 3 | 0101 | 010 | 70.0 |
| 10 | 1 | 5 | 2 | 2 | 0110 | 001 | 80.0 |
| 10 | 1 | 7 | 1 | 1 | 0111 | 000 | 90.0 |
| 9 | 1 | 2 | 3 | 3 | 0100 | 010 | 66.7 |
| 9 | 1 | 4 | 2 | 2 | 0101 | 001 | 77.8 |
| 9 | 1 | 6 | 1 | 1 | 0110 | 000 | 88.9 |
| 8 | 1 | 1 | 3 | 3 | 0011 | 010 | 62.5 |
| 8 | 1 | 3 | 2 | 2 | 0100 | 001 | 75.0 |
| 8 | 1 | 5 | 1 | 1 | 0101 | 000 | 87.5 |
| 7[a] | 1 | 2 | 2 | 2 | 0011 | 001 | 71.4 |
| 7[a] | 1 | 4 | 1 | 1 | 0100 | 000 | 85.7 |
| 6[a] | 1 | 1 | 2 | 2 | 0010 | 001 | 66.7 |
| 6[a] | 1 | 3 | 1 | 1 | 0011 | 000 | 83.3 |
| 5[a] | 1 | 2 | 1 | 1 | 0010 | 000 | 80.0 |
| 4[a] | 1 | 1 | 1 | 1 | 0001 | 000 | 75.0 |

a) Setting with a DBT value of 7 or less is valid only when the value of the FCNnCMBRPRS register is other than $00_H$.

**Caution** The values in *Table 18-23 "Settable bit rate combinations"* do not guarantee the operation of the network. Thoroughly check the effect on the network, taking into consideration oscillation errors and delays of the CAN bus and CAN transceiver.

### 18.15.2 Clock prescaler and baudrate generator settings

In order to achieve the desired baudrate, time quanta, respectively data bit rate, combinations, described in the section above, the clock prescaler and the baudrate generator must be set up properly.

The diagram belows shows the clocks and how they are generated.

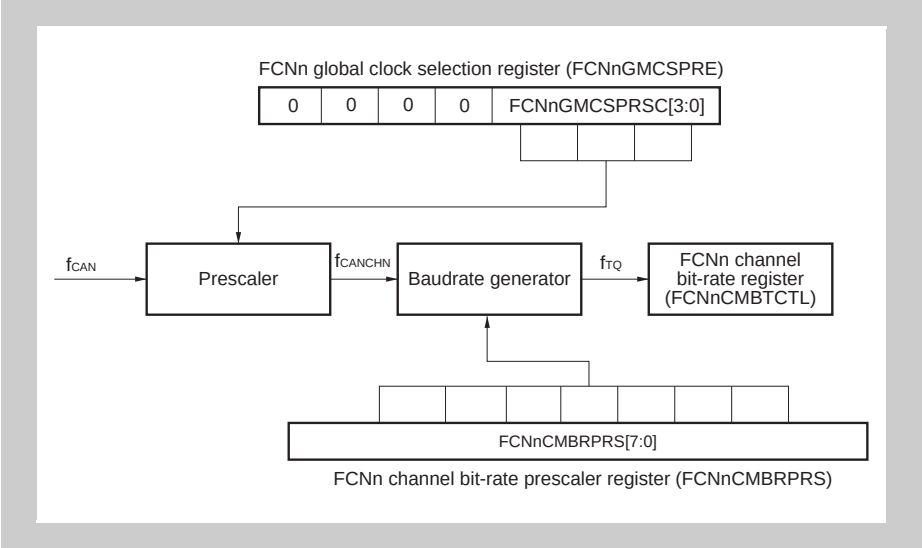



**Figure 18-15 CAN Controller clocks**

The setting of the global clock selection register bits FCNnGMCSPRE.FCNnCSPRSC[3:0] for the desired CAN channel clock $f_{CANCHN}$ is calculated as follows:

$$FCNnGMCSPRSC = \frac{f_{CAN}}{f_{CANCHN}} - 1$$

$f_{CAN}$: clock supplied to CAN Controller (unit: Hz)

$f_{CANCHN}$: CAN channel clock (unit: Hz)

Following conditions must be met:

- $0 \leq FCNnCSPRSC \leq 15$
- FCNnCSPRSC defines an integer value with no remainder.

The setting of the channel bit rate prescaler register FCNnCMBRPRS for the desired data bit time DBT and the baudrate is calculated as follows:

$$FCNnCMBRPRS = \frac{f_{CANCHN}}{DBT \bullet BRT} - 1$$

$f_{CANCHN}$: CAN channel clock (unit: Hz)

DBT: Data bit time (unit: 1/bit)

BRT: Baudrate (unit: bit/s)

If the baudrate is given, $f_{CANCHN}$ and DBT must be selected such, that following conditions are met:

- $0 \leq$ FCNnCMBRPRS $\leq 15$
- FCNnCMBRPRS defines an integer value with no remainder.
- $8 \leq$ DBT $\leq 25$
- The restrictions in *Table 18-23 “Settable bit rate combinations” on page 994* have to be obeyed.
- Restrictions of DBT selection and baudrate depending on clock $f_{CAN}$ have to be obeyed. Refer to the section *“CAN baudrate and time quanta”* above in this chapter.

## 18.16 Operation of the CAN Controller

The processing procedure for showing in this chapter is recommended processing procedure to operate CAN Controller.

Develop the program referring to recommended processing procedure in this chapter.

### 18.16.1 Initialization

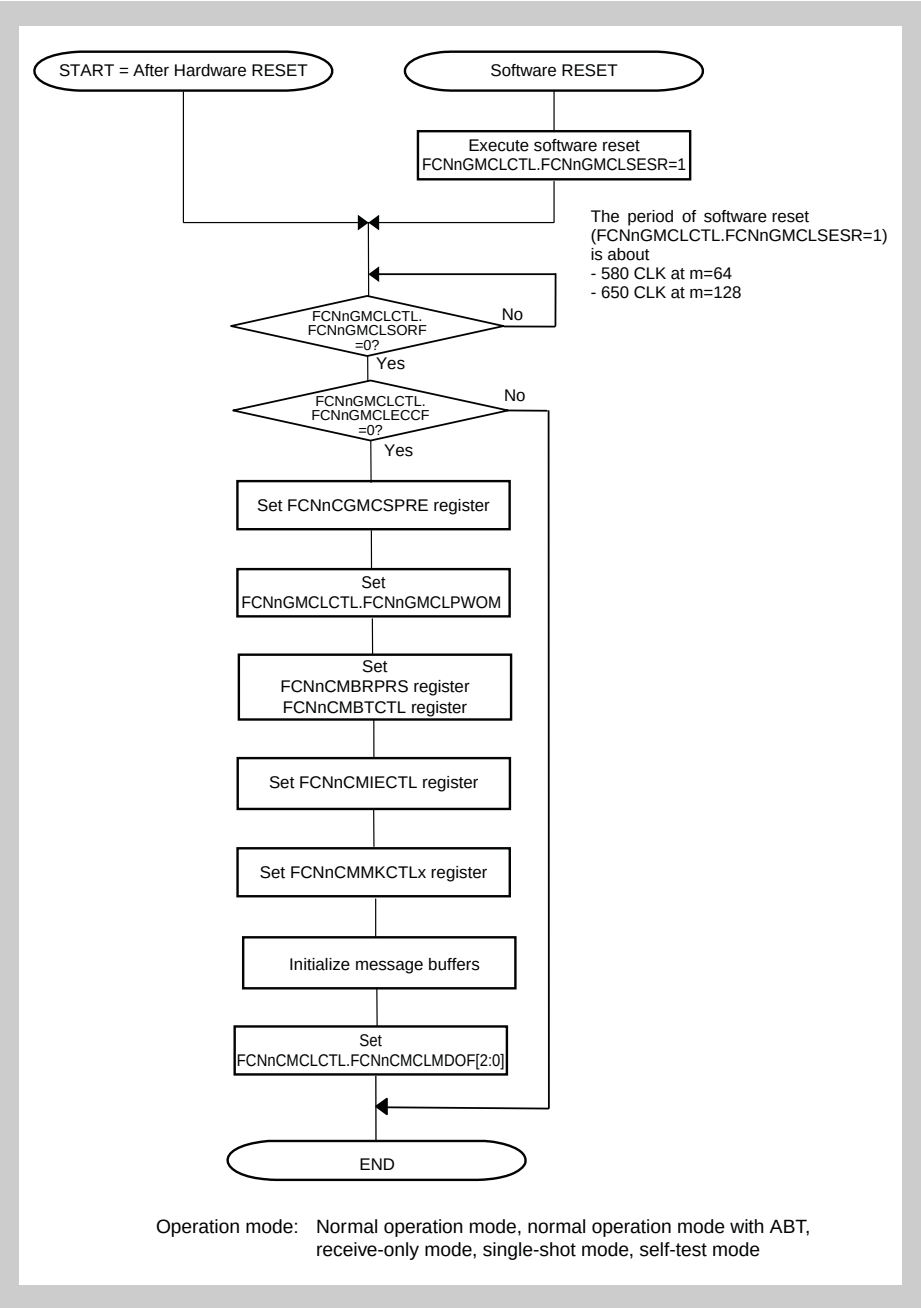



**Figure 18-16 Initialization**

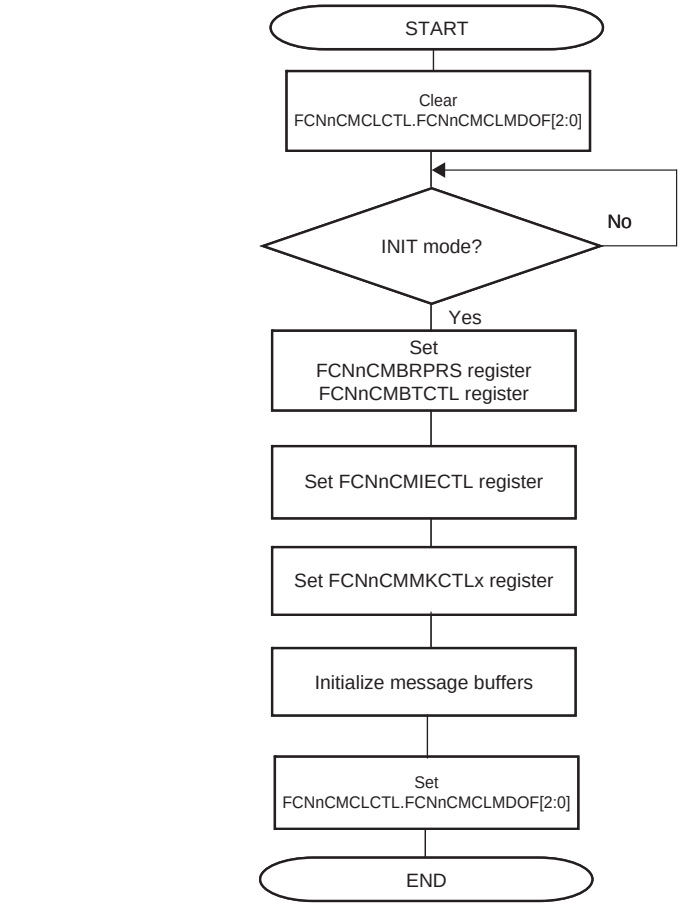



**Figure 18-17 Re-initialization without Software Reset function**

**Caution** When clearing the error counter in Re-initialization, set FCNnCMCLERCF bit only following conditions:

- After starting the CAN Controller (FCNnGMCLPWOM is set at FCNnGMCLPWOM=0), while in initialization mode.
- After aborting all transmit request according to *Figure 18-27 “Transmission abort processing (except normal operation mode with ABT)” on page 1011* or *Figure 18-28 “Transmission abort processing (normal operation mode with ABT) - Repeat option for aborted message” on page 1012* , while in initialization mode.

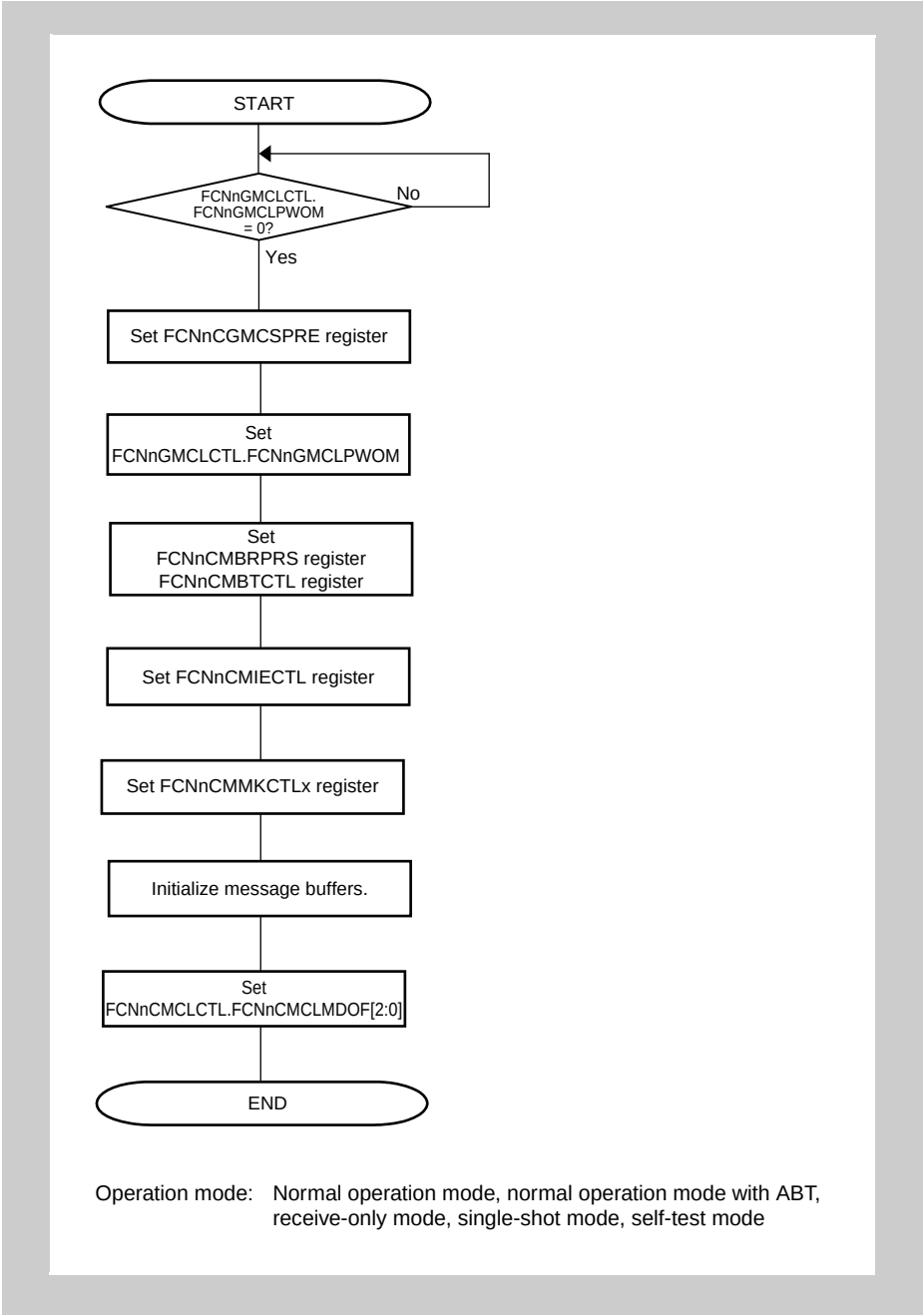



**Figure 18-18 Re-initialization with Software Reset function**

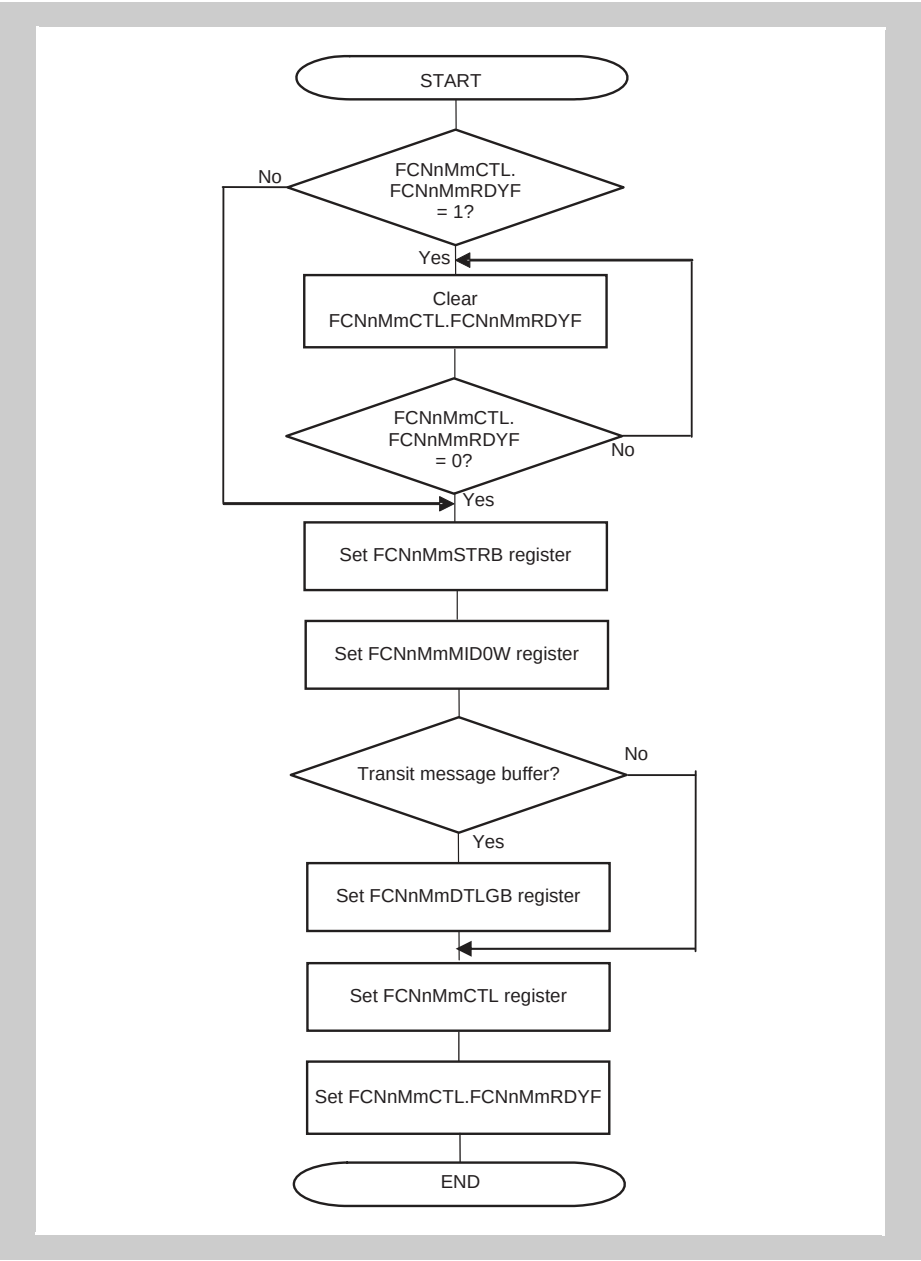



**Figure 18-19 Message buffer initialization**

**Cautions**

1. Before a message buffer is initialized, FCNnMmCTL.FCNnMmRDYF must be cleared.
2. Make the following settings for message buffers not used by the application.
   - Clear FCNnMmRDYF, FCNnMmTRQF, and FCNnMmDTNF bits of the FCNnMmCTL register to 0.
   - Clear FCNnMmSTRB.FCNnMmSSAM to 0.

*Figure 18-20 "Message buffer redefinition during reception"* shows the processing for a receive message buffer (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0001_B$ to $1000_B$).

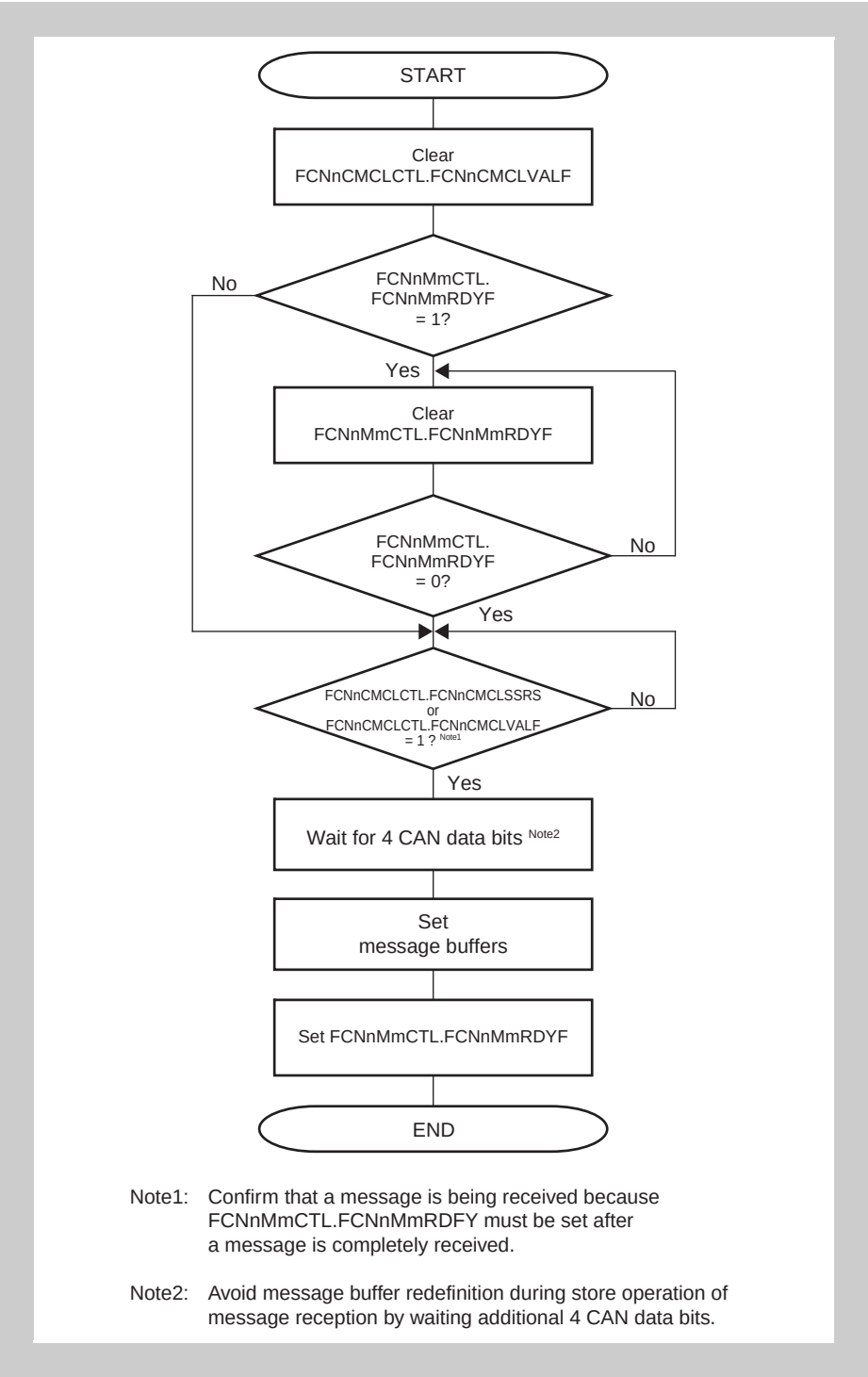



**Figure 18-20 Message buffer redefinition during reception**

*Figure 18-21 "Message buffer redefinition during transmission"* shows the processing for a transmit message buffer during transmission (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0000_B$).

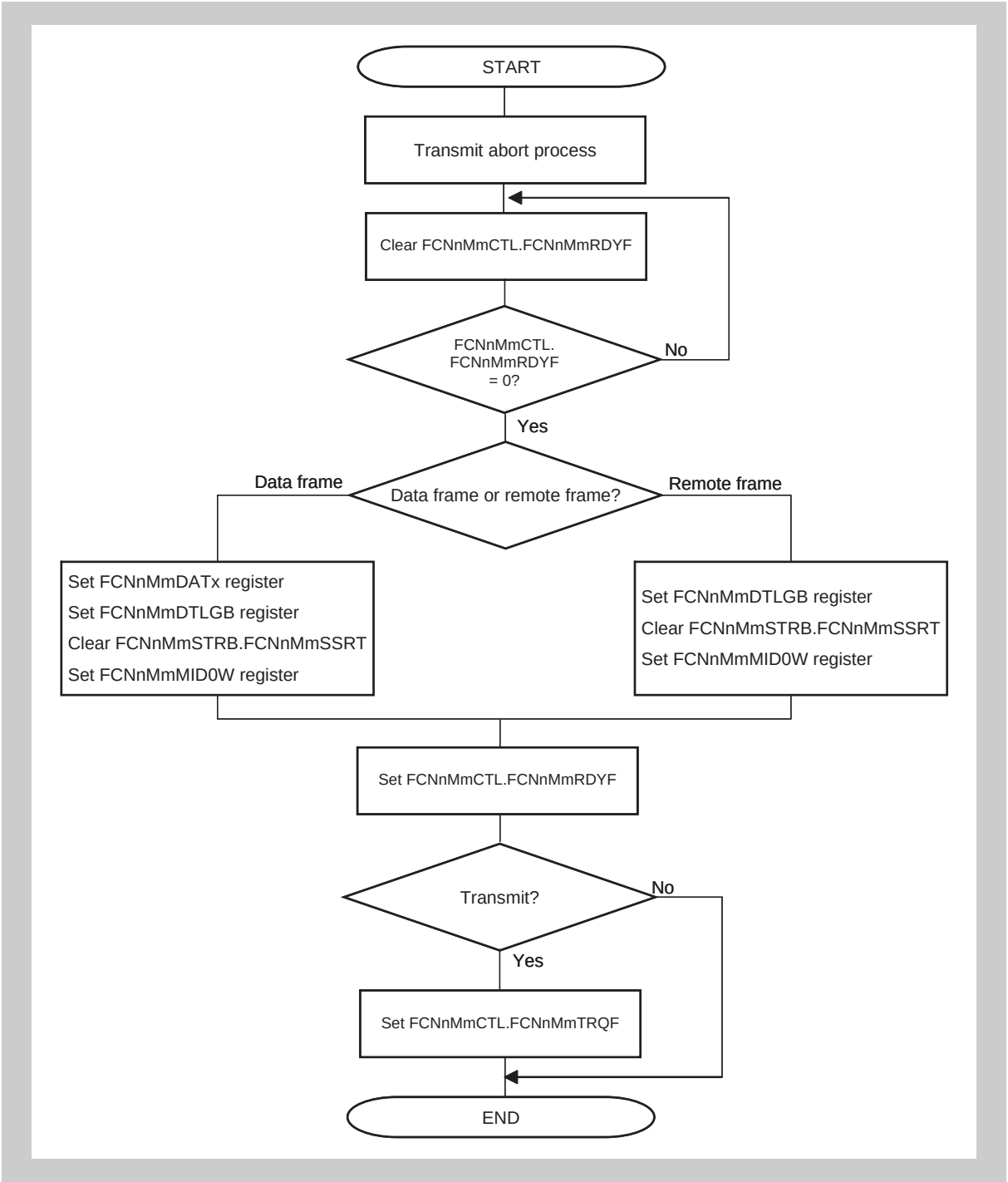



**Figure 18-21 Message buffer redefinition during transmission**

### 18.16.2 Message transmission

*Figure 18-22 "Message transmit processing"* shows the processing for a transmit message buffer (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0000_B$).

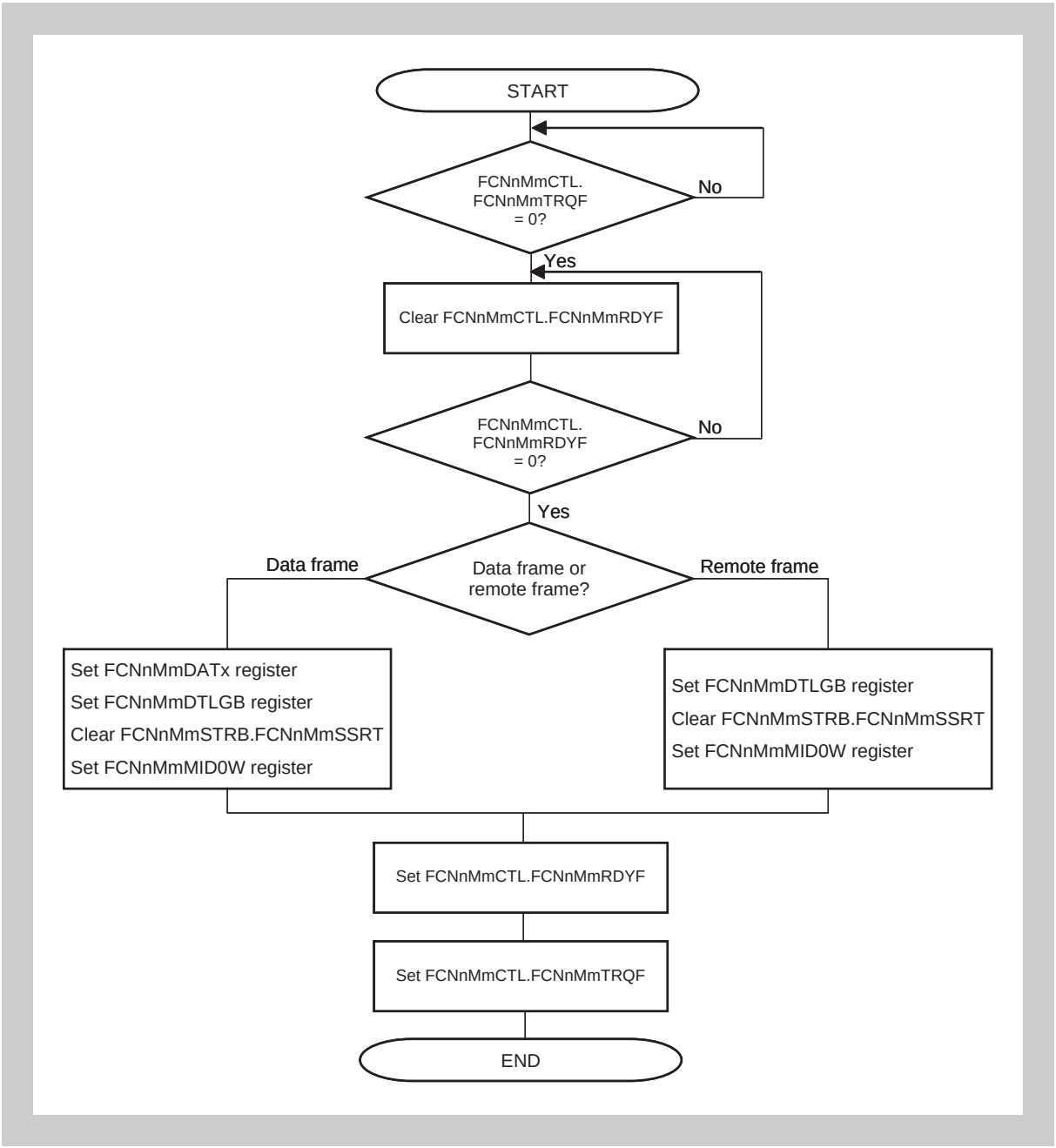



**Figure 18-22 Message transmit processing**

**Cautions**

1. FCNnMmCTL.FCNnMmTRQF should be set after FCNnMmCTL.FCNnMmRDYF is set.
2. FCNnMmCTL.FCNnMmRDYF and FCNnMmCTL.FCNnMmTRQF should not be set at the same time.

*Figure 18-23 "ABT message transmit processing"* shows the processing for a transmit message buffer (FCNnMmSTRB.FCNnMmSSMT[3:0] = $0000_B$)

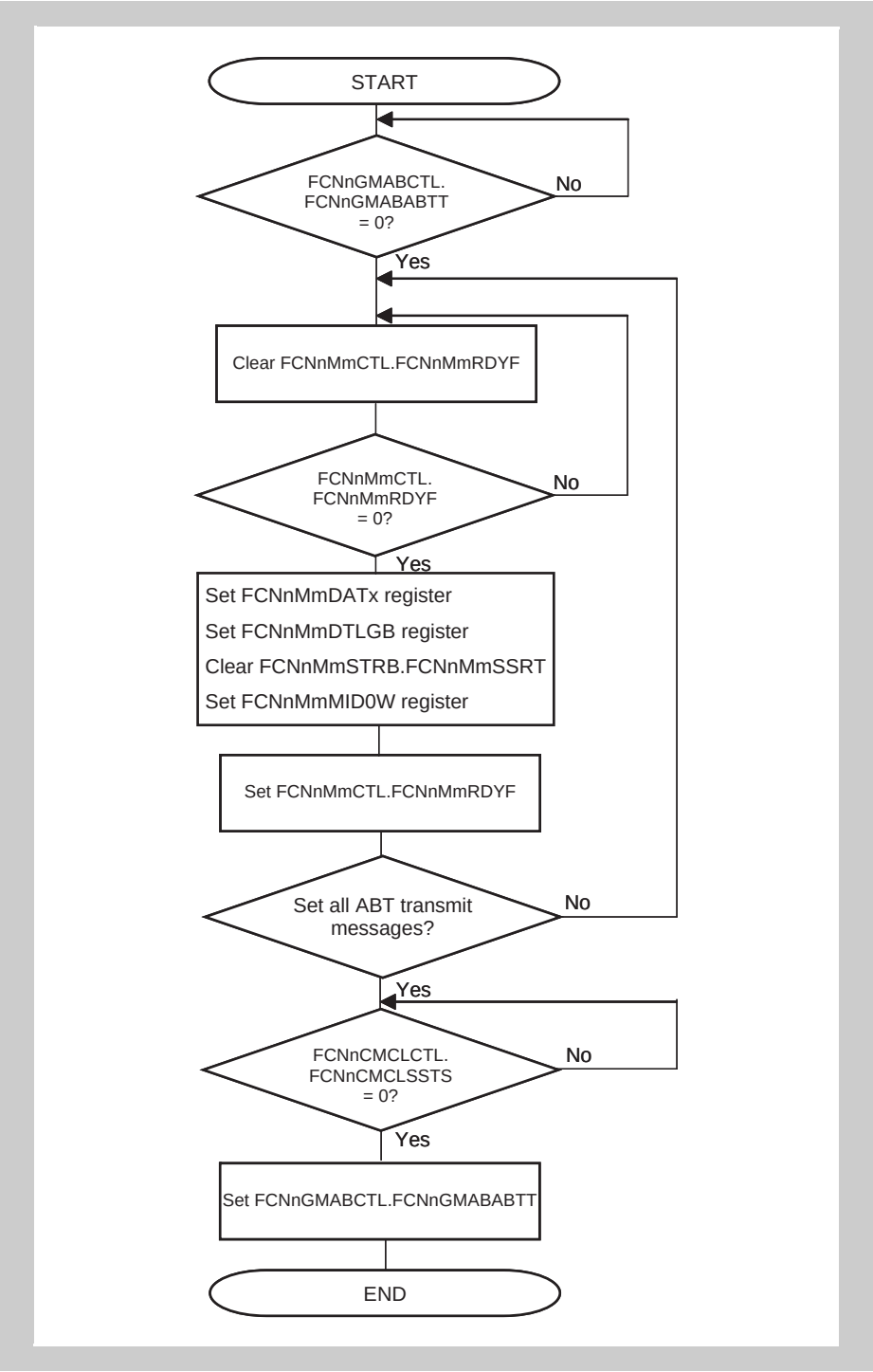



**Figure 18-23 ABT message transmit processing**

**Note** This processing (normal operation mode with ABT) can only be applied to message buffers usable with ABT mode. For message buffers other than the ABT message buffers, see *Figure 18-22 "Message transmit processing" on page 1005* .

**Caution** FCNnGMABCTL.FCNnGMABSEAT should be set to 1 after FCNnCMCLCTL.FCNnCMCLSSTS is cleared to 0. Checking FCNnCMCLCTL.FCNnCMCLSSTS and setting FCNnGMABCTL.FCNnGMABSEAT = 1 must be processed consecutively.

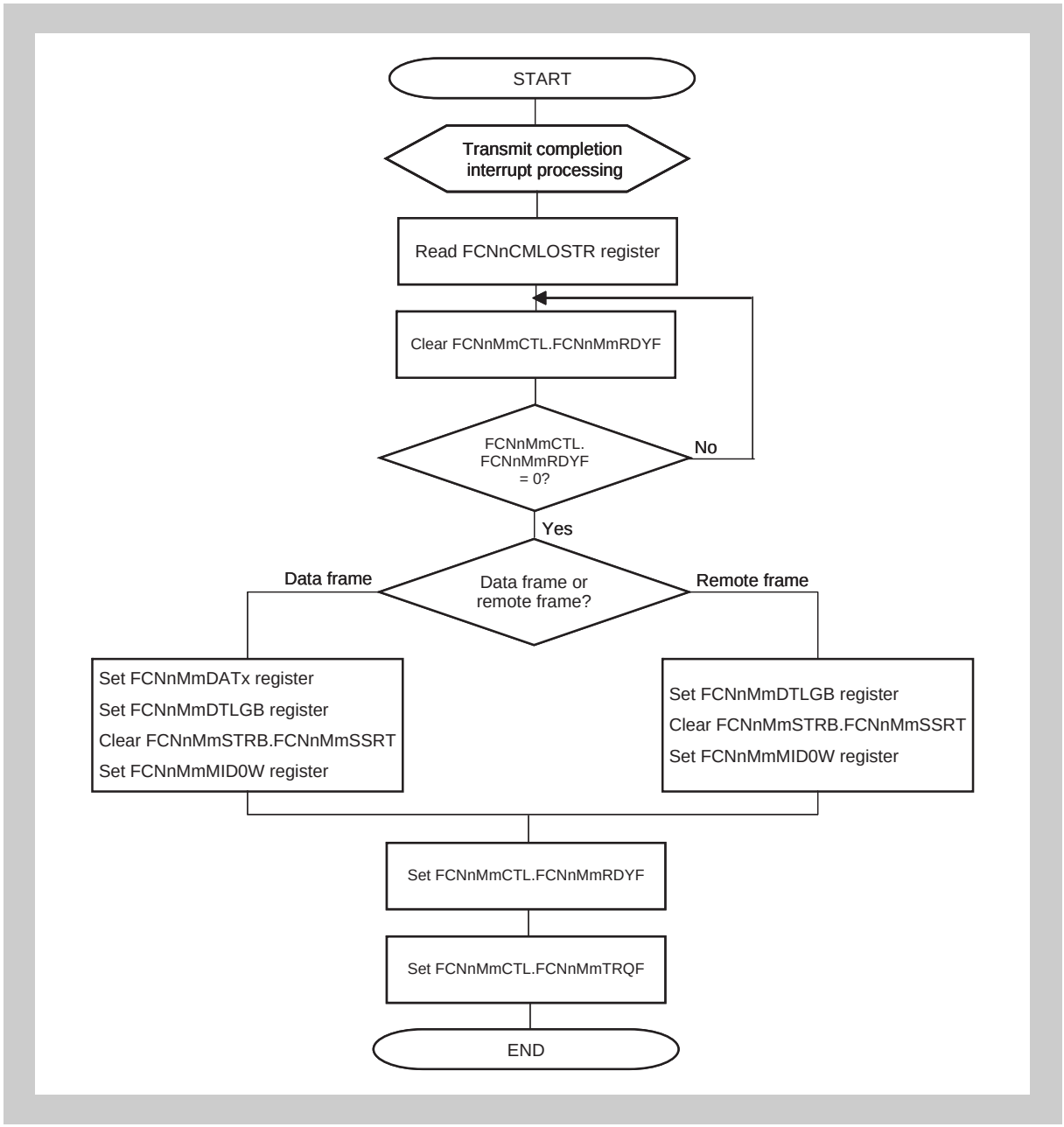



**Figure 18-24 Transmission via interrupt (using FCNnCMLOSTR register)**

**Cautions**

1. FCNnMmCTL.FCNnMmTRQF should be set after FCNnMmCTL.FCNnMmRDYF is set.
2. FCNnMmCTL.FCNnMmRDYF and FCNnMmCTL.FCNnMmTRQF should not be set at the same time.

**Note** Also check the FCNnGMCLSSMO flag at the beginning and at the end of the interrupt routine, in order to check the access to the message buffers as well as TX history list registers, in case a pending sleep mode had been executed. If FCNnGMCLSSMO is detected to be cleared at any check, the actions and results of the processing have to be discarded and processed again, after FCNnGMCLSSMO is set again.
It is recommended to cancel any sleep mode requests, before processing TX interrupts.

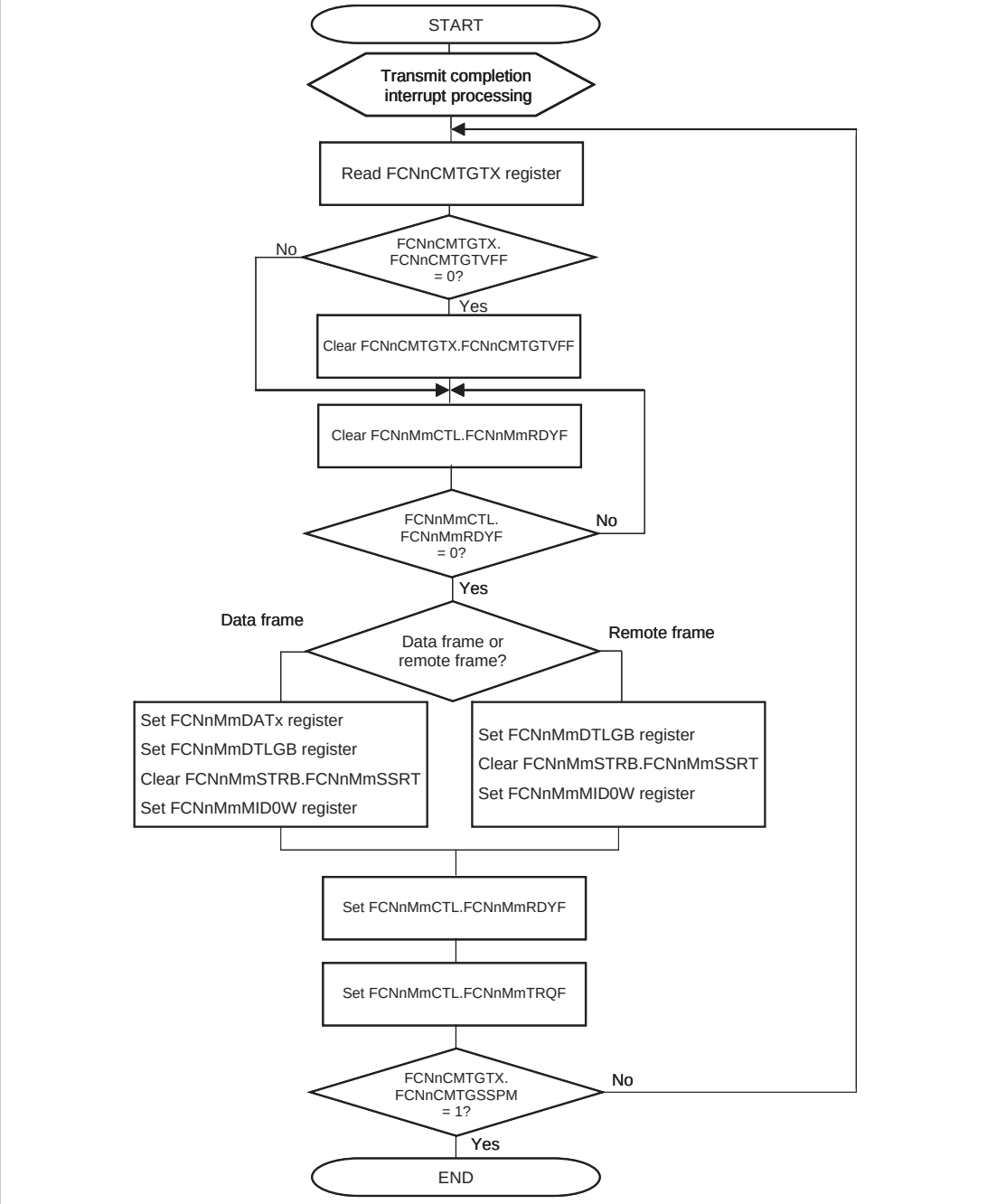



**Figure 18-25 Transmission via interrupt (using FCNnCMTGTX register)**

**Cautions** **1.** FCNnMmCTL.FCNnMmTRQF should be set after

FCNnMmCTL.FCNnMmRDYF is set.

2. FCNnMmCTL.FCNnMmRDYF and FCNnMmCTL.FCNnMmTRQF should not be set at the same time.

---

**Notes**

1. Also check the FCNnGMCLSSMO flag at the beginning and at the end of the interrupt routine, in order to check the access to the message buffers as well as TX history list registers, in case a pending sleep mode had been executed. If FCNnGMCLSSMO is detected to be cleared at any check, the actions and results of the processing have to be discarded and processed again, after FCNnGMCLSSMO is set again.
   It is recommended to cancel any sleep mode requests, before processing TX interrupts.
2. If FCNnCMTGTX.FCNnCMTGTVFF was set once, the transmit history list is inconsistent. Consider to scan all configured transmit buffers for completed transmissions.

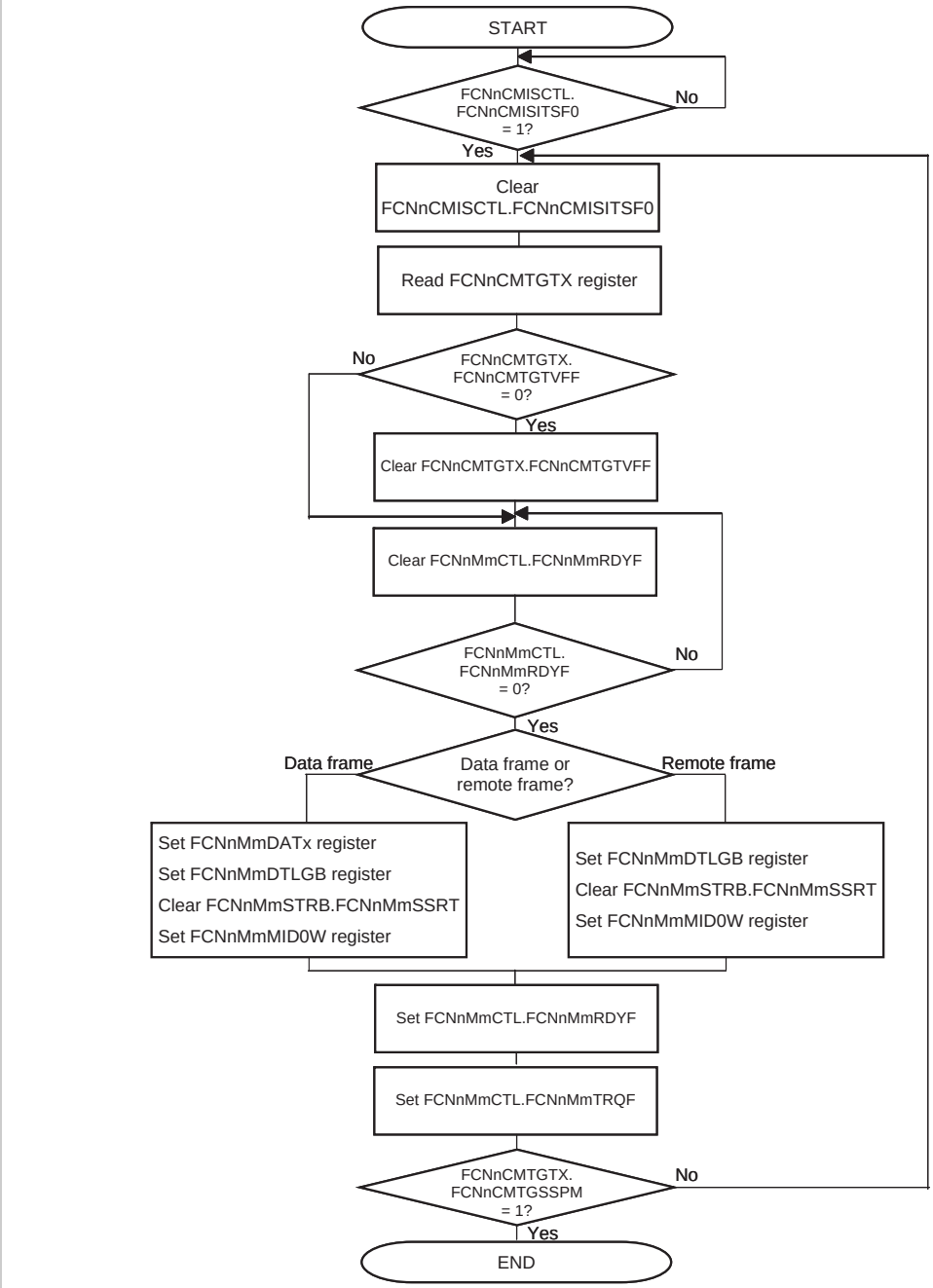



**Figure 18-26 Transmission via software polling**

**Cautions**

1. FCNnMmCTL.FCNnMmTRQF should be set after FCNnMmCTL.FCNnMmRDYF is set.
2. FCNnMmCTL.FCNnMmRDYF and FCNnMmCTL.FCNnMmTRQF should not be set at the same time.

**Notes**

1. Also check the FCNnGMCLSSMO flag at the beginning and at the end of the polling routine, in order to check the access to the message buffers as well as TX history list registers, in case a pending sleep mode had been executed. If FCNnGMCLSSMO is detected to be cleared at any check, the actions and results of the processing have to be discarded and processed again, after FCNnGMCLSSMO is set again.
2. If FCNnCMTGTX.FCNnCMTGTVFF was set once, the transmit history list is inconsistent. Consider to scan all configured transmit buffers for completed transmissions.

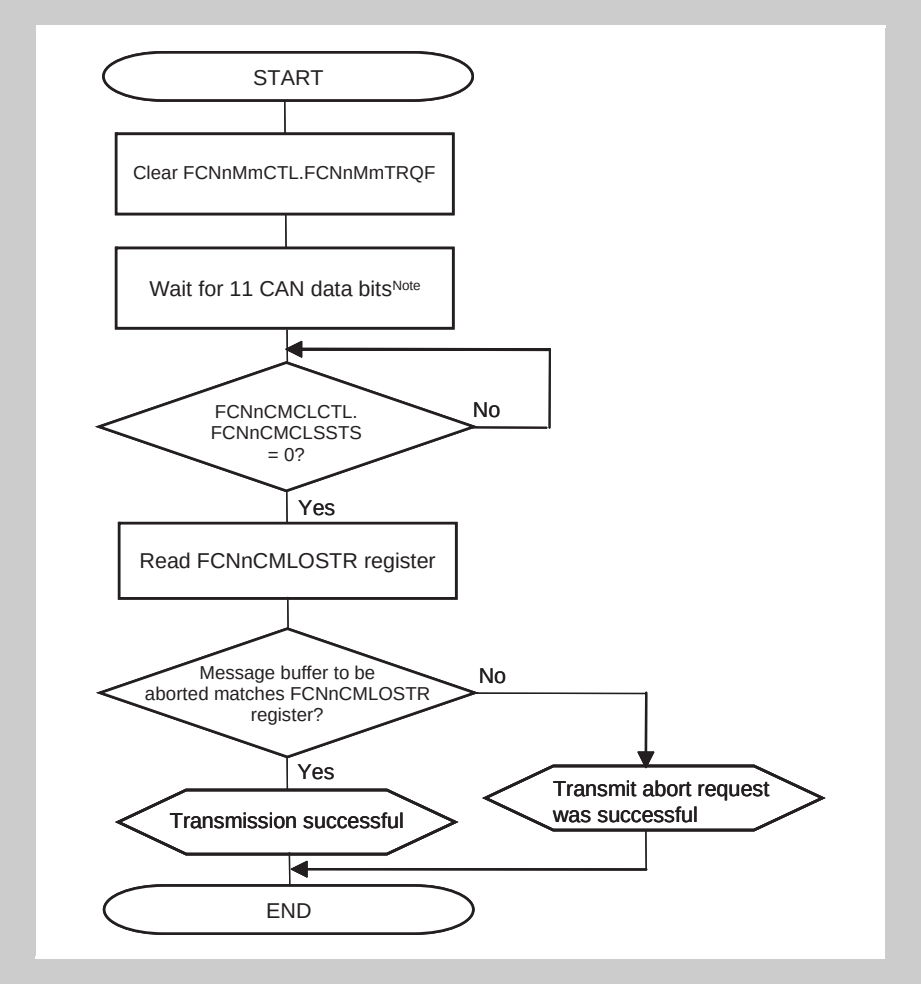



**Figure 18-27 Transmission abort processing (except normal operation mode with ABT)**

**Note** There is a possibility of starting the transmission without being aborted even if FCNnMmCTL.FCNnMmTRQF is cleared, because the transmission request to protocol layer might already been accepted between 11 bits, total of interframe space (3 bits) and suspend transmission (8 bits).

**Cautions**

1. Clear FCNnMmCTL.FCNnMmTRQF for aborting transmission request, not FCNnMmCTL.FCNnMmRDYF.
2. Before making a sleep mode transition request, confirm that there is no transmission request left using this processing.
3. FCNnCMCLCTL.FCNnCMCLSSTS can be periodically checked by a user application or can be checked after the transmit completion interrupt.

4. Do not execute any new transmission request including in the other message buffers while transmission abort processing is in progress.

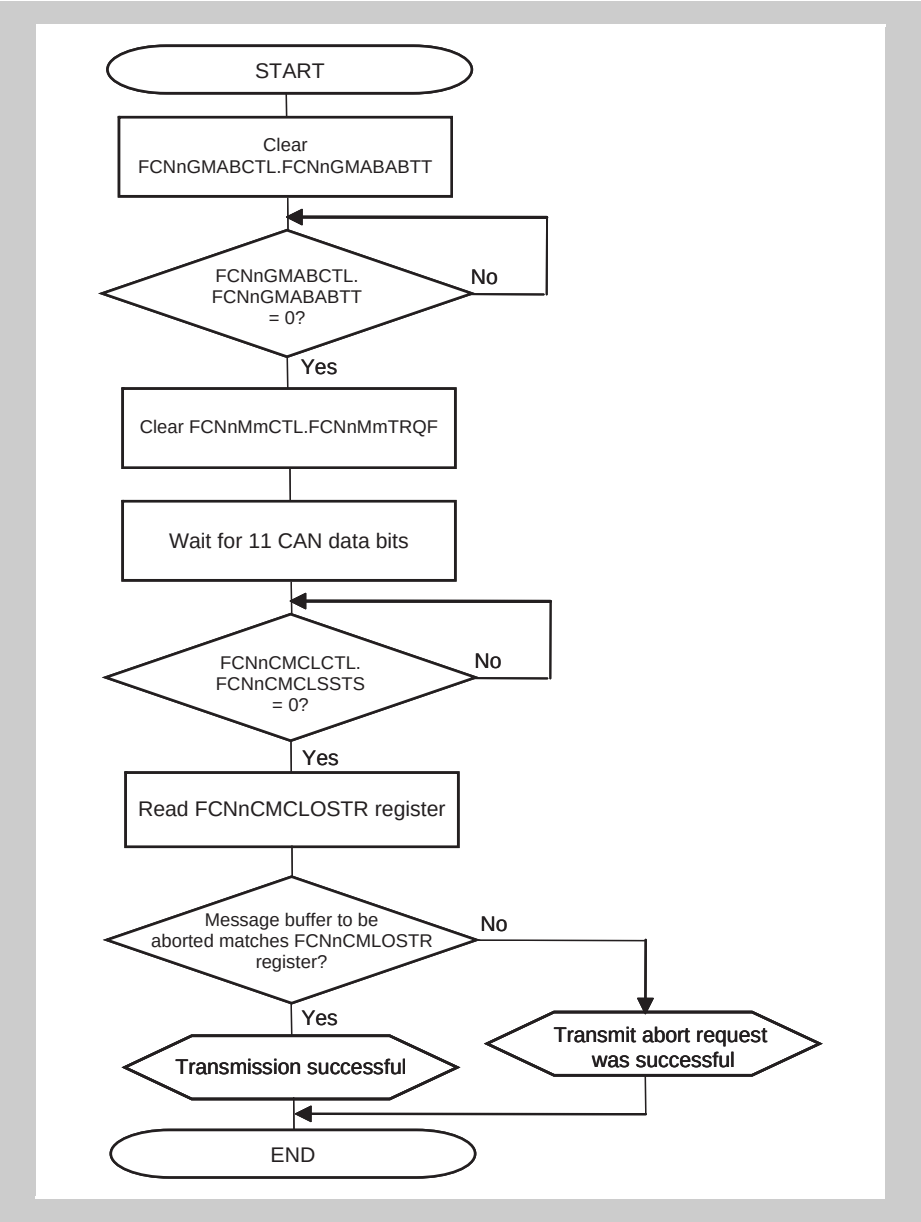



**Figure 18-28 Transmission abort processing (normal operation mode with ABT) - Repeat option for aborted message**

**Cautions**

1. Clear FCNnMmCTL.FCNnMmTRQF for aborting transmission request, not FCNnMmCTL.FCNnMmRDYF.
2. Before making a sleep mode transition request, confirm that there is no transmission request left using this processing.
3. FCNnCMCLCTL.FCNnCMCLSSTS can be periodically checked by a user application or can be checked after the transmit completion interrupt.
4. Do not execute any new transmission request including in the other message buffers while transmission abort processing is in progress.

*Figure 18-29 "Transmission abort processing (normal operation mode with ABT) - No repetition option for aborted message"* shows the processing to skip resumption of transmitting a message that was stopped when transmission of an ABT message buffer was aborted.

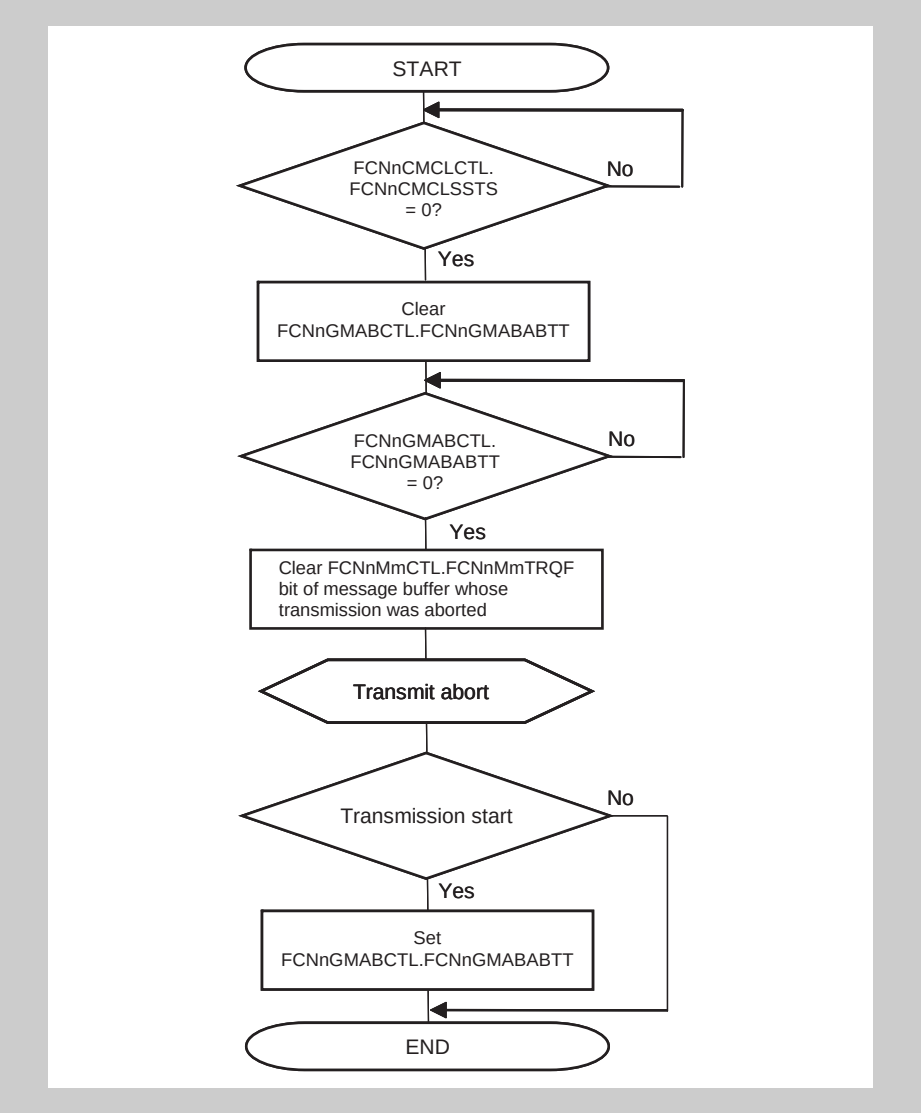



**Figure 18-29 Transmission abort processing (normal operation mode with ABT) - No repetition option for aborted message**

**Cautions**

1. Do not set any transmission requests while ABT transmission abort processing is in progress.
2. Make a CAN Controller sleep/stop mode transition request after FCNnGMABCTL.FCNnGMABABTT is cleared (after ABT mode is aborted) following the procedure shown in *Figure 18-29 "Transmission abort processing (normal operation mode with ABT) - No repetition option for aborted message"* or *Figure 18-30 "ABT transmission request abort processing (normal operation mode with ABT)"*. When clearing a transmission request in an area other than the ABT area, follow the procedure shown in *Figure 18-27 "Transmission abort processing (except normal operation mode with ABT)" on page 1011* .

*Figure 18-30 "ABT transmission request abort processing (normal operation mode with ABT)"* shows the processing to not skip resumption of transmitting a message that was stopped when transmission of an ABT message buffer was aborted.

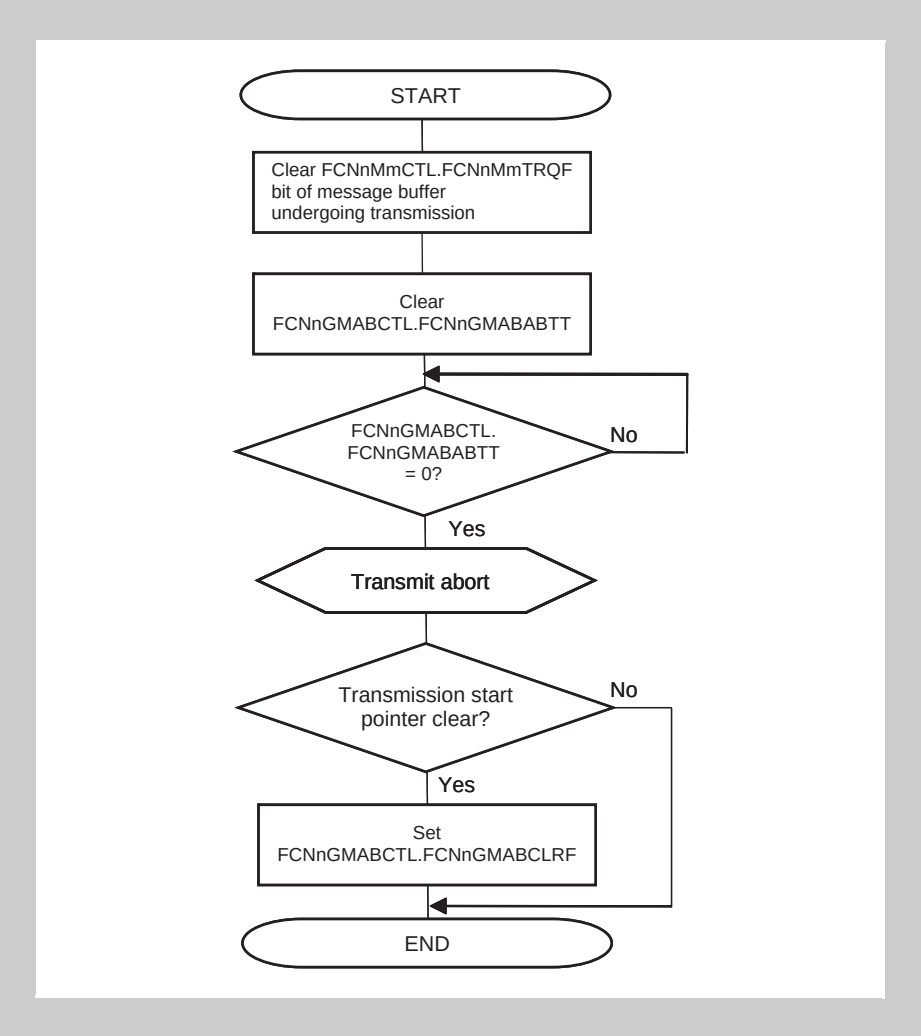



**Figure 18-30 ABT transmission request abort processing (normal operation mode with ABT)**

**Cautions**

1. Do not set any transmission requests while ABT transmission abort processing is in progress.
2. Make a CAN Controller sleep/stop mode request after FCNnGMABCTL.FCNnGMABABTT is cleared (after ABT mode is stopped) following the procedure shown in *Figure 18-29 "Transmission abort processing (normal operation mode with ABT) - No repetition option for aborted message"* or *Figure 18-30 "ABT transmission request abort processing (normal operation mode with ABT)"*. When clearing a transmission request in an area other than the ABT area, follow the procedure shown in *Figure 18-27 "Transmission abort processing (except normal operation mode with ABT)" on page 1011* .

*Figure 18-31 "ABT transmission request abort processing (normal operation mode with ABT) with transmit abort interrupt flag"* shows the processing on ABT mode, when using the Transmit Abort functionality (Transmit Abort Flag). The box "Transmission Abort Success" represents the checking of the transmission abort success by checking the FCNnMmTCPF flag within the ABT message buffers.

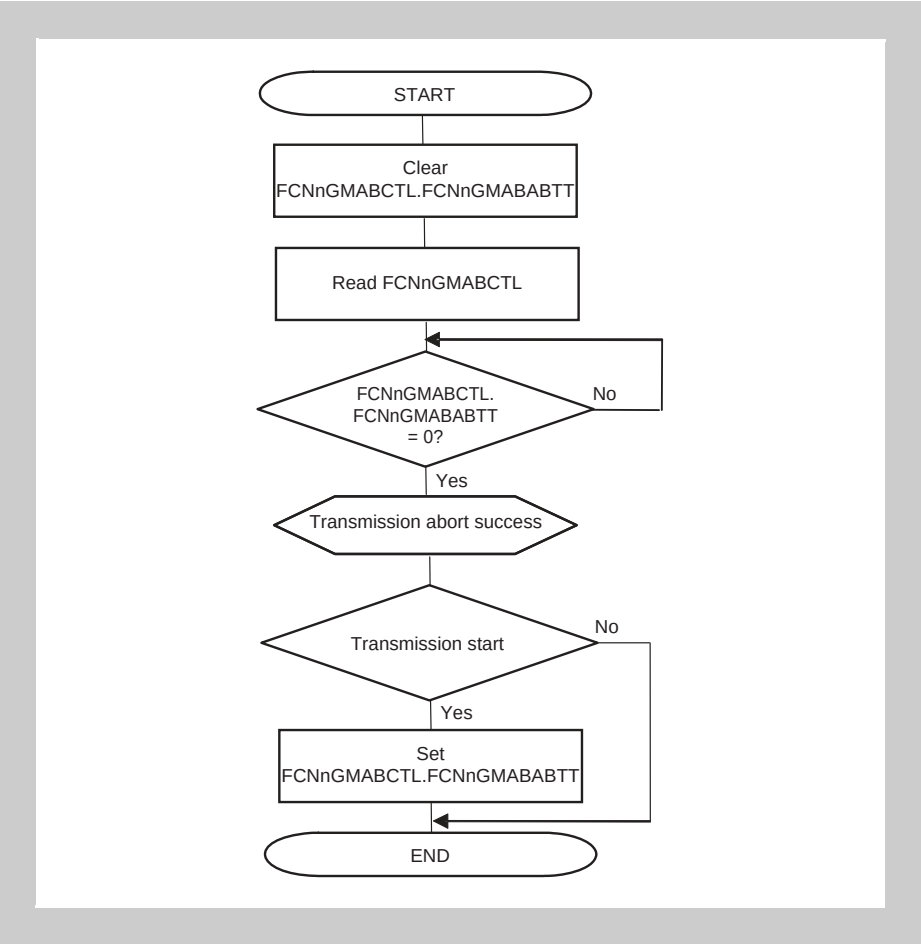



**Figure 18-31 ABT transmission request abort processing (normal operation mode with ABT) with transmit abort interrupt flag**

**Cautions**

1. Do not set any transmission requests while ABT transmission abort processing is in progress.
2. Issue a CAN sleep mode/CAN stop mode request only, after ABTTRG is cleared (and after ABT mode has stopped) following the procedure shown above. When clearing a transmission request in an area other than the ABT area, follow the procedures shown in flowcharts of aborting transmit requests in operation modes except ABT mode.

**Note** There is the case that all ABT is transmitted completely even if transmission abort interrupt is occurred. Then it is possible to know which message is finished transmission.

*Figure 18-32 "Transmission request abort processing (except normal operation mode with ABT) with transmit abort interrupt and transmission completely finished flag"* shows the processing when using the Transmit Abort functionality (Transmit Abort Interrupt).

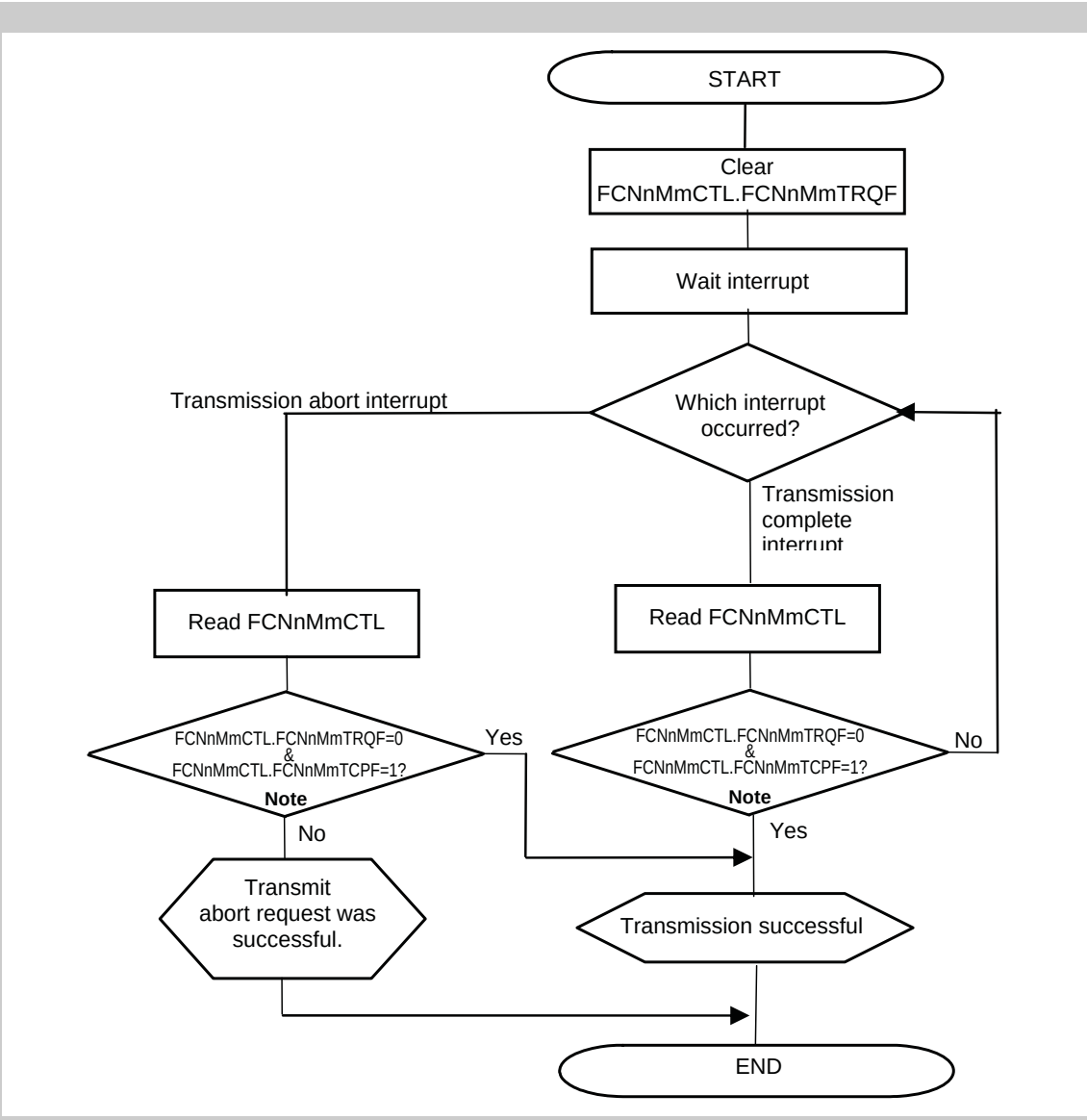



**Figure 18-32 Transmission request abort processing (except normal operation mode with ABT) with transmit abort interrupt and transmission completely finished flag**

**Note** Check FCNnMmCTL.FCNnMmMUCF and FCNnMmCTL.FCNnMmDTNF bits using one read access.

**Cautions**

1. Execute transmission request abort processing by clearing the TRQ bit, not the RDY bit.
2. Before making a sleep mode transition request, confirm that there is no transmission request left using this processing.
3. Do not execute another new transmission request in this or the other message buffers, while transmission abort processing is in progress.

4. It is prohibited to clear the transmit request flags of other messages, while transmission abort is in progress.
5. If a new transmission request is executed for a message buffer within 400 clocks of the CAN Controller, after transmission abort process and before IFS (Inter-Frame Space), that message might be transmitted in the next following transmission, although its ID priority was low.

**Note** Even if the TRQ bit is cleared, there is the possibility that the transmission still starts in the following 11 bits of interframe space (3bit) and suspend transmission (8bit), because the transmission request has already been received by protocol layer.

*Figure 18-33 "Transmission request abort processing (except normal operation mode with ABT) with transmission completely finished flag"* shows the processing when using the Transmit Abort functionality (Transmission Completely Finished Flag FCNnMmTCPF).

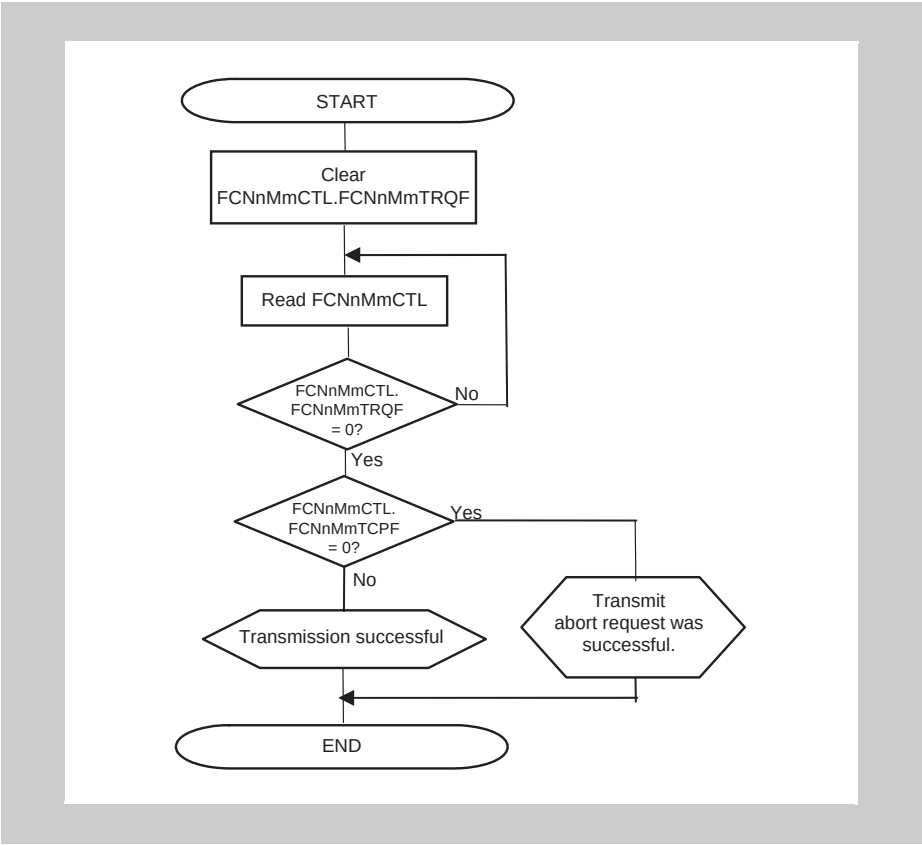



**Figure 18-33 Transmission request abort processing (except normal operation mode with ABT) with transmission completely finished flag**

**Cautions**

1. Execute transmission request abort processing by clearing the TRQ bit, not the RDY bit.
2. Before making a sleep mode transition request, confirm that there is no transmission request left using this processing.
3. Do not execute another new transmission request in this or the other message buffers, while transmission abort processing is in progress.
4. It is prohibited to clear the transmit request flags of other messages, while transmission abort is in progress.
5. If a new transmission request is executed for a message buffer within 400 $T_{CANCH}$ of the CAN Controller, after transmission abort process and before IFS (Inter-Frame Space), that message might be transmitted in the next following transmission, although its ID priority was low.

**Note** Even if the TRQ bit is cleared, there is the possibility that the transmission still starts in the following 11 bits of interframe space (3bit) and suspend transmission (8bit), because the transmission request has already been received by protocol layer.

### 18.16.3 Message reception

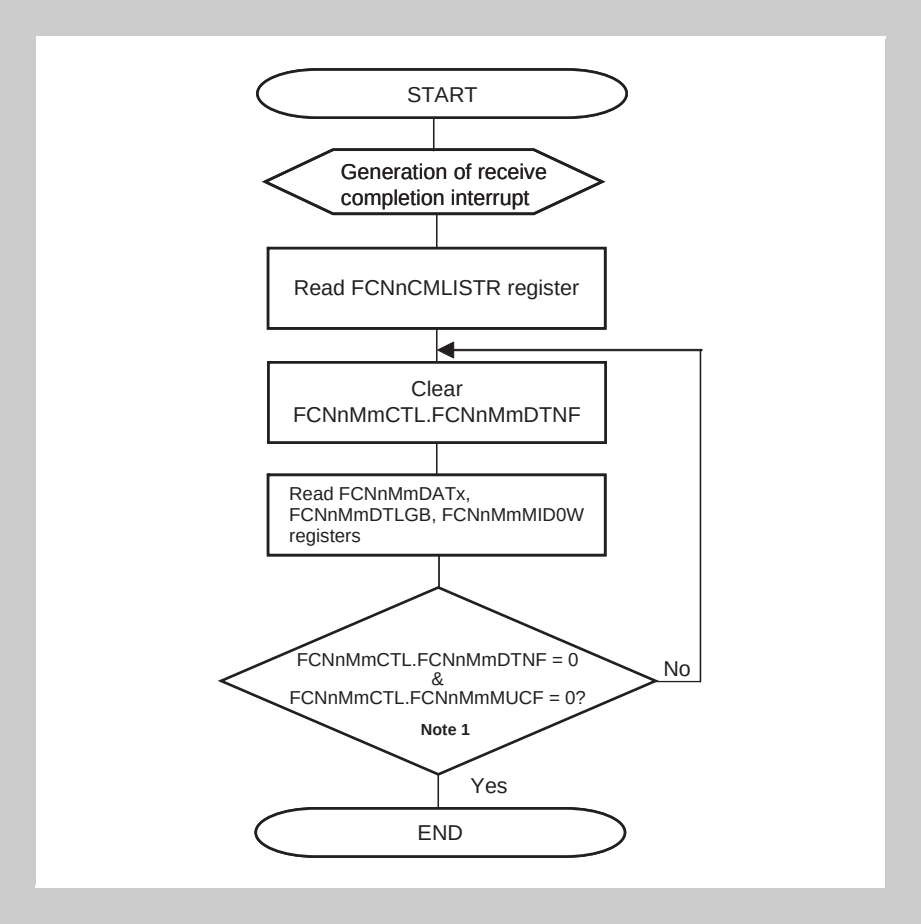



**Figure 18-34 Reception via interrupt (using FCNnCMLISTR register)**

**Notes**

1. Check FCNnMmCTL.FCNnMmMUCF and FCNnMmCTL.FCNnMmDTNF bits using one read access.
2. Also check the FCNnGMCLSSMO flag at the beginning and at the end of the interrupt routine, in order to check the access to the message buffers as well as reception history list registers, in case a pending sleep mode had been executed. If FCNnGMCLSSMO is detected to be cleared at any check, the actions and results of the processing have to be discarded and processed again, after FCNnGMCLSSMO is set again.
   It is recommended to cancel any sleep mode requests, before processing RX interrupts.

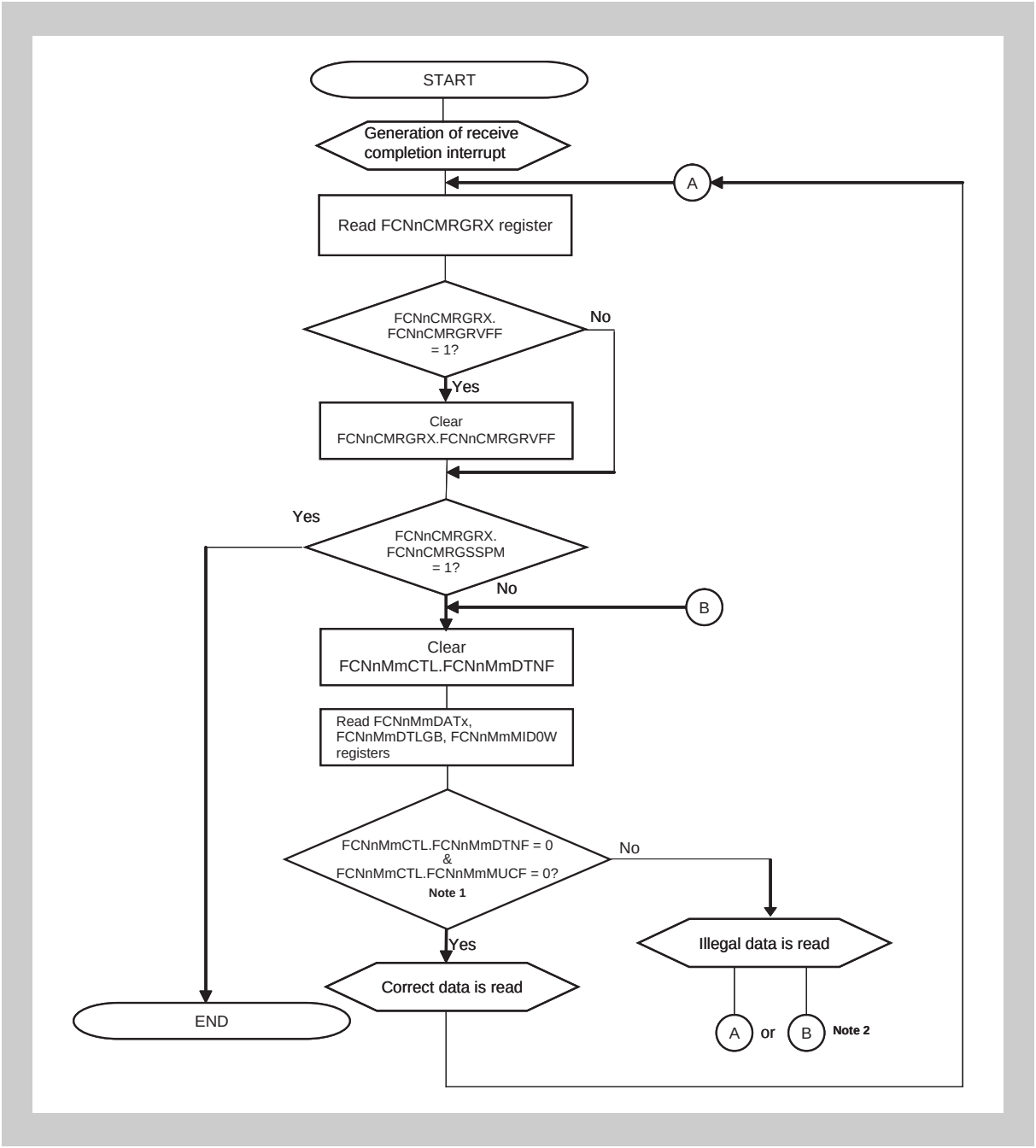



**Figure 18-35 Reception via interrupt (using FCNnCMRGRX register)**

**Notes**

1. Check FCNnMmCTL.FCNnMmMUCF and FCNnMmCTL.FCNnMmDTNF bits using one read access.
2. Depending of the processing target of the application, two ways are possible:
   – Way A: The message is not processed within this pass, but with the next pass, depending on the timing this can happen latest with the next Receive Interrupt.
   Other messages will be processed earlier.
   – Way B: The message is processed within this pass, the loop waits on this message.
   Other messages will be processed later.

3. Also check the FCNnGMCLSSMO flag at the beginning and at the end of the interrupt routine, in order to check the access to the message buffers as well as reception history list registers, in case a pending sleep mode had been executed. If FCNnGMCLSSMO is detected to be cleared at any check, the actions and results of the processing have to be discarded and processed again, after FCNnGMCLSSMO is set again.
   It is recommended to cancel any sleep mode requests, before processing RX interrupts.
4. If FCNnCMRGRX.FCNnCMRGRVFF was set once, the receive history list is inconsistent. Consider to scan all configured receive buffers for receptions.

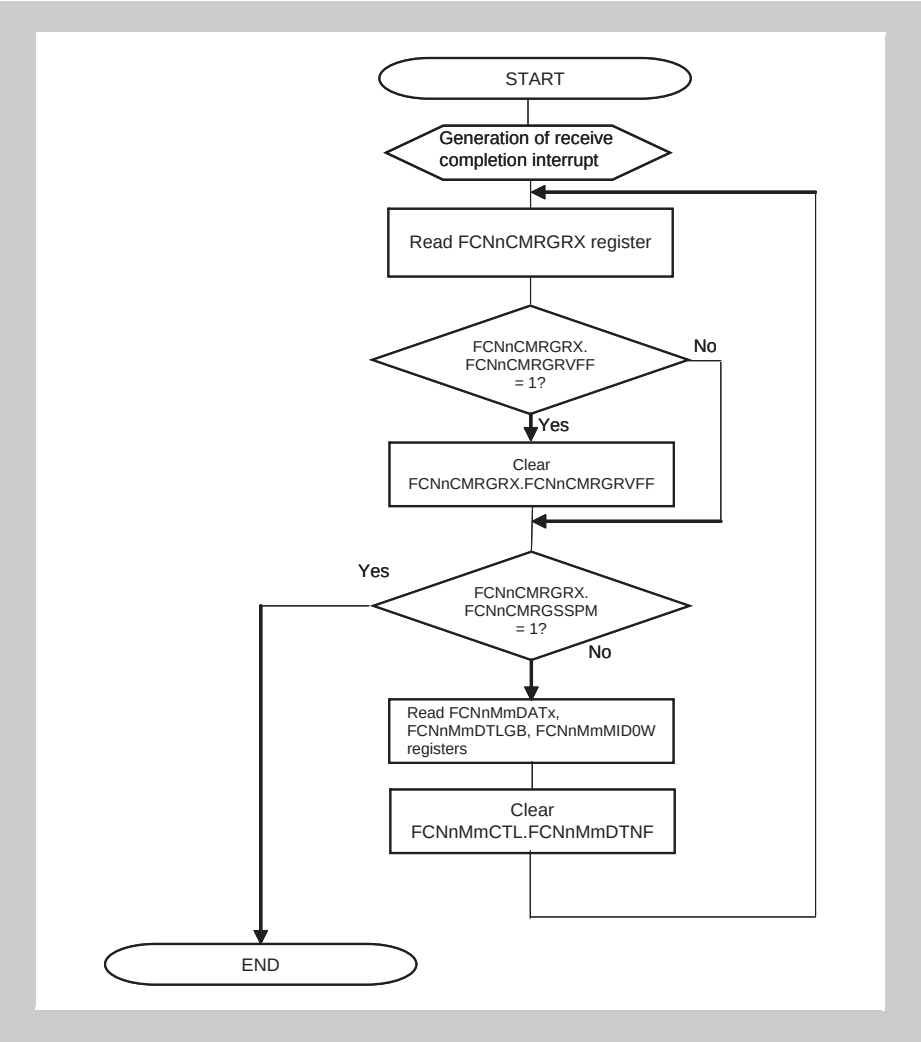



**Figure 18-36 Reception via interrupt (using FCNnCMRGRX register), alternative way**

**Notes**

1. Also check the FCNnGMCLSSMO flag at the beginning and at the end of the interrupt routine, in order to check the access to the message buffers as well as reception history list registers, in case a pending sleep mode had been executed. If FCNnGMCLSSMO is detected to be cleared at any check, the actions and results of the processing have to be discarded and processed again, after FCNnGMCLSSMO is set again.
   It is recommended to cancel any sleep mode requests, before processing RX interrupts.
2. If FCNnCMRGRX.FCNnCMRGRVFF was set once, the receive history list is inconsistent. Consider to scan all configured receive buffers for receptions.
3. This flow will not provide most recently received data for the application. However, due to less effort on processing, it reduces interrupt load.
4. The overwrite function (FCNnMmSTRB.FCNnMmSSOW=1) must not be used with this flow - data inconsistency could occur.
5. It can be used alternatively to *Figure 18-35 “Reception via interrupt (using FCNnCMRGRX register)” on page 1020* .

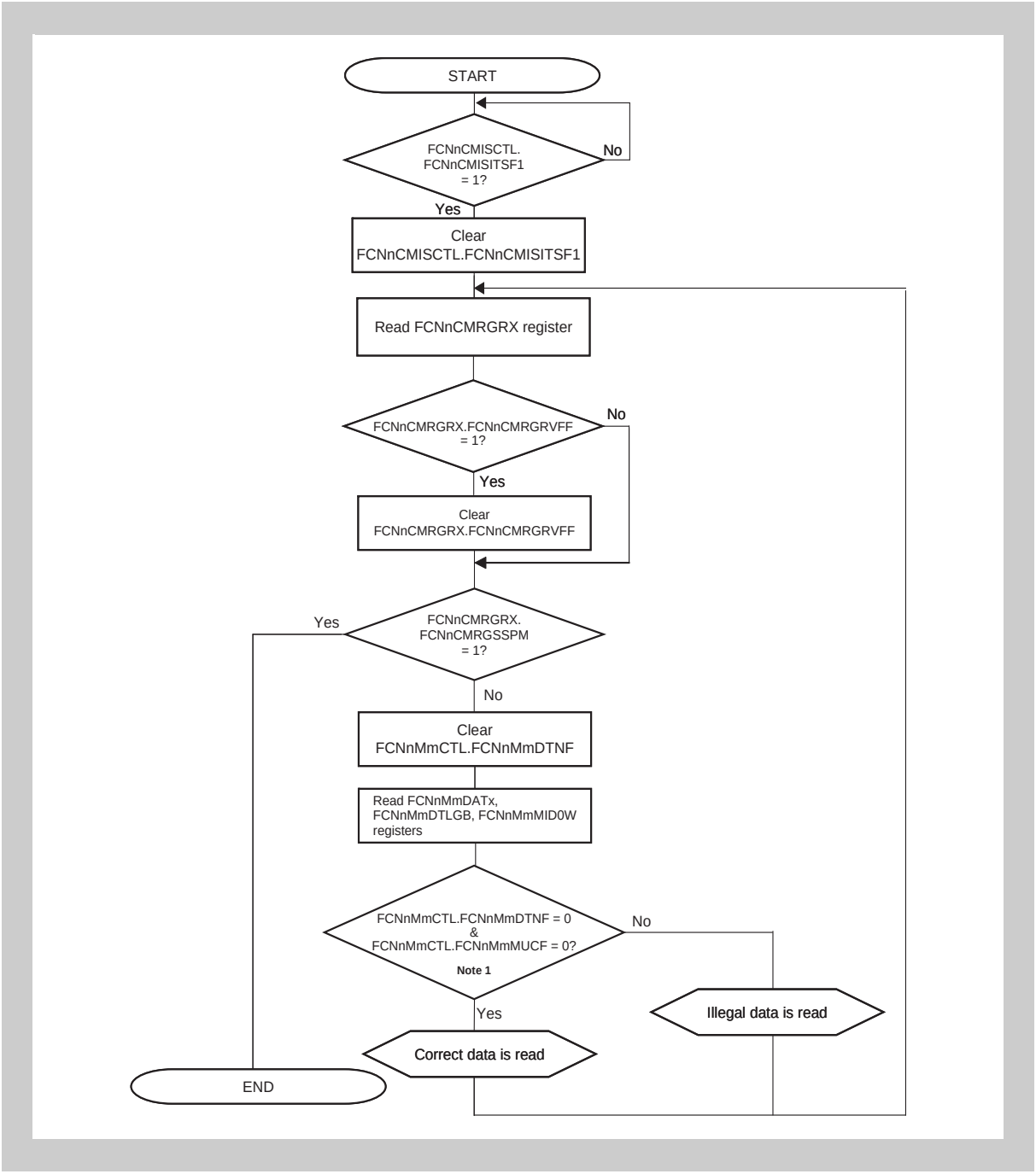



**Figure 18-37 Reception via software polling**

**Notes**

1. Check FCNnMmCTL.FCNnMmMUCF and FCNnMmCTL.FCNnMmDTNF bits using one read access.
2. Also check the FCNnGMCLSSMO flag at the beginning and at the end of the polling routine, in order to check the access to the message buffers as well as reception history list registers, in case a pending sleep mode had been executed. If FCNnGMCLSSMO is detected to be cleared at any check, the actions and results of the processing have to be discarded and processed again, after FCNnGMCLSSMO is set again.
3. If FCNnCMRGRX.FCNnCMRGRVFF was set once, the receive history list is inconsistent. Consider to scan all configured receive buffers for receptions.

### 18.16.4 Power save modes

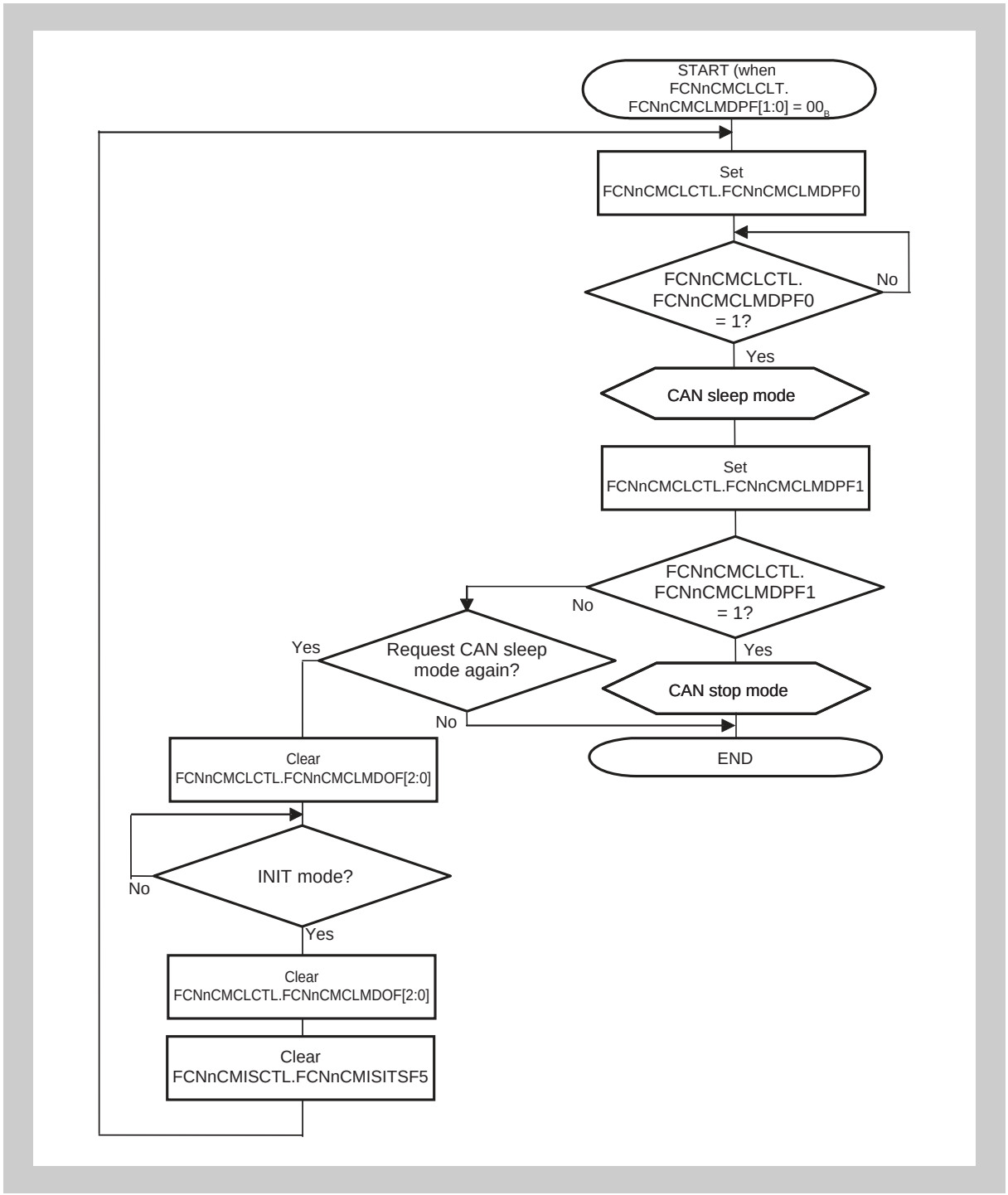



**Figure 18-38 Setting CAN Controller sleep mode/stop mode**

**Caution** To abort transmission before making a request for the CAN Controller sleep mode, perform processing according to previously given flowcharts.

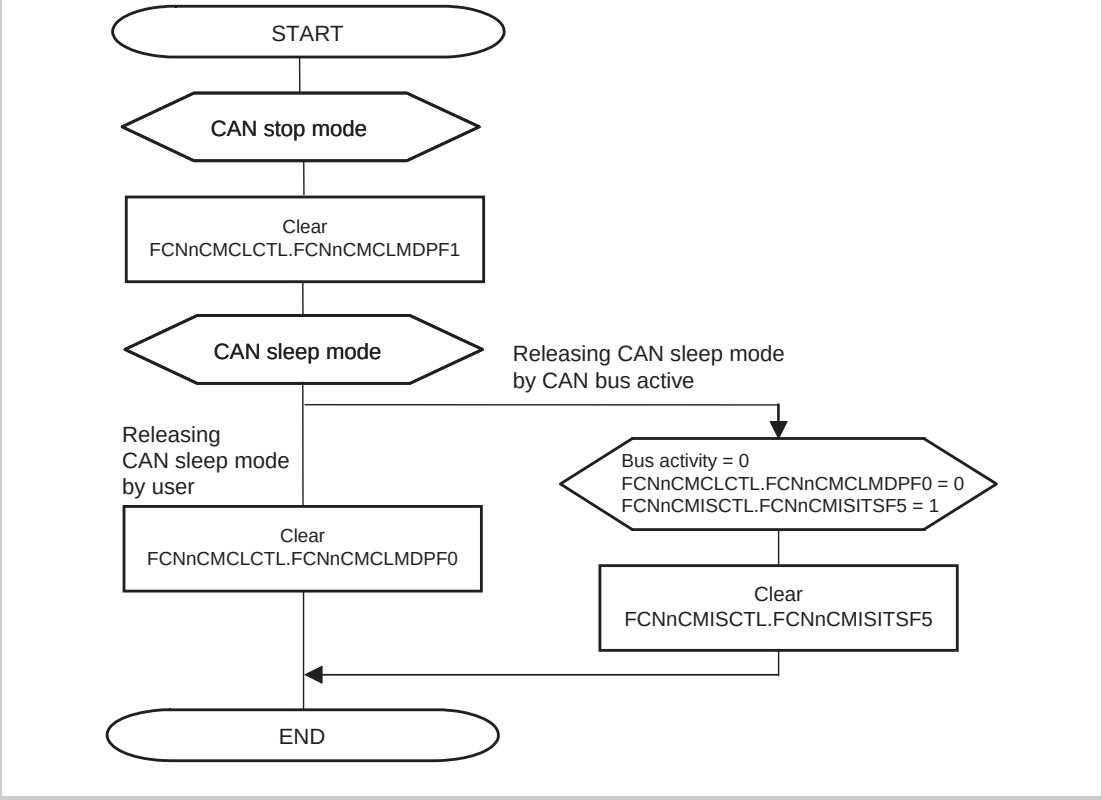



**Figure 18-39 Clear CAN Controller sleep/stop mode**

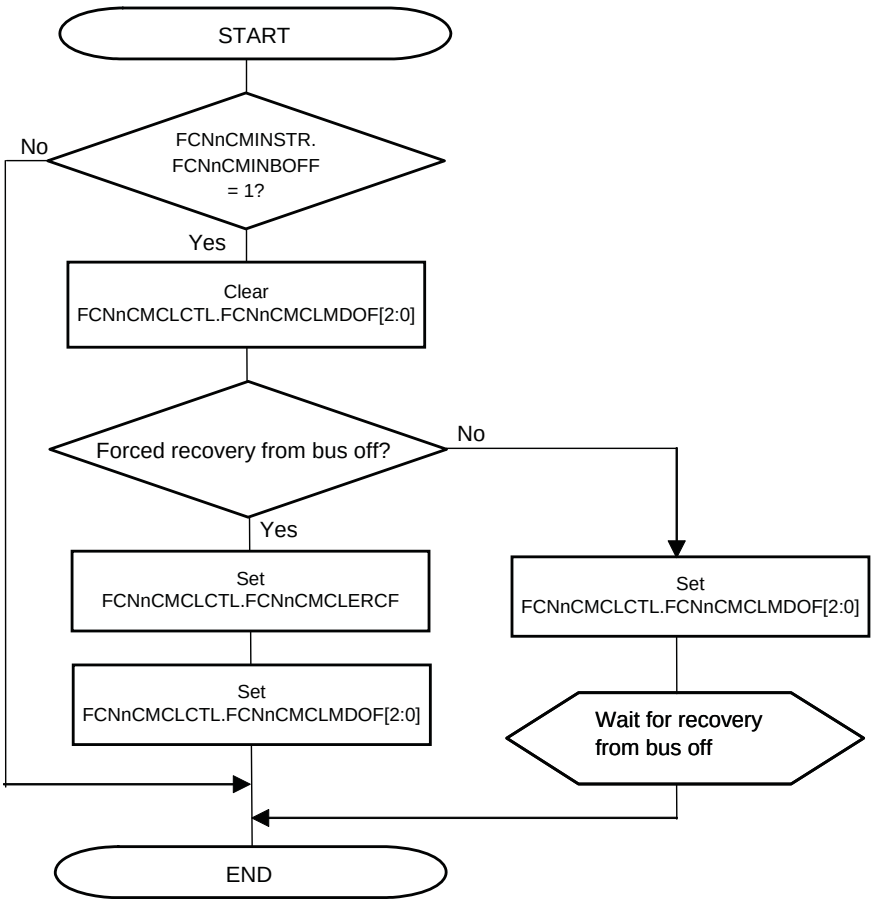



**Figure 18-40 Bus-off recovery (except normal operation mode with ABT)**

**Caution** When the transmission from the initialization mode to any operation modes is requested to execute bus-off recovery sequence again in the bus-off recovery sequence, reception error counter is cleared.
Therefore it is necessary to detect 11 consecutive recessive-level bits 128 times on the bus again.

**Note** Operation mode: Normal operation mode, normal operation mode with ABT, receive-only mode, singleshot mode, self-test mode.

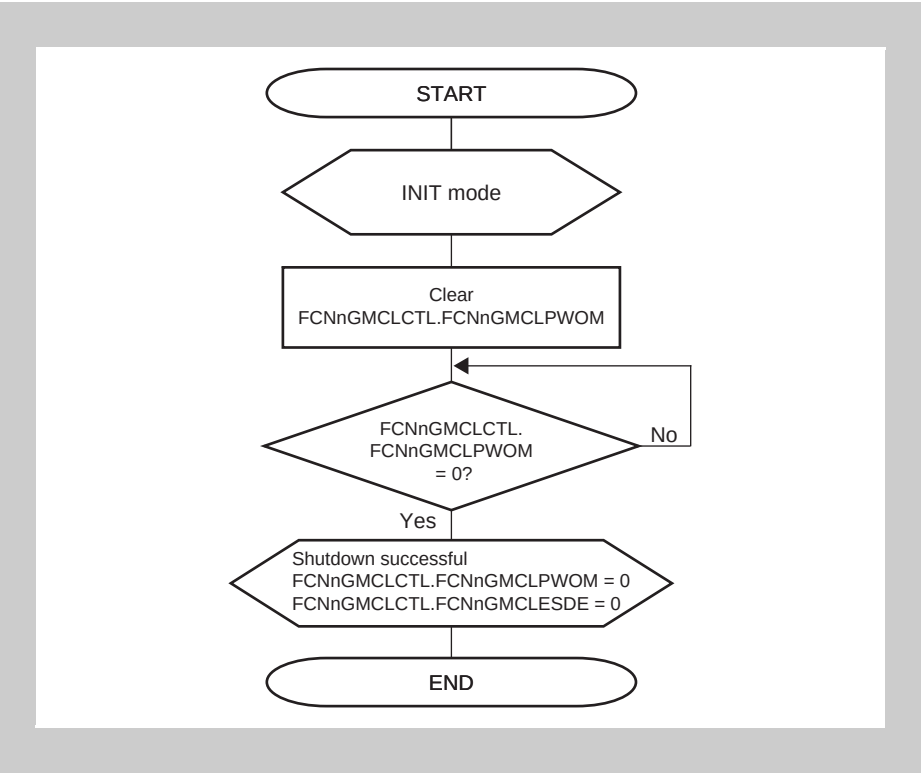



**Figure 18-41 Normal shutdown process**

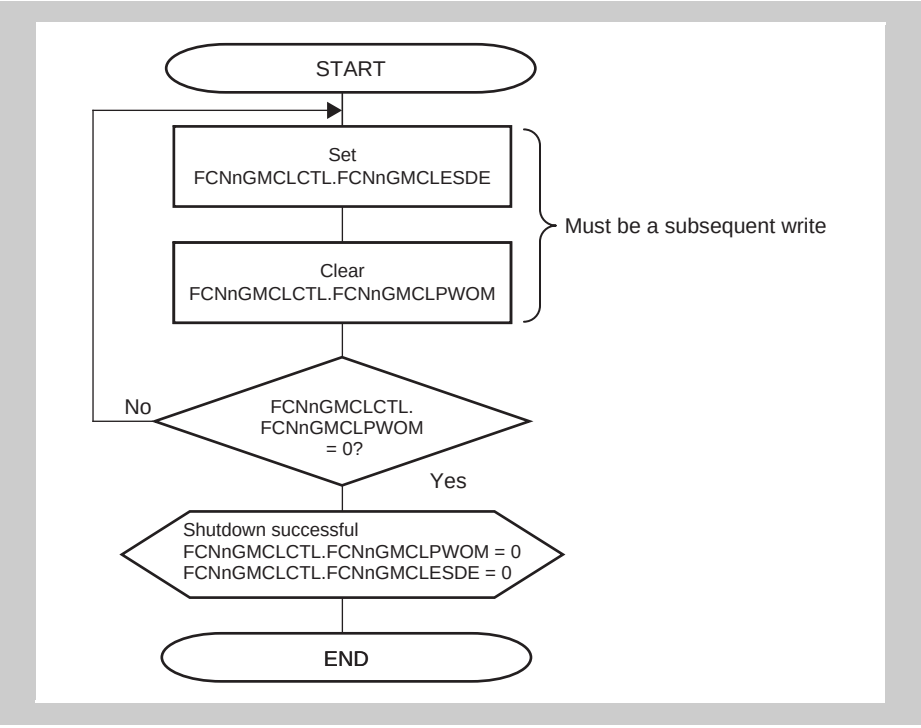



**Figure 18-42 Forced shutdown process**

**Caution** Do not read- or write-access any registers by software between setting the FCNnGMCLESDE bit and clearing the FCNnGMCLPWOM bit.

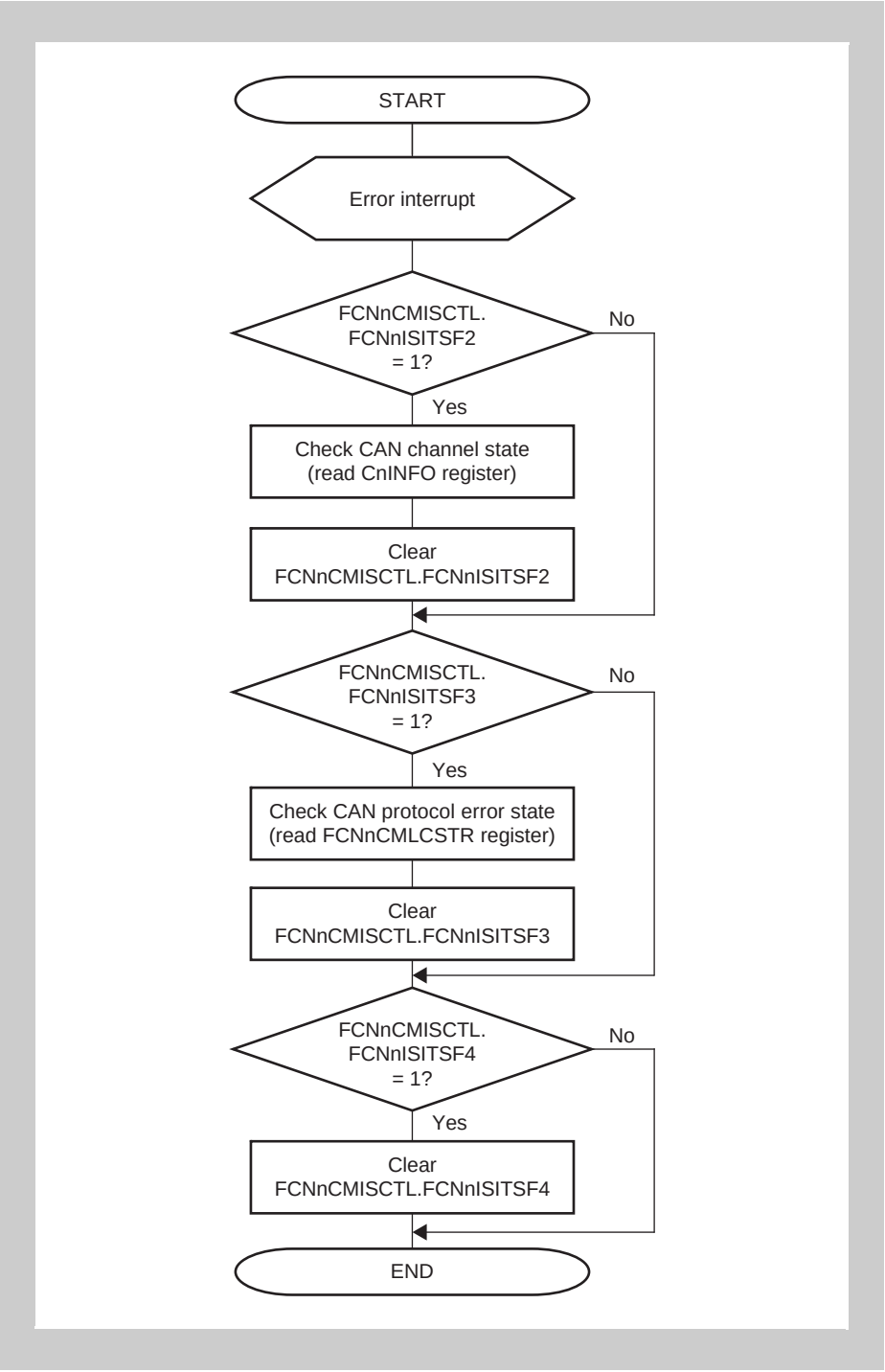



**Figure 18-43 Error handling**

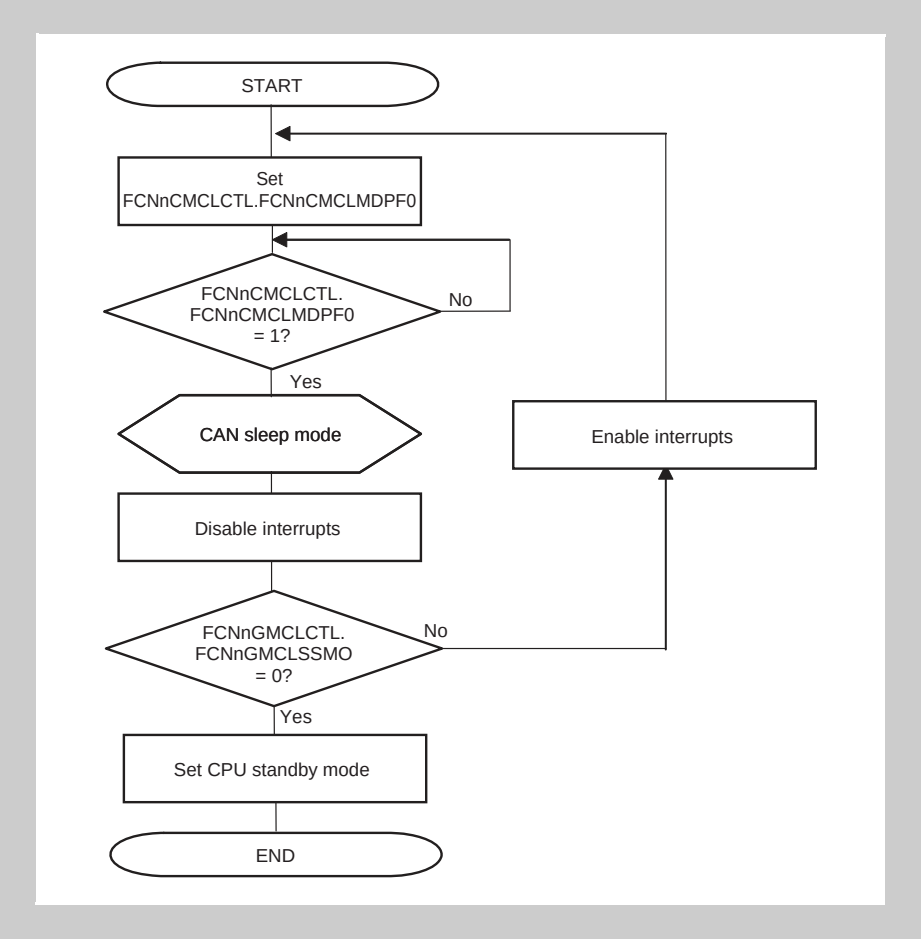



**Figure 18-44 Setting CPU stand-by (from CAN Controller sleep mode)**

**Notes**

1. Before the CPU is set in the CPU standby mode, please check if the CAN Controller sleep mode has been reached.
   However, after check of the sleep mode, until the CPU is set in the CPU standby mode, the sleep mode may be cancelled by wakeup from CAN bus.
2. There is a possibility, that between the check of FCNnGMCLSSMO = 0 and setting of the CPU standby mode a wake up condition on the CAN bus occurs. In that case the CAN Controller releases the SLEEP mode, the FCNnCMISITSF5 bit is set and if enabled the wake up interrupt will be generated.

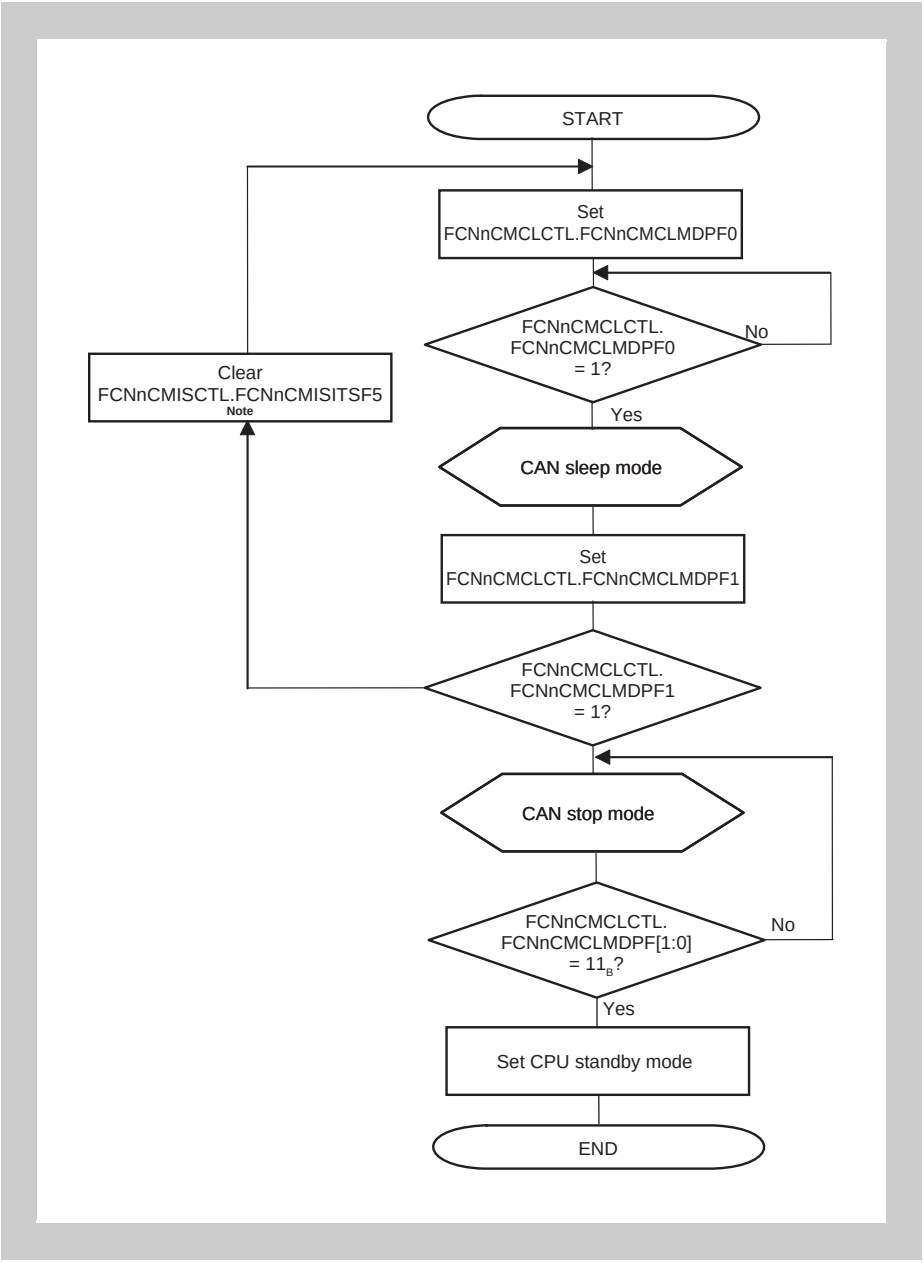



**Figure 18-45 Setting CPU stand-by (from CAN Controller stop mode)**

**Caution** The CAN Controller stop mode can only be released by setting $01_B$ to FCNnCMCLCTL.FCNnCMCLMDPF[1:0] and not by a change in the CAN Controller bus state.

# Chapter 19 Clocked Serial Interface G (CSIG)

This chapter contains a generic description of the Clocked Serial Interface G (CSIG).

The first section describes all V850E2/Fx4-G specific properties, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 19.1 V850E2/Fx4-G CSIG Features

**Instances**

This microcontroller has following number of instances of the Clocked Serial Interface G.

**Table 19-1 Instances of CSIG**

| Clocked Serial Interface G | V850E2/FF4-G | V850E2/FG4-G |
|---|---|---|
| Instance | 2 | 3 |
| Name | CSIG0, CSIG4 | CSIG0, CSIG4, CSIG7 |

**Instances index n**

Throughout this chapter, the individual instances of a Clocked Serial Interface G is identified by the index “n” (n = 0, 4, 7), for example, CSIGnCTL0 for the CSIGn control register 0.

**Register addresses**

All CSIGn register addresses are given as address offsets to the individual base address <CSIGn_base>.
The base address <CSIGn_base> of each CSIGn is listed in the following table:

**Table 19-2 Register base addresses <CSIGn_base>**

| CSIGn instance | <CSIGn_base> address |
|---|---|
| CSIG0 | FF70 0000$_H$ |
| CSIG4 | FF74 0000$_H$ |
| CSIG7 | FF69 0000$_H$ |

**Clock supply** All Clocked Serial Interface G provide one clock input:

**Table 19-3 CSIGn clock supply**

| CSIGn instance | CSIGn clock | Connected to |
|---|---|---|
| CSIG0 | PCLK | Clock Controller CKSCLK_028 |
| CSIG4 | PCLK | Clock Controller CKSCLK_011 |
| CSIG7 | PCLK | Clock Controller CKSCLK_029 |

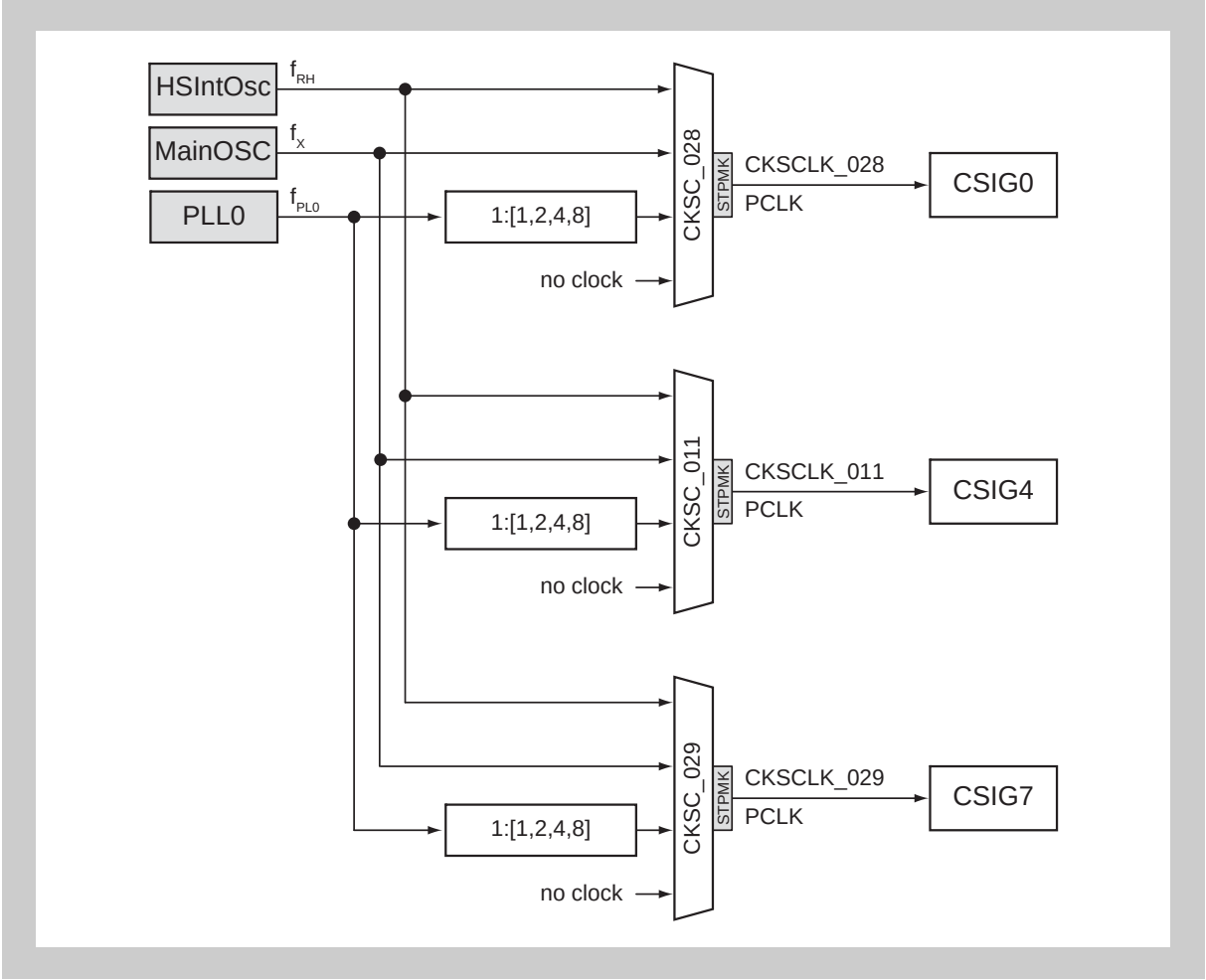



**Figure 19-1 CSIG clock supply**

### Interrupts and DMA

The Clocked Serial Interface G can generate the following interrupt and DMA requests:

**Table 19-4 CSIGn interrupt and DMA requests**

| CSIGn signals | Function | Connected to |
|---|---|---|
| **CSIG0:** | | |
| CSIGTIC | Communication status interrupt | Interrupt Controller INTCSIG0IC<br>DMA Controller trigger 35 |
| CSIGTIR | Reception status interrupt | Interrupt Controller INTCSIG0IR<br>DMA Controller trigger 34 |
| CSIGTIRE | Reception error interrupt | Interrupt Controller INTCSIG0IRE |
| **CSIG4:** | | |
| CSIGTIC | Communication status interrupt | Interrupt Controller INTCSIG4IC[a]<br>DMA Controller trigger 49 |
| CSIGTIR | Reception status interrupt | Interrupt Controller INTCSIG4IR[a]<br>DMA Controller trigger 48 |
| CSIGTIRE | Reception error interrupt | Interrupt Controller INTCSIG4IRE[a] |
| **CSIG7:** | | |
| CSIGTIC | Communication status interrupt | Interrupt Controller INTCSIG7IC<br>DMA Controller trigger 57 |
| CSIGTIR | Reception status interrupt | Interrupt Controller INTCSIG7IR<br>DMA Controller trigger 56 |
| CSIGTIRE | Reception error interrupt | Interrupt Controller INTCSIG7IRE |

a) These interrupts can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

### CSIG H/W reset

The Clocked Serial Interfaces G and their registers are initialized by the following reset signal:

**Table 19-5 CSIGn reset signal**

| CSIGn | Reset signal |
|---|---|
| CSIGn | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

### Internal signals

The internal signal connections of the Clocked Serial Interface G are listed in the following table.

**Table 19-6 CSIGn internal signal connections**

| CSIGn signal | Function | Connected to |
|---|---|---|
| **CSIG0:** | | |
| CSIGTSSO | CSIGTSO output buffer control | Port CSIG0SO output buffer control |
| **CSIG4:** | | |
| CSIGTSSO | CSIGTSO output buffer control | Port CSIG4SO output buffer control |
| **CSIG7:** | | |
| CSIGTSSO | CSIGTSO output buffer control | Port CSIG7SO output buffer control |

### Handshake function

The CSIGn handshake function uses the CSIGn module's handshake signal CSITSHSG, which is connected to the port signals CSIGnRY.
If the CSIGn module operates in master mode, CSIGnRY is an input and in slave mode an output signal.
The same CSIGnRY can not be used as alternative input and output port function. The following table summarizes the usage options of the different CSIGnRY signals.

**Table 19-7 CSIGn handshake signal ports**

| CSIGn I/O port signal | Port | Alternative function | CSIGnRY usage |
|---|---|---|---|
| **CSIG0:** | | | |
| CSIG0RY | P3_4 | ALT_IN4 ALT_OUT4 | in CSIG0 master or slave mode |
| | P4_9 | ALT_OUT2 | only in CSIG0 slave mode |
| **CSIG4:** | | | |
| CSIG4RY | P4_10 | ALT_IN2 | only in CSIG4 master mode |
| **CSIG7:** | | | |
| CSIG7RY | P1_5 | ALT_IN4 ALT_OUT4 | in CSIG7 master or slave mode |

### 19.1.1 Data consistency check

The following table shows the CSIGnSO ports and their capability to use them for data consistency checks. Refer to *"Error detection"* in the section *"Functional Description"* below for details about data consistency checks.

**Table 19-8 CSIGn data consistency check ports**

| CSIGn I/O port signal | Port | Alternative function | Data consistency check |
|---|---|---|---|
| **CSIG0:** | | | |
| CSIG0SO | P0_14 | ALT_OUT4 | possible |
| | P3_6 | ALT_OUT4 | possible |
| | P4_4 | ALT_OUT2 | not possible |
| **CSIG4:** | | | |
| CSIG4SO | P0_1 | ALT_OUT2 | possible |
| | P4_7 | ALT_OUT2 | not possible |
| **CSIG7:** | | | |
| CSIG7SO | P1_3 | ALT_OUT4 | possible |

**Port configuration**

Input/output control of the CSIGnSO port is done by the CSIGn module, thus set PIPCn.PIPCn_m = 1.
All other necessary configuration is done automatically, when the data consistency check is enabled.

## 19.2 Functional Overview

**Features summary**

- Three-wire serial synchronous data transfer
- Master mode and slave mode
- Slave select input signal ($\overline{\text{CSIGTSSI}}$)
- Built-in baud rate generator
- Adjustable baud rate; in slave mode it is determined by the input clock
- Maximum transmission speed:
  - in master mode: PCLK/4
  - in slave mode: PCLK/6
- Adjustable clock phase and data phase
- Data transfer with MSB or LSB first
- Transfer data length selectable from 7 to 16 bits in 1-bit units
- EDL (Extended Data Length) function for transferring data with more than 16 bits
- Three selectable transfer modes:
  - transmit-only mode
  - receive-only mode
  - transmit/receive mode
- Built-in handshake function
- Separate transmit and receive buffers (two 16-bit registers)
- Error detection (data consistency check, parity, overrun)
- Three different interrupt request signals (CSIGTIC, CSIGTIR, CSIGTIRE)
- Various conditions for interrupt generation
- LBM (Loop Back Mode) function for self test

The block diagram shows the main components of the CSIG.

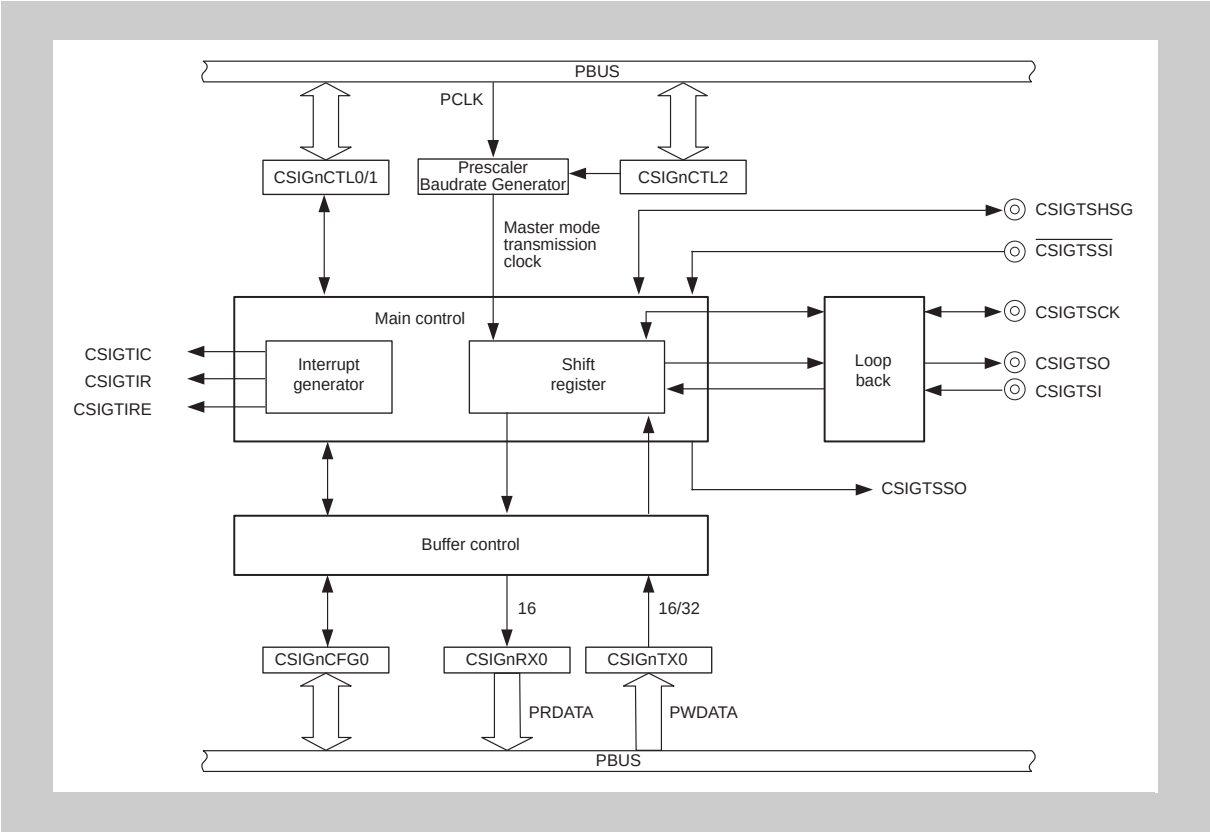



**Figure 19-2 CSIG block diagram**

In master mode, the transmission clock CSIGTSCK is generated by the built-in baud rate generator (BRG). In slave mode, the transmission clock is received from an external source.

## 19.3 Functional Description

The Clocked Serial Interface G uses three signals for communication:

- Transmission clock CSIGTSCK (output in master mode, in slave mode input)
- Data output signal CSIGTSO
- Data input signal CSIGTSI

If the CSIGn is operated in slave mode, the additional input signal $\overline{\text{CSIGTSSI}}$ acts as a chip select signal, that selects the CSIG to take part in data transfers.

Data transmission is bit-wise and serial and synchronous to the transmission clock.

The most important registers for setting up the CSIG are:

| Register | Function |
|---|---|
| CSIGnCTL0 | Enables/disables transmission clock, data transmission, and data reception |
| CSIGnCTL1 | Controls options like interrupt timing, extended data length, data consistency check, loop-back mode, handshake, etc. |
| CSIGnCTL2 | Selects master/slave mode and – effective in master mode – the baud rate of the internal generator |
| CSIGnCFG0 | Configures the communication protocol |

### 19.3.1 Master/slave mode

The master or slave mode primarily determines the source of the transmission clock.

#### (1) Master mode

In master mode, the serial communication clock is generated by the internal baud rate generator (BRG) and provided by signal CSIGTSCK.

Master mode is enabled by setting CSIGnCTL2.CSIGnPRS[2:0] to anything but $111_B$. In master mode, the frequency setting of the BRG becomes effective.

The initial level of CSIGTSCK depends on the clock phase selection bit: it is high when CSIGnCTL1.CSIGnCKR = 0, and is low when CSIGnCTL1.CSIGnCKR = 1.

The example below shows the communication in master mode for 8 data bits, CSIGnCTL1.CSIGnCKR = 0, CSIGnCFG0.CSIGnDAP = 0, and MSB first:

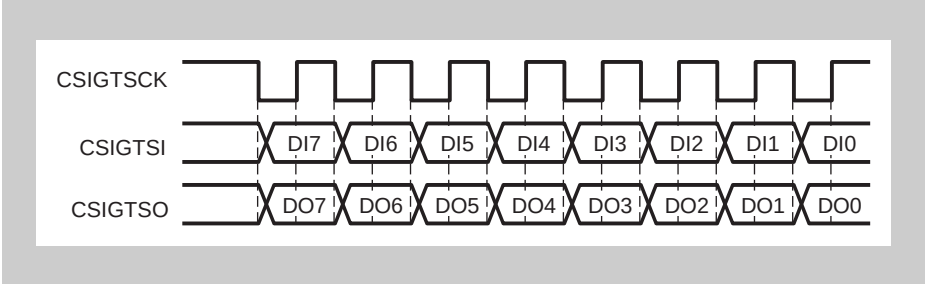



**Figure 19-3 Transmit/receive in master mode**

**(2) Slave mode**

In slave mode, another device is the communication master. The external clock is received by signal CSIGTSCK. Send/receive operation starts as soon as a clock signal is detected.

Slave mode is selected by setting CSIGnCTL2.CSIGnPRS[2:0] to $111_B$.

**Note** When using slave mode, disable the baud rate generator (BRG) by clearing bits CSIGnCTL2.CSIGnBRS[11:0]. When the BRG is disabled, CSIGTSCK stays on the level specified by CSIGnCTL1.CSIGnCKR.

The example below shows the communication in slave mode for 8 data bits, CSIGnCTL1.CSIGnCKR = 0, CSIGnCFG0.CSIGnDAP = 0, and MSB first:

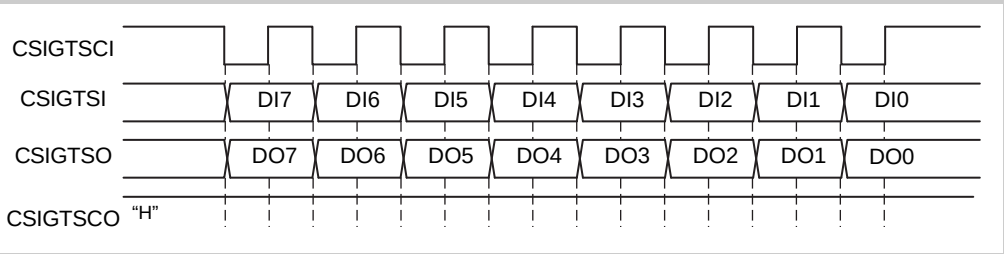



**Figure 19-4 Transmit/receive in slave mode**

## 19.3.2 Master/slave connections

**(1) One master and one slave**

The following figure illustrates the connections between one master and one slave.

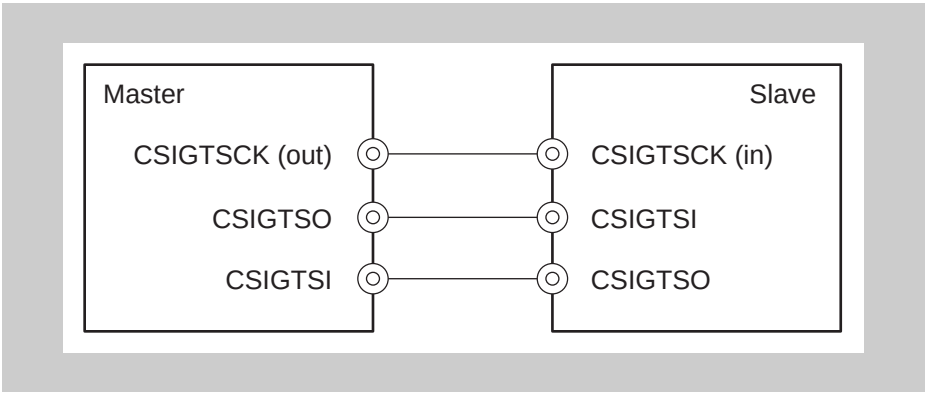



**Figure 19-5 Simple master/slave connection**

**(2) One master and multiple slaves**

The following figure illustrates the connections between one master and multiple slaves. In this case, the master must provide one slave select (SS) signal to each of the slaves. This signal is connected to the slave select input $\overline{\text{CSIGTSSI}}$ of the slave.

The recognition of the $\overline{\text{CSIGTSSI}}$ signal can be enabled/disabled by bit CSIGnCTL1.CSIGnSSE.

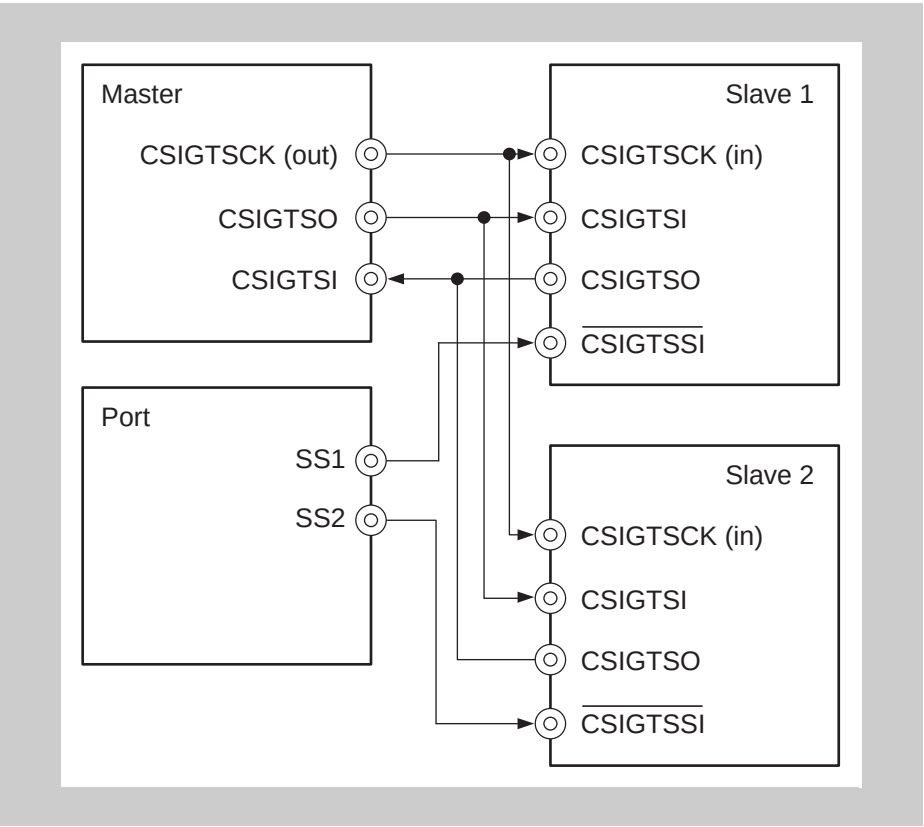



**Figure 19-6 Master to multiple slaves connection**

A slave is selected (enabled) when its $\overline{\text{CSIGTSSI}}$ signal is low.

If it is not selected, the slave will neither receive nor transmit data. In addition, the CSIGTSO output buffer is disabled order to avoid interference with the output of another slave which was selected.

**CSIGTSO buffer control**

The CSIG provides the signal CSIGTSSO to control the port output buffer of the serial output signal CSIGTSO. By use of this signal the CSIGTSO output buffer can be disabled under following conditions:

- The CSIG is enabled (CSIGnCTL0.CSIGnPWR = 1)
- The CSIG is operated in transmit-only or transmit/receive mode (CSIGnCTL0.CSIGnTXE = 1)
- The CSIG is operated with slave select enabled (CSIGnCTL1.CSIGnSSE = 1).
- The slave mode selection signal $\overline{\text{CSIGTSSI}}$ is inactive, i.e. on high level.

By this signal congestions on the external CSIGTSO signal line are avoided.

### 19.3.3 Transmission clock selection

In master mode, the transmission baud rate is selectable using the CSIGnPRS[2:0] and CSIGnBRS[11:0] bits in the CSIGnCTL2 register. The baud rate generator (BRG) counts up at every rising edge of PCLK.

The following figure shows a block diagram of the BRG.

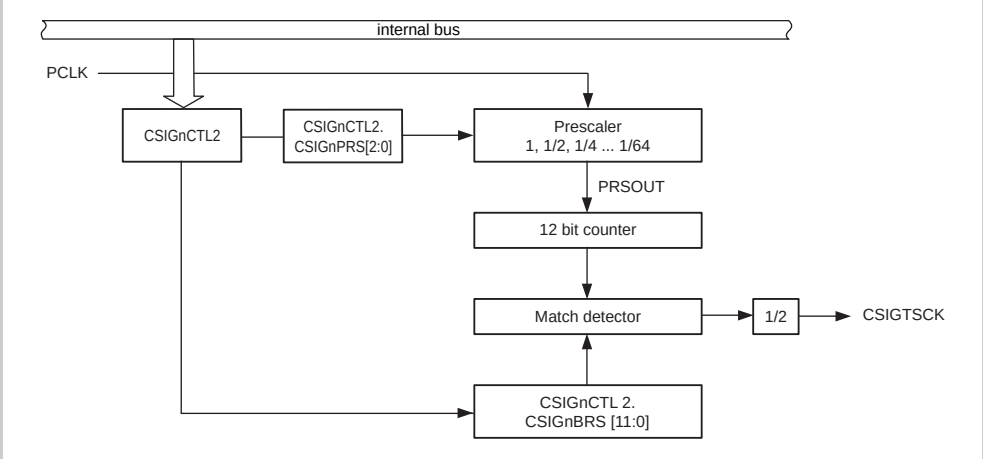



**Figure 19-7 BRG block diagram**

Clearing CSIGnCTL2.CSIGnBRS[11:0] disables the BRG.

**Baud rate calculation**

The baud rate is calculated as: PCLK / ($2^m$ × k × 2), where

m = CSIGnCTL2.CSIGnPRS[2:0] = 0 to 6

k = CSIGnCTL2.CSIGnBRS[11:0] = 1 to 4095

**Baud rate limits**

When setting the baud rate, please note:

- Maximum acceptable baud rate in master mode is PCLK / 4.
- Maximum acceptable baud rate in slave mode is PCLK / 6 and must be ensured by the external master.
- Minimum baud rate in both modes is PCLK / 524160.

### 19.3.4 Data transfer modes

#### (1) Transmit-only mode

Setting CSIGnCTL0.CSIGnTXE = 1 and CSIGnCTL0.CSIGnRXE = 0 puts the CSIG in transmit-only mode. Transmission starts when transmit data is written in the CSIGnTX0W or CSIGnTX0H register.

**Caution** In case transmit-only mode has been entered after any reception mode, the data in the CSIGnRX0 buffer becomes undefined after completion of the first transmission.
Consequently the reception register CSIDnRX0 has to be read before changing to transmit-only mode.

#### (2) Receive-only mode

Setting CSIGnCTL0.CSIGnTXE = 0 and CSIGnCTL0.CSIGnRXE = 1 puts the CSIG in receive-only mode.

In master mode, reception starts when dummy data is written to the CSIGnTX0W or CSIGnTX0H register.
All following receptions are triggered by a read from the receive data register CSIGnRX0, as long as CSIGnBCTL0.CSIGnSCE = 1.

In slave mode, reception starts when the communication clock CSIGTSCK from the master is received. In this case, it is not necessary to write data to the CSIGnTX0W or CSIGnTX0H register of the slave.

**Note** In receive-only mode, any previously received data must be read from the reception register CSIGnRX0 in order to avoid any overwrite situation.

Moreover the communication start bit CSIGnBCTL0.CSIGnSCE has to be set to 1 and has to set back to 0 before reading the last received data from CSIGnRX0.

The recommended procedure is:

1. Set CSIGnBCTL0.CSIGnSCE = 1.
2. Before starting the first receive operation, read CSIGnRX0 (dummy data).
3. Wait for the reception interrupt CSIGTIR.
4. Read CSIGnRX0 (received data).
   In case of further data receptions continue at 4. until all data has been received.
   Before reading the last received data from CSIGnRX0, set CSIGnBCTL0.CSIGnSCE = 0.

#### (3) Transmit / receive mode

Setting CSIGnCTL0.CSIGnTXE = 1 and CSIGnCTL0.CSIGnRXE = 1 puts the CSIG in transmit/receive mode.

Data transfer (transmission and reception) starts when transmit data is written to the CSIGnTX0W or CSIGnTX0H register.

### 19.3.5 Data length selection

#### (1) Data length selection without extended length

Transmission data length is selectable from 7 to 16 bits using the CSIGnDLS[3:0] bits in the CSIGnCFG0 register. The examples below show the communication with MSB first (CSIGnCFG0.CSIGnDIR = 0):

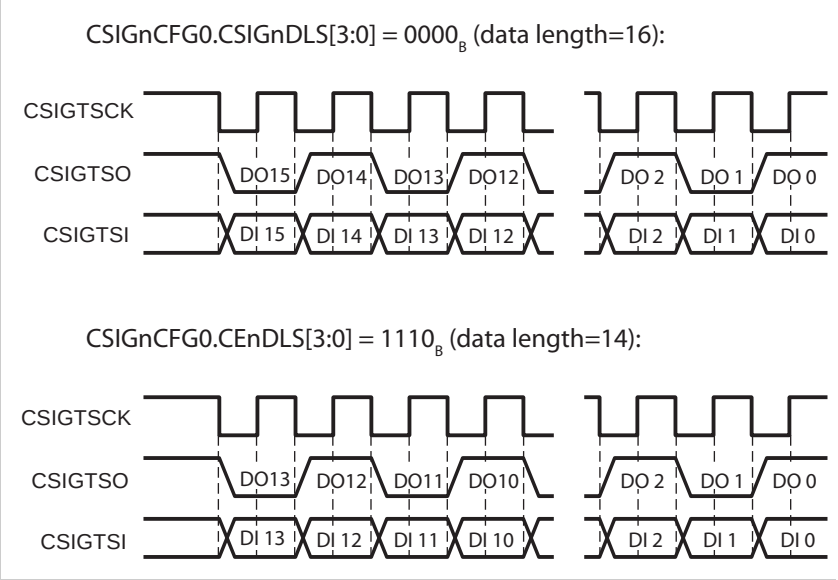



**Figure 19-8 Data length select function**

#### (2) Data length selection with extended data length

If the data to be sent/received exceeds 16 bits, the extended data length (EDL) feature can be used.

The EDL function is enabled by setting bit CSIGnCTL1.CSIGnEDLE to 1.

The EDL function works as follows:

- The data has to be broken into 16-bit blocks plus remainder. For example, a string of 42 bits would be broken into two 16-bit blocks plus 10 bits.
- The remainder defines the “data length” that has to be specified in CSIGnCFG0.CSIGnDLS[3:0].
- CSIGnTX0W.CSIGNEDL must be set to 1. In this case, the data writen to CSIGnTX0W is sent as a 16-bit data length regardless of the CSIGnCFG0.CSIGnDLS[3:0] bits.
- The transfer is complete after the data with the specified data length (the remainder with CSIGnTX0W.CSIGNEDL = 0) has been sent.

**Example** Example for sending 40-bit data, for example the string $123456789A_H$:

40 bits are split into 2 × 16 bits plus 8 bits.

- Initialize CSIGnCFG0.CSIGnDLS[3:0] = 8.
- To send the string $123456789A_H$ with MSB first, write the following sequence to CSIGnTX0W:
  - $2000\ 1234_H$ (CSIGnTX0W.CSIGnEDL = 1)
  - $2000\ 5678_H$ (CSIGnTX0W.CSIGnEDL = 1)
  - $0000\ 009A_H$ (CSIGnTX0W.CSIGnEDL = 0)

The following figure illustrates the timing.

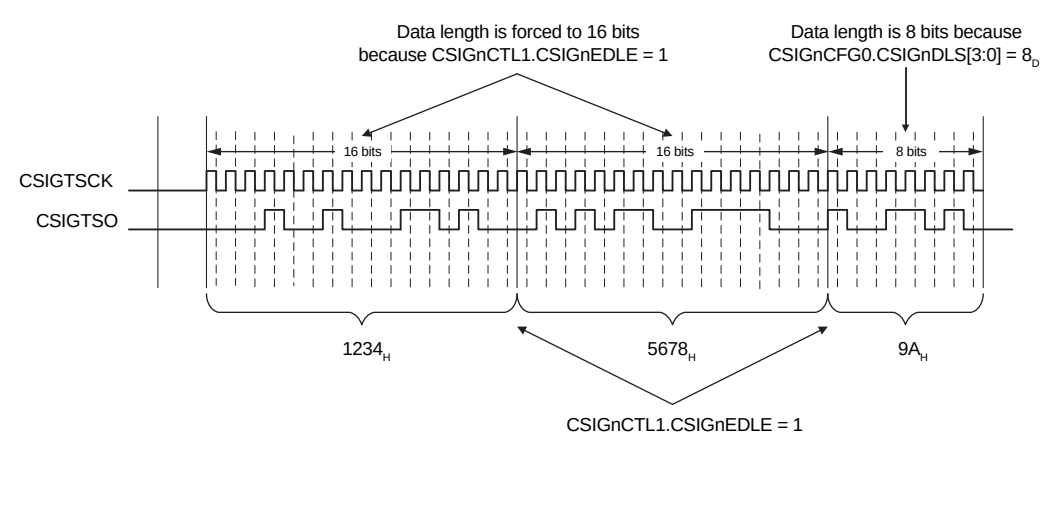



**Figure 19-9 EDL timing diagram**

**Notes**

1. It is not possible to send two consecutive data with a data length of less than 7 bits.
2. If parity is enabled, the parity bit is added after the last bit.
3. To consider the data direction, pay attention to the following example: (example data to be sent $123456_H$)
   - CSIGnCFG0.CSIGnDIR = 1: LSB first
     – CSIGnTX0 = $2000\ 3456_H$ (EDL bit = 1)
     – CSIGnTX0 = $0000\ 0012_H$ (EDL bit = 0)
   - CSIGnCFG0.CSIGnDIR = 0: MSB first
     – CSIGnTX0 = $2000\ 1234_H$ (EDL bit = 1)
     – CSIGnTX0 = $0000\ 0056_H$ (EDL bit = 0)

### 19.3.6 Serial data direction select function

The serial data direction is selectable using the CSIGnDIR bit in the CSIGnCFG0 register. The examples below show the communication for 8-bit data (CSIGnCFG0.CSIGnDLS[3:0] = $1000_B$):

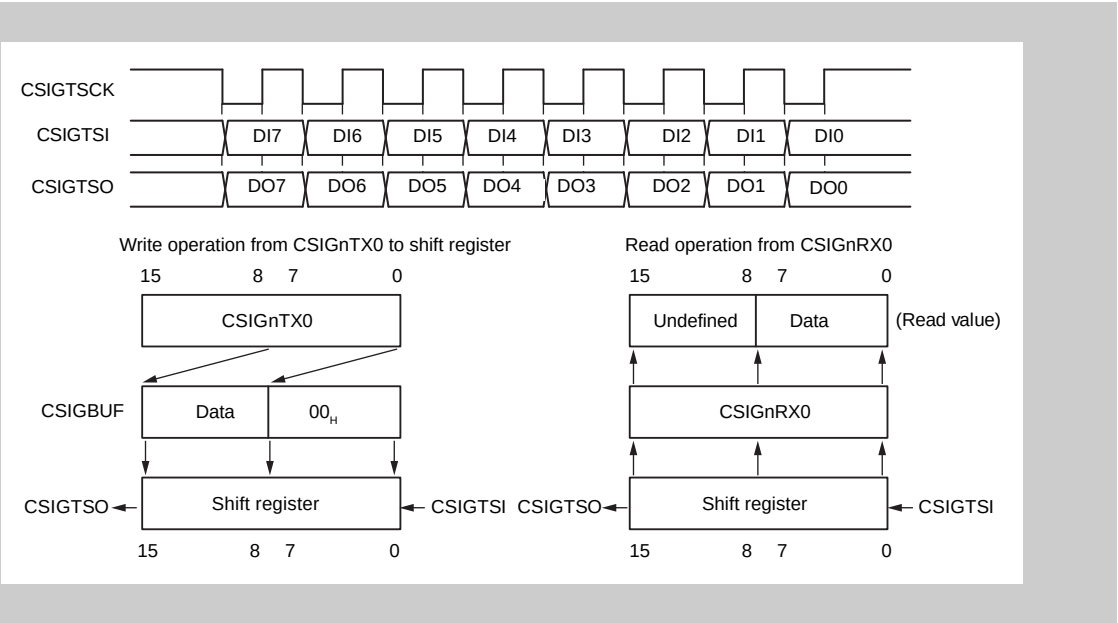



**Figure 19-10 Serial data direction select function - MSB first (CSIGnDIR = 0)**

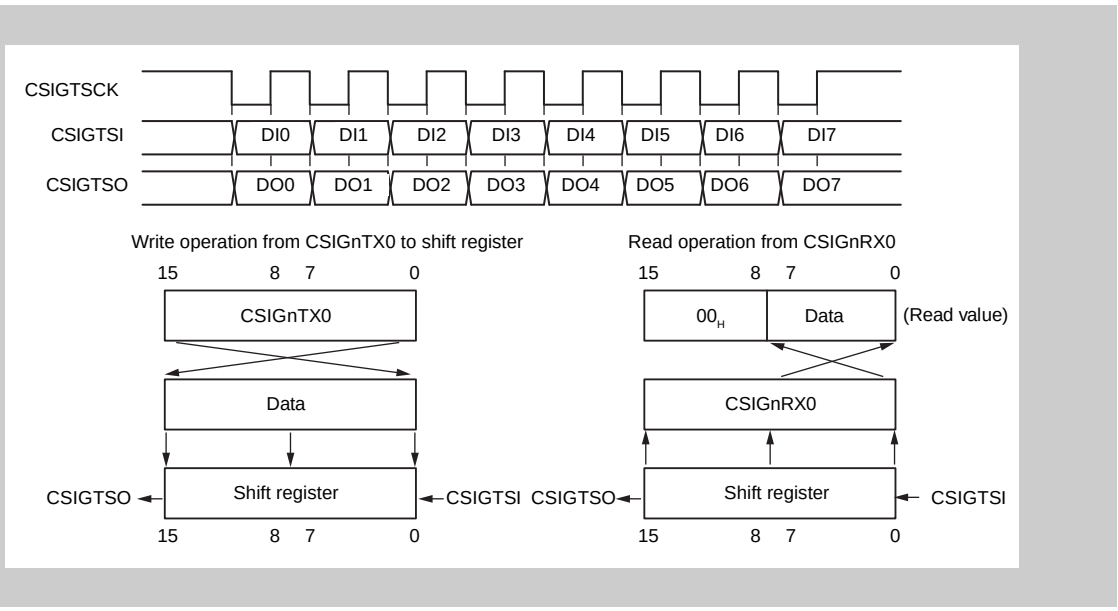



**Figure 19-11 Serial data direction select function - LSB first (CSIGnDIR = 1)**

### 19.3.7 Communication in slave mode

The following figure illustrates the communication signals and timings in slave mode.

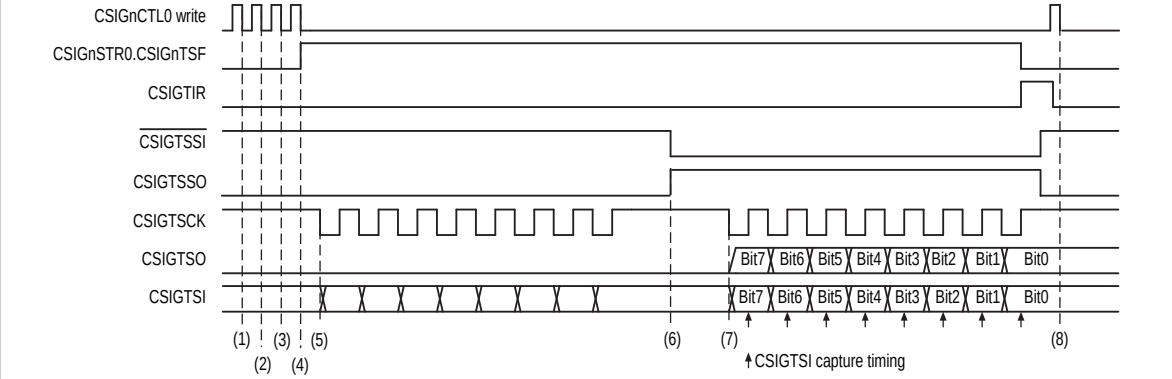



**Figure 19-12 Rx/Tx communication timing in slave mode**

1. CSIG is put into slave mode by setting CSIGnCTL2.CSIGnPRS[2:0] = $111_B$. CSIGnCTL1.CSIGnCKR and CSIGnCFG0.CSIGnDAP are 0.
2. Data length is 8 bits (CSIGnCFG0.CSIGnDLS[3:0] = $1000_B$). Data direction is MSB first (CSIGnCFG0.CSIGnDIR = 0).
3. CSIG is set to transmit/receive operation (CSIGnCTL0.CSIGnTXE = 1, CSIGnCTL0.CSIGnRXE = 1). Communication start is enabled.
4. The “transmission in progress” flag CSIGnSTR0.CSIGnTSF is automatically set when transfer data is written to the transmission register CSIGnTX0H.
5. As long as signal $\overline{\text{CSIGTSSI}}$ is high, transmission/reception is not started, even if an external transmission clock is applied to clock input signal CSIGTSCI. Input at CSIGTSI is ignored.
6. As soon as $\overline{\text{CSIGTSSI}}$ falls to low level, signal CSIGTSSO goes high, indicating that CSIGTSO is enabled and ready for transmission.
7. Now, as soon as the external clock signal appears at CSIGTSCK, the slave transmits data to CSIGTSO and simultaneously captures data from CSIGTSI.
8. Interrupt CSIGTIR indicates when the reception is complete. The register CSIGnRX0 can be read.

### 19.3.8 CSIG interrupts

CSIG can generate the following interrupts:

- CSIGTIC
- CSIGTIR
- CSIGTIRE

**Interrupt delay**

In master mode, all interrupts generated by the master can be delayed by one half period of the transmission clock CSIGTSCK. This is not possible in slave mode.

The delay is specified by setting bit CSIGnCTL1.CSIGnSIT = 1.

The following example illustrates the interrupt delay function, assuming a setting of CSIGnCTL1.CSIGnSIT = 1 (interrupt delay enabled), CSIGnCTL1.CSIGnCKR = 0, CSIGnCFG0.CSIGnDAP = 0 (normal clock and data phase), and CSIGnCFG0.CSIGnDLS[3:0] = $1000_B$ (data length 8 bits).

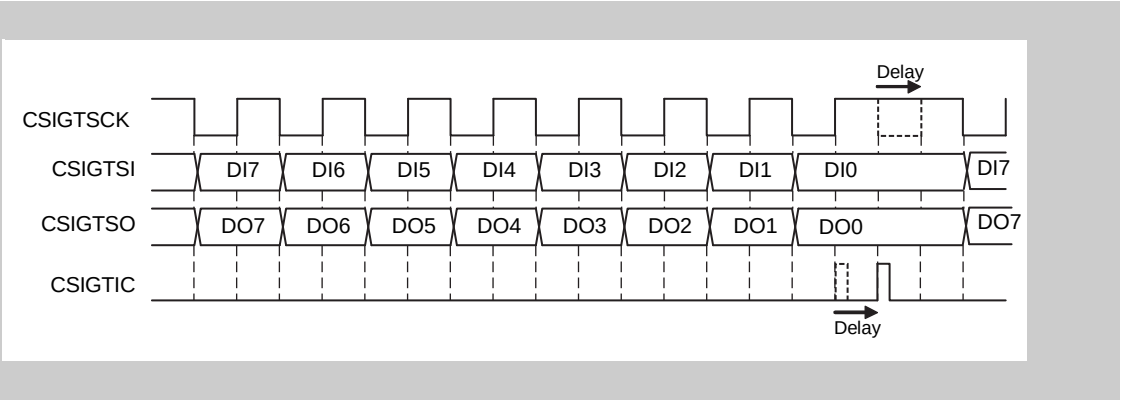



**Figure 19-13 Interrupt delay function (CSIGnCTL1.CSIGnSIT = 1)**

#### (1) CSIGTIC communication interrupt

This interrupt is normally generated after every data transfer. It can be used to trigger a DMA for writing new transmission data to register CSIGnTX0W or CSIGnTX0H.

The following example assumes master mode and a setting of CSIGnCTL1.CSIGnSIT = 0 (no interrupt delay), CSIGnCTL1.CSIGnCKR = 0, CSIGnCFG0.CSIGnDAP = 0 (normal clock and data phase), CSIGnCFG0.CSIGnDLS[3:0] = $1000_B$ (data length 8 bits), and CSIGnCTL1.CSIGnSLIT = 0 (normal interrupt timing).

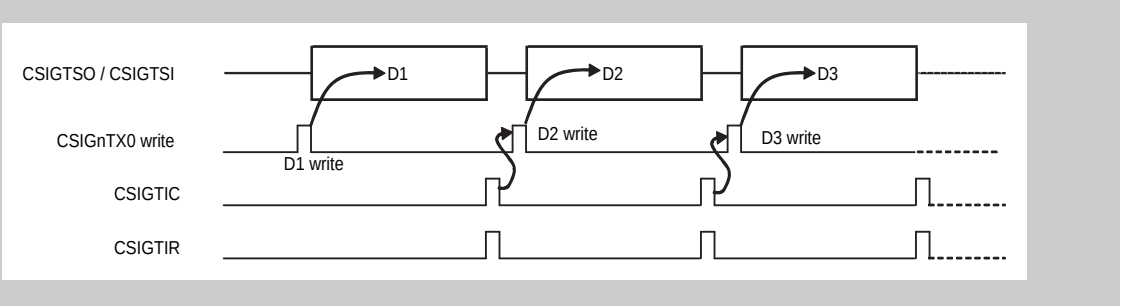



**Figure 19-14 Generation of CSIGTIC after communication (CSIGnCTL1.CSIGnSLIT = 0)**

However, CSIGTIC can also be set up to occur when the CSIGnTX0 register is free for the next data. This is specified by setting CSIGnCTL1.CSIGnSLIT = 1.

This mode allows more efficient data transfers.

The effect is illustrated in the figure below.

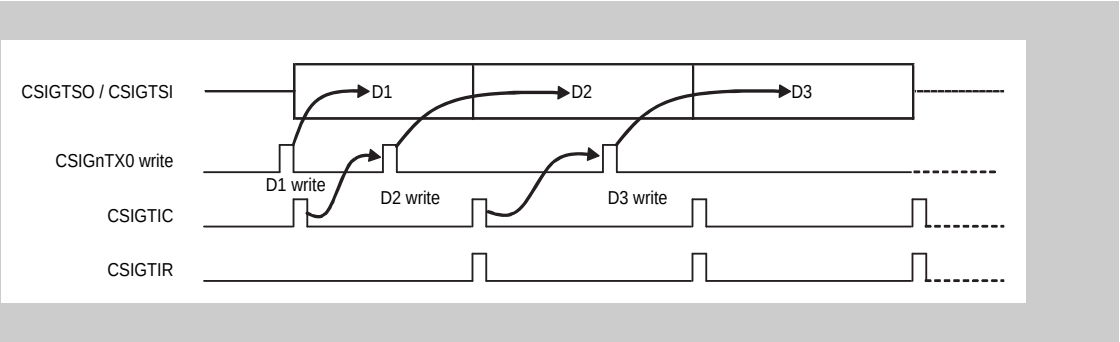



**Figure 19-15 Generation of CSIGTIC at the beginning of communication**

### (2) CSIGTIR reception interrupt

This interrupt is generated in receive-only and transmit/receive mode after data has been received and is available in the reception register. It can be used to trigger a DMA for reading the received data from register CSIGnRX0.

The following example assumes master mode and a setting of CSIGnCTL1.CSIGnSIT = 0 (no interrupt delay), CSIGnCTL1.CSIGnCKR = 0, CSIGnCFG0.CSIGnDAP = 0 (normal clock and data phase), and CSIGnCFG0.CSIGnDLS[3:0] = $1000_B$ (data length 8 bits).

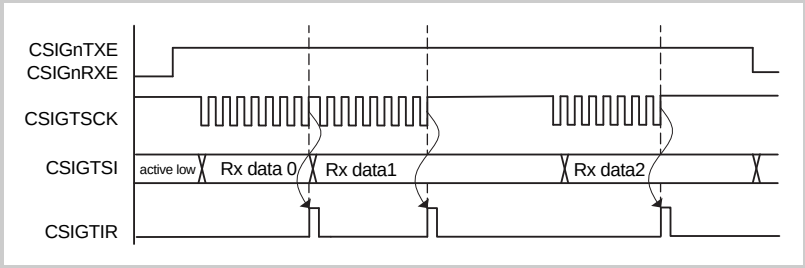



**Figure 19-16 Generation of CSIGTIR**

**(3) CSIGTIRE reception error interrupt**

This interrupt is generated whenever an error is detected.

**Table 19-9 Data error types**

| Error type | Communication status after error interrupt |
|---|---|
| Parity error | Interrupt is generated and communication continues |
| Data consistency error | Interrupt is generated and communication continues |
| Overrun error | Interrupt is generated and communication is stopped |

The type of error that caused the generation of CSIGTIRE is indicated in register CSIGnSTR0.

For details about the various error types refer to *19.3.11 “Error detection” on page 1053* .

## 19.3.9 Handshake function

CSIG features a handshake function to synchronize the master and the slave devices. This function can be enabled/disabled by bit CSIGnCTL1.CSIGnHSE. For handshake, the signal CSIGTSHSG is used.

The timing depends on the data phase selection bit CSIGnCFG0.CSIGnDAP.

**(1) Slave mode**

When CSIGnCTL1.CSIGnHSE = 1, the slave outputs CSITSHSG = 0 when it is busy. This happens when previous receive data is still in the CSIGnRX0 register, and new data cannot be copied from the shift register to CSIGnRX0 (CSIGnRX0 full condition).

The following examples assume a data length of 8 bits.

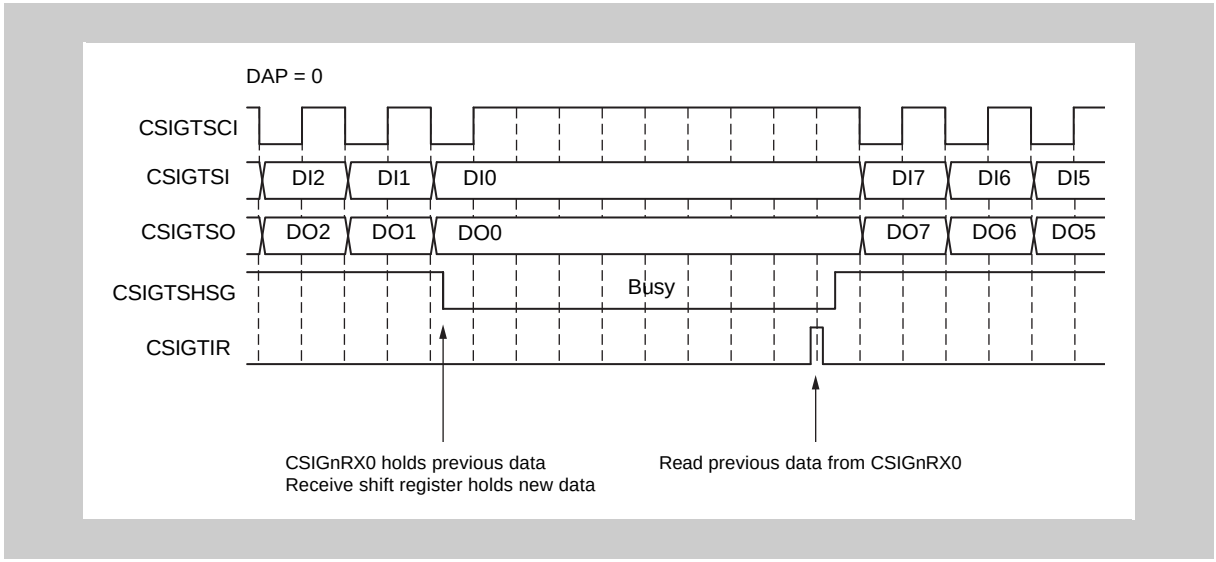



**Figure 19-17 Ready/busy signal from slave (CSIGnCFG0.CSIGnDAP = 0)**

As long as the slave is busy, the master has to wait (i.e. suspend the transmission clock). The slave sets CSIGTSHSG to high (“ready”) as soon as the reception register CSIGnRX0 has been read.

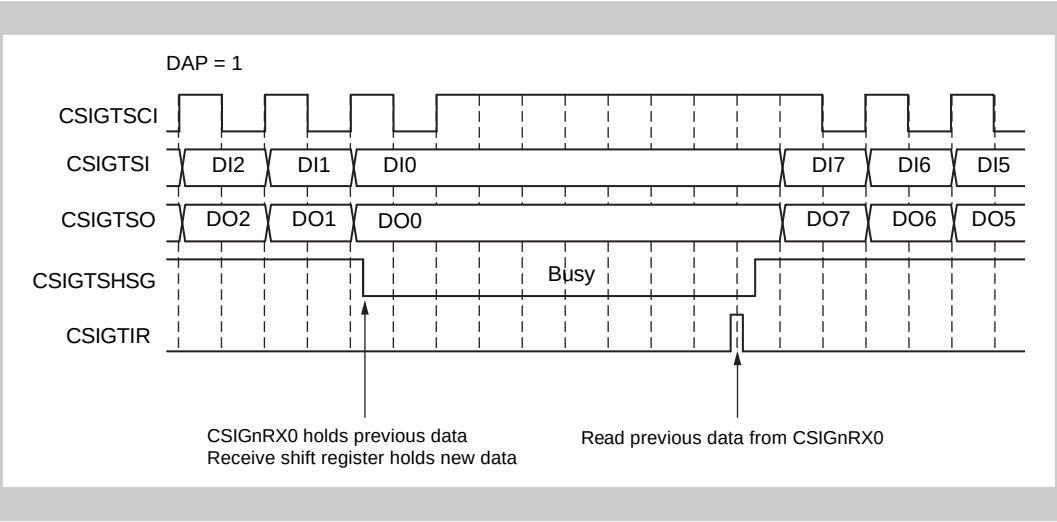



**Figure 19-18 Ready/busy signal from slave (CSIGnCFG0.CSIGnDAP = 1)**

### (2) Master mode

When the master detects CSIGTSHSG = 0, the following transfer is put on hold, and the master goes into wait status. It suspends the clock at CSIGTSCK.

The CSIGTSHSG level is checked at each half clock cycle of CSIGTSCK.

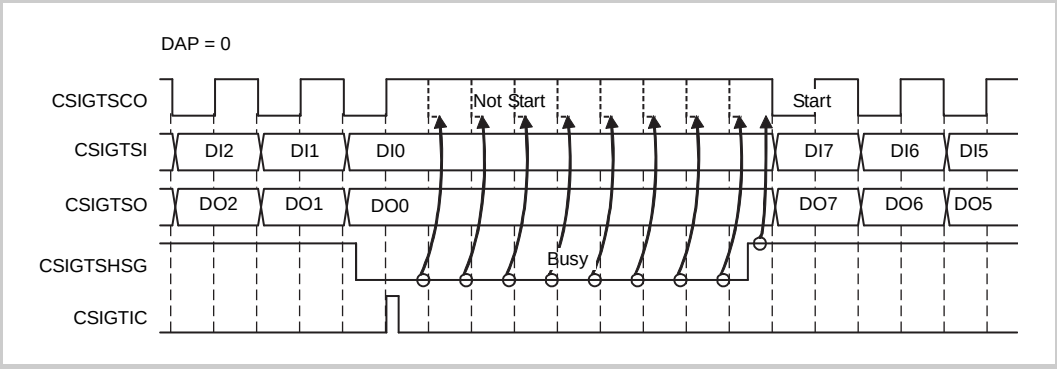



**Figure 19-19 Master's reaction to CSIGTSHSG (CSIGnCFG0.CSIGnDAP = 0)**

If the CSIGTSHSG low signal comes from the slave while data transfer is in progress, the serial clock is suspended after the transfer is complete.

The master resumes the communication as soon as CSIGTSHSG becomes high (slave is "ready").

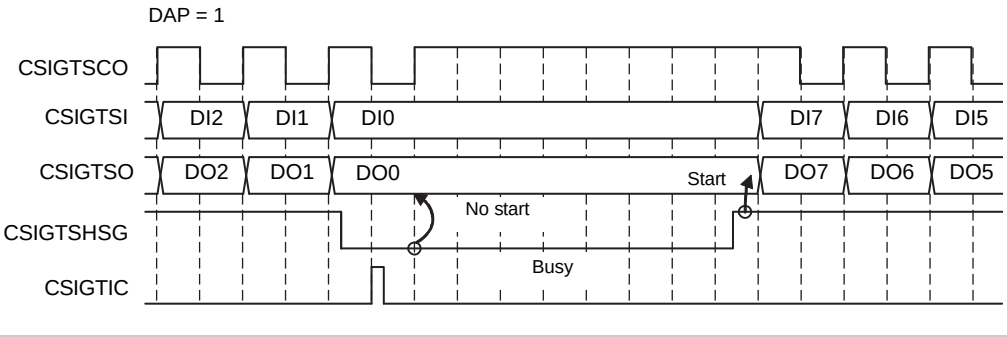



**Figure 19-20 Master's reaction to CSIGTSHSG (CSIGnCFG0.CSIGnDAP = 1)**

**Caution** If multiple slaves are connected, the master must only detect the CSIGTSHSG signal of the slave it has selected for communication.

CSIGTSHSG must be pulled down by the external slave before the next transfer starts. Even if the signal is pulled down by the slave during the transfer, the transfer will be finished.

### 19.3.10 Loop-back mode

Loop-back mode is a special mode for self-test. This feature is only available in master mode.

When this mode is active, the transmit and receive signals are internally connected, as shown in the figures below. The signals CSIGTSCK, CSIGTSO, and CSIGTSI are disconnected from the ports. In addition, the CSIGTSO output level is fixed to low, and CSIGTSCK is set to the inactive state, as defined by CSIGnCTL1.CSIGnCKR. The rest of CSIG works as in normal operation.

In order to test the CSIG, enable loop-back mode and carry out normal transfer operations. Then check that the received data is the same as the transmitted data.

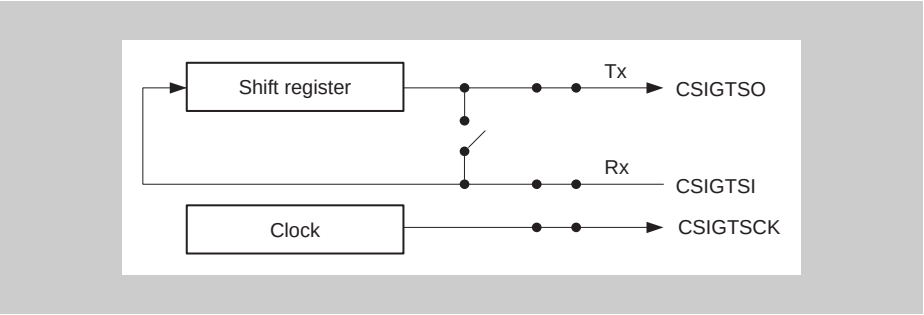



**Figure 19-21 Normal operation**

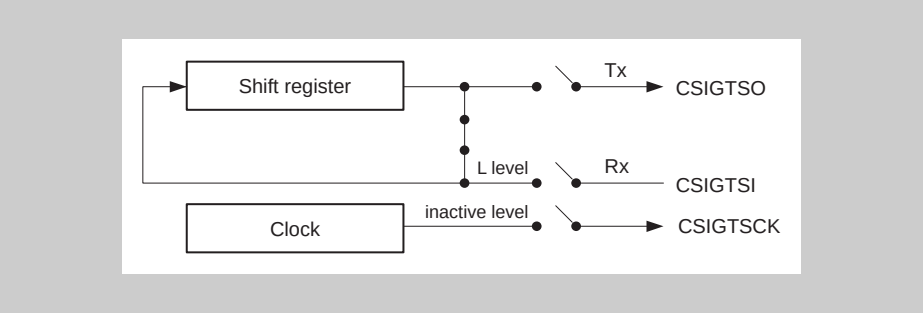



**Figure 19-22 Operation in loop-back mode**

### 19.3.11 Error detection

CSIG can detect three error types:

- Data consistency error (transmission data)
- Parity error (received data)
- Overrun error

Error check can be individually enabled/disabled for each type.

If one of these errors is detected, the interrupt CSIGTIRE is generated.

#### (1) Data consistency check

The purpose of the data consistency check is to ensure that the data physically sent to the output pin is identical to the original data that was copied to the shift register.

The data consistency check can be enabled/disabled by bit CSIGnCTL1.CSIGnDCS. It is not active if data transmission is disabled (CSIGnCTL0.CSIGnTXE = 0).

When the data consistency check is active, the data transferred from CSIGnTX0 to the shift register is copied to a separate register. In addition, the physical levels at CSIGTSO is read back via the CSIGTDCS signal into an own shift register.

After completion of the transmission, the data sent is compared with the original transmission data.

Mismatch is considered as a data consistency error:

- Interrupt CSIGTIRE is generated.
- Bit CSIGnSTR0.CSIGnDCE is set.

The function is illustrated in the following block diagram.

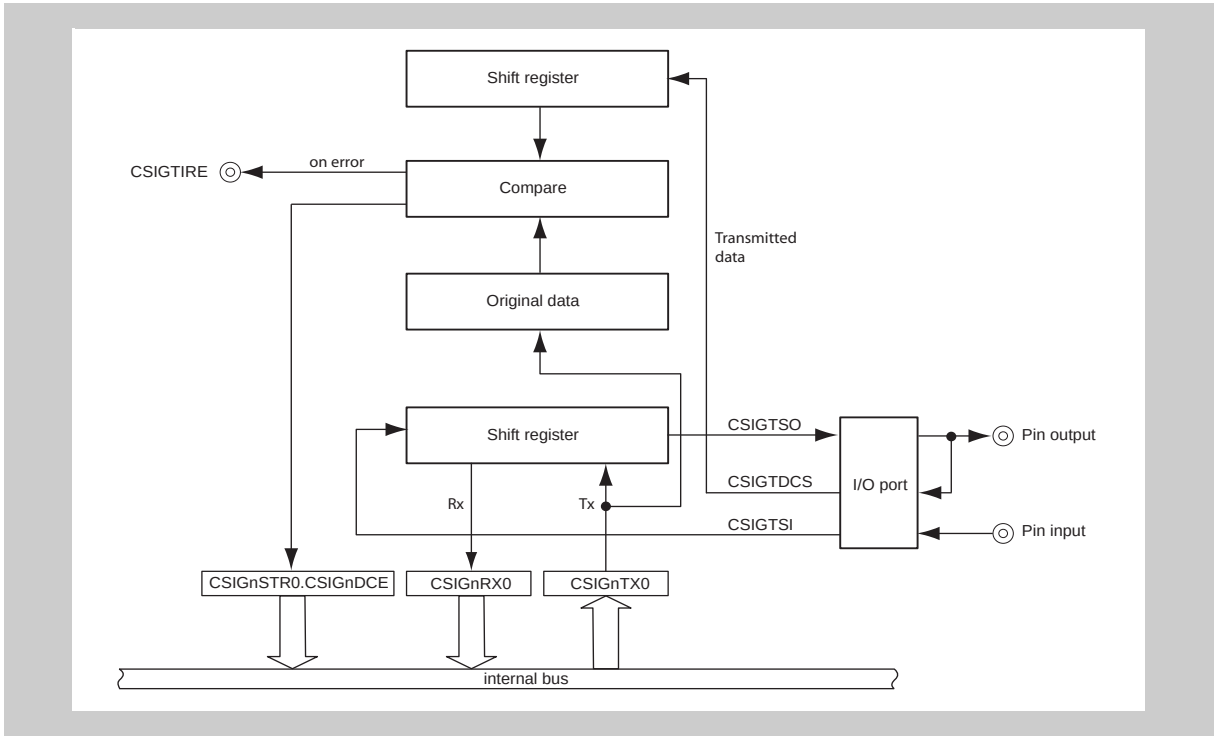



**Figure 19-23 Functional block diagram of the data consistency check**

**(2) Parity check**

Parity is a common mean to detect a single bit failure during data transmission. CSIG can append a parity bit to the last data bit (even if extended data length is used).

The use and type of parity is specified in CSIGnCFG0.CSIGnPS[1:0].

Parity check is enabled if CSIGnCFG0.CSIGnPS[1] = 1.

The parity bit is checked after reception is complete. If a parity error occurs:

- Interrupt CSIGTIRE is generated
- Bit CSIGnSTR0.CSIGnPE is set

The following figure shows an example.

Data length is 8 bits. The data transmitted is $05_H$ and $35_H$. Parity type is odd.

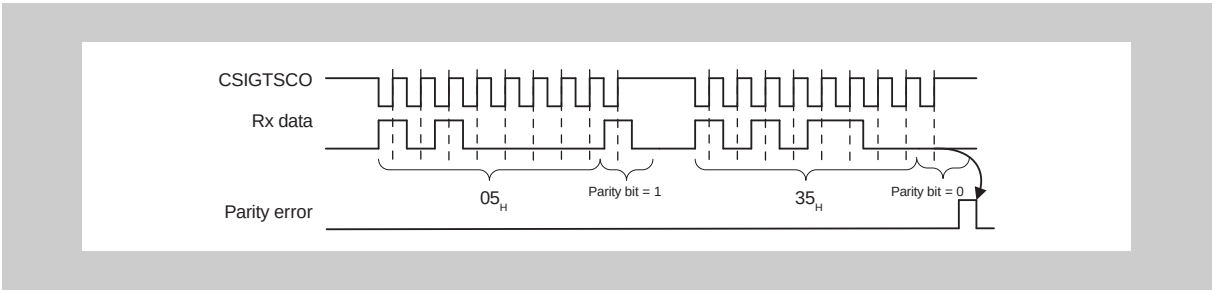



**Figure 19-24 Parity check example**

For the first 8 bits, the parity bit is 1. There is no parity error, because the total number of ones (including the parity bit) is odd.

For the second 8 bits, the parity bit is 0. This is detected as a parity error, because the total number of ones (including the parity bit) is even.

**(3) Overrun error**

This error occurs when previously received data still resides in the reception register CSIGnRX0, because it wasn't read, and new data is received.

The overrun error is not generated if data reception is disabled (CSIGnCTL0.CSIGnRXE = 0).

If overrun occurs:

- Interrupt CSIGTIRE is generated
- Bit CSIGnSTR0.CSIGnOVE is set
- Communication is stopped

The following figure illustrates the function.

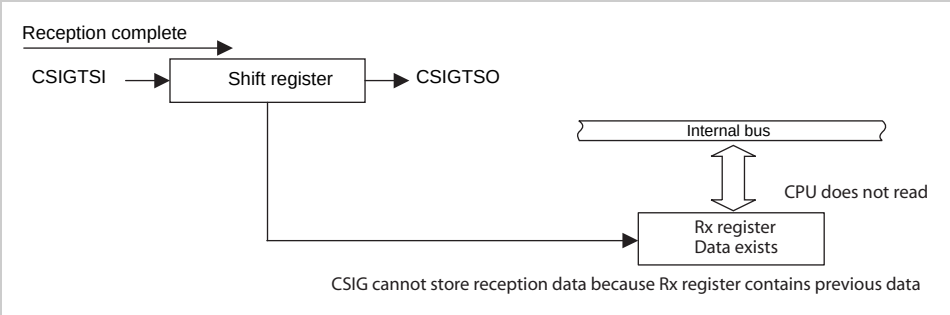



**Figure 19-25 Overrun error detection**

The following figure illustrates an example where:

- Rx data 3 was not read
- Rx data 4 was received, but cannot be stored

Thus an overrun error occurs.

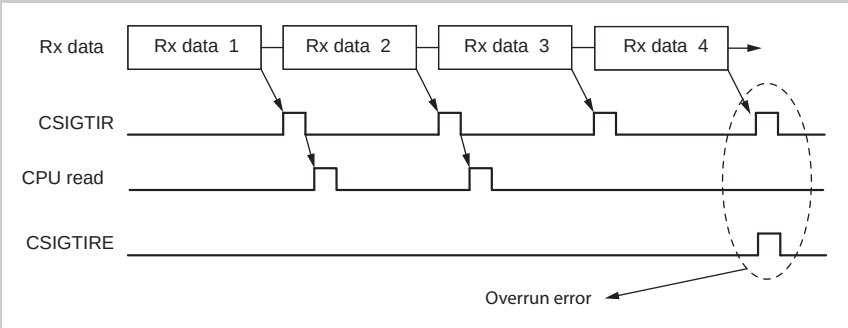



**Figure 19-26 Overrun error detection - example**

**Notes**

1. An Rx data overrun error can be avoided in slave mode by using the handshake.
   When handshake is used in slave mode, the receiver (slave) signals to the transmitter (master) that it is busy. The transmitter then waits until the receiver has read its reception register and is ready again.
   For details see *19.3.9 "Handshake function" on page 1049* .
2. In master mode no overrun error can occur, since the data clock CSIGTSCK is stopped, until the data in the CSIGnRX0 register is read.

## 19.4 CSIG Control Registers

The CSIGn is controlled and operated by means of the following registers:

**Table 19-10 CSIGn register overview**

| Register name | Shortcut | Address |
|---|---|---|
| Control register 0 | CSIGnCTL0 | <CSIGn_base> |
| Control register 1 | CSIGnCTL1 | <CSIGn_base> + $10_H$ |
| Control register 2 | CSIGnCTL2 | <CSIGn_base> + $14_H$ |
| Status register 0 | CSIGnSTR0 | <CSIGn_base> + $04_H$ |
| Status clear register 0 | CSIGnSTCR0 | <CSIGn_base> + $08_H$ |
| Rx-only mode control register 0 | CSIGnBCTL0 | <CSIGn_base> + $1000_H$ |
| Configuration register 0 | CSIGnCFG0 | <CSIGn_base> + $1010_H$ |
| Transmit data register 0 for word access | CSIGnTX0W | <CSIGn_base> + $1004_H$ |
| Transmit data register 0 for half word access | CSIGnTX0H | <CSIGn_base> + $1008_H$ |
| Receive data register 0 | CSIGnRX0 | <CSIGn_base> + $100C_H$ |
| Emulation register | CSIGnEMU | <CSIGn_base> + $0018_H$ |

**<CSIGn_base>** The base addresses <CSIGn_base> of the CSIGn is defined in the first section of this chapter under the key word “Register addresses”.

### (1) CSIGnCTL0 - CSIG control register 0

This register controls the operation clock and enables/disables transmission/ reception.

**Access** This register can be read/written in 8-bit units.

**Address** <CSIGn_base> + $00_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CSIGn PWR | CSIGn TXE | CSIGn RXE | 0 | 0 | 0 | 0 | 0[a] |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

a) The default value “0” of bit 0 of CSIGnCTL0 must be changed to “1” before the CSIGn is used.

**Caution** The default value “0” of bit 0 of CSIGnCTL0 must be changed to “1” before the CSIGn is used.

**Table 19-11 CSIGnCTL0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | CSIGnPWR | Controls operation clock:<br>0: Stop operation clock<br>1: Provide operation clock<br>Clearing CSIGnPWR to 0 resets the internal circuits, stops operation, and sets the CSIG to standby state. No clock is provided to internal circuits.<br>If CSIGnPWR is cleared during communication, ongoing communication is aborted. A restart of the communication is then required. |
| 6 | CSIGnTXE | Enables/disables transmission:<br>0: Transmission disabled<br>1: Transmission enabled |
| 5 | CSIGnRXE | Enables/disables reception:<br>0: Receive disabled<br>1: Receive enabled |
| 0 | Bit 0 | **Caution** The default value “0” of this must be changed to “1” before the CSIGn is used. |

**Cautions**

1. Do not modify CSIGnRXE or CSIGnTXE while CSIGnPWR = 0. However both bits can be modified in the same write operation when setting CSIGnPWR = 1.
2. Do not modify CSIGnRXE or CSIGnTXE while a data transmission is pending or ongoing, i.e. if CSIGnSTR0.CSIGnTSF = 1.

### (2) CSIGnCTL1 - CSIG control register 1

This register specifies the interrupt timing and the interrupt delay mode. It enables/disables extended data length control, data consistency check, loop-back mode, handshake function, and slave select function.

**Access** This register can be read/written in 32-bit units.

**Address** <CSIGn_base> + $10_H$

**Initial Value** 0000 0000H

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | CSIGnCKR | CSIGnSLIT |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | CSIGnEDLE | 0 | CSIGnDCS | 0 | CSIGnLBM | CSIGnSIT | CSIGnHSE | CSIGnSSE |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Caution** Changing the contents of this register is only permitted when CSIGnCTL0.CSIGnPWR = 0.

**Table 19-12 CSIGnCTL1 register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 17 | CSIGnCKR | Selects the initial level of the data clock CSIGTSCK<br>0: Initial level of CSIGTSCK is high<br>1: Initial level of CSIGTSCK is low<br>For the relation between CSIGTSCK, CSIGTSO and the sample point, refer to the description of the CSIGnCFG0.CSIGnDAP bit. |
| 16 | CSIGnSLIT | Selects the timing of interrupt CSIGTIC:<br>0: Normal interrupt timing (interrupt is generated after the transfer)<br>1: Interrupt generation when CSIGnTX0 is free for next data.<br>For details refer to *1 “CSIGTIC communication interrupt” on page 1047* . |
| 7 | CSIGnEDLE | Enables/disables extended data length (EDL) mode:<br>0: Extended data length mode disabled<br>1: Extended data length mode enabled<br>For details refer to *2 “Data length selection with extended data length” on page 1043* . |
| 5 | CSIGnDCS | Enables/disables data consistency check:<br>0: Data consistency check disabled<br>1: Data consistency check enabled<br>For details refer to *1 “Data consistency check” on page 1053* . |
| 3 | CSIGnLBM | Controls loop-back mode (LBM):<br>0: Loop-back mode deactivated<br>1: Loop-back mode activated<br>For details refer to *19.3.10 “Loop-back mode” on page 1052* . |

**Table 19-12 CSIGnCTL1 register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | CSIGnSIT | Selects interrupt delay mode:<br>0: No delay<br>1: Half clock delay for all interrupts<br>This bit is only valid in master mode. In slave mode, no delay is generated.<br>For details refer to *19.3.8 "CSIG interrupts" on page 1047* . |
| 1 | CSIGnHSE | Enables/disables handshake mode:<br>0: Handshake function disabled<br>1: Handshake function enabled<br>For details refer to *19.3.9 "Handshake function" on page 1049* . |
| 0 | CSIGnSSE | Enables/disables slave select function:<br>0: Input signal $\overline{\text{CSIGTSSI}}$ is ignored<br>1: Input signal $\overline{\text{CSIGTSSI}}$ is enabled<br>If the slave select function is not used, this bit must be set to 0 (see also *19.3.2 "Master/slave connections" on page 1039* ). |

Details about CSIGnCTL1.CSIGnSSE:

**Table 19-13 Operation of the slave select function during reception**

| CSIGnCTL0. CSIGnRXE | CSIGnCTL1. CSIGnSSE | $\overline{\text{CSIGTSSI}}$ | Receive operation |
|---|---|---|---|
| 0 | - | - | Reception disabled |
| 1 | 0 | - | Possible |
| 1 | 1 | 0 | Possible |
| 1 | 1 | 1 | Impossible |

**Table 19-14 Operation of the slave select function during transmission**

| CSIGnCTL0. CSIGnTXE | CSIGnCTL1. CSIGnSSE | $\overline{\text{CSIGTSSI}}$ | Transmit operation |
|---|---|---|---|
| 0 | - | - | Transmission disabled |
| 1 | 0 | - | Possible |
| 1 | 1 | 0 | Possible |
| 1 | 1 | 1 | Impossible |

### (3) CSIGnCTL2 - CSIG control register 2

This register selects the communication clock.

**Access** This register can be read/written in 16-bit units.

**Address** <CSIGn_base> + $14_H$

**Initial Value** $E000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSIGn PRS[2:0] | | | 0 | CSIGn BRS[11:0] | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Caution** Changing the contents of this register is only permitted when CSIGnCTL0.CSIGnPWR = 0.

**Table 19-15 CSIGnCTL2 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 13 | CSIGnPRS [2:0] | Selects the value m of the prescaler: |
| 11 to 0 | CSIGnBRS [11:0] | Selects the baud rate (m = CSIGnPRS[2:0]): |

| CSIGn PRS2 | CSIGn PRS1 | CSIGn PRS0 | Prescaler output (PRSOUT) |
|---|---|---|---|
| 0 | 0 | 0 | PCLK (master mode) |
| 0 | 0 | 1 | PCLK / 2 (master mode) |
| 0 | 1 | 0 | PCLK / 4 (master mode) |
| 0 | 1 | 1 | PCLK / 8 (master mode) |
| 1 | 0 | 0 | PCLK / 16 (master mode) |
| 1 | 0 | 1 | PCLK / 32 (master mode) |
| 1 | 1 | 0 | PCLK / 64 (master mode) |
| 1 | 1 | 1 | External clock via CSIGTSCI (slave mode) |

| CSIGnBRS[11:0] | Baud rate at CSIGTSCK |
|---|---|
| 0 | BRG is stopped |
| 1 | PCLK / ($2^m$ × 1 × 2) |
| 2 | PCLK / ($2^m$ × 2 × 2) |
| 3 | PCLK / ($2^m$ × 3 × 2) |
| 4 | PCLK / ($2^m$ × 4 × 2) |
| ... | ... |
| 4095 | PCLK / ($2^m$ × 4095 × 2) |

### (4) CSIGnSTR0 - CSIG status register 0

This register indicates the status of the CSIG.

**Access** This register can be read in 32-bit units.

**Address** <CSIGn_base> + $04_H$

**Initial Value** $0000\ 0010_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | CSIGn TSF | 0 | 0 | 1 | CSIGn DCE | 0 | CSIGn PE | CSIGn OVE |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 19-16 CSIGnSTR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | CSIGnTSF | Transfer status flag:<br>0: Idle state<br>1: Transmission is in progress or being prepared<br>Setting and clearing of this bit is as follows: |
| 3 | CSIGnDCE | Data consistency check error flag:<br>0: No data consistency error detected<br>1: Data consistency error detected<br>This bit is cleared by writing 1 to CSIGnSTCR0.CSIGnDCEC.<br>This bit is initialized when CSIGnCTL0.CSIGnPWR changes from 0 to 1 or from 1 to 0. |
| 1 | CSIGnPE | Parity error flag:<br>0: No parity error detected<br>1: Parity error detected<br>This bit is cleared by writing 1 to CSIGnSTCR0.CSIGnPEC.<br>This bit is initialized when CSIGnCTL0.CSIGnPWR changes from 0 to 1 or from 1 to 0. |
| 0 | CSIGnOVE | Overrun error flag:<br>0: No overrun error detected<br>1: Overrun error detected<br>This bit is cleared by writing 1 to CSIGnSTCR0.CSIGnOVEC.<br>This bit is initialized when CSIGnCTL0.CSIGnPWR changes from 0 to 1 or from 1 to 0. |

| Master mode | Set by | Cleared by |
|---|---|---|
| Tx only mode | Writing to transmit register | Rising edge of CSIGTIC |
| Tx / Rx mode | | Rising edge of CSIGTIR |
| Rx only mode | | |

| Slave mode | Set by | Cleared by |
|---|---|---|
| Tx only mode | Writing to transmit register | Rising edge of CSIGTIC |
| Tx / Rx mode | | Rising edge of CSIGTIR |
| Rx only mode | CSIGTSCK | |

### (5) CSIGnSTCR0 - CSIG status clear register 0

This register clears the status flags of the CSIGnSTR0 status register.

**Access** This register can be read/written in 16-bit units.

**Address** <CSIGn_base> + $08_H$

**Initial Value** Reading this register returns always $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | CSIGn DCEC | 0 | CSIGn PEC | CSIGn OVEC |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 19-17 CSIGnSTCR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 3 | CSIGnDCEC | 0: No operation. Read value is always 0.<br>1: Clear data consistency error flag (CSIGnSTR0.CSIGnDCE) |
| 1 | CSIGnPEC | 0: No operation. Read value is always 0.<br>1: Clear parity error flag (CSIGnSTR0.CSIGnPE) |
| 0 | CSIGnOVEC | 0: No operation. Read value is always 0.<br>1: Clear overrun error flag (CSIGnSTR0.CSIGnOVE) |

### (6) CSIGnBCTL0 - CSIG Rx-only mode control register 0

This register enables/disables the data transfer in Rx-only mode.

**Access** This register can be read/written in 8-bit units.

**Address** <CSIGn_base> + $1000_H$

**Initial Value** $01_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | CSIGnSCE |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 19-18 CSIGnBCTL0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | CSIGnSCE | Disables/enables next data reception start by reading CSIGnRX0:<br>0: Next reception disabled<br>1: Next reception enabled<br>For details refer to *2 "Receive-only mode" on page 1042* and *19.3.7 "Communication in slave mode" on page 1046* . |

**Caution** CSIGnSCE must be set to 0 if transmit mode is enabled (CSIGnCTL0.CSIGnTXE = 1).

### (7) CSIGnCFG0 - CSIG configuration register 0

This register configures the communication protocol – data length, parity, transfer direction, clock phase, and data phase.

**Access** This register can be read/written in 32-bit units.

**Address** <CSIGn_base> + $1010_H$

**Initial Value** $0000\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | CSIGn PS[1:0] | | CSIGn DLS[3:0] | | | | 0 | 0 | 0 | 0 | 0 | CSIGn DIR | 0 | CSIGn DAP |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

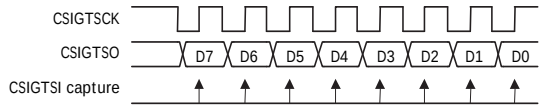
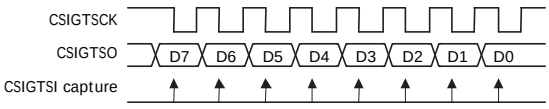
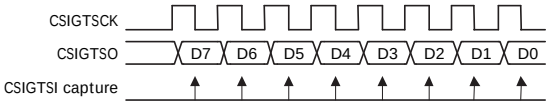
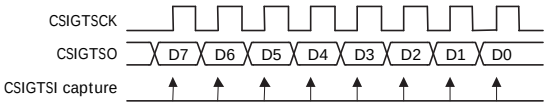
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Caution** Changing the contents of this register is only permitted when CSIGnCTL0.CSIGnPWR = 0.

**Table 19-19 CSIGnCFG0 register contents (1/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>29 to 28</td><td>CSIGnPS [1:0]</td><td>Specifies parity:
<table>
<tr><th>CSIGn PS1</th><th>CSIGn PS0</th><th>Transmission</th><th>Reception</th></tr>
<tr><td>0</td><td>0</td><td>No parity transmitted</td><td>No parity expected</td></tr>
<tr><td>0</td><td>1</td><td>Add parity bit fixed at 0</td><td>Parity bit is expected but not judged</td></tr>
<tr><td>1</td><td>0</td><td>Add odd parity</td><td>Odd parity bit is expected</td></tr>
<tr><td>1</td><td>1</td><td>Add even parity</td><td>Even parity bit is expected</td></tr>
</table></td></tr>
<tr><td>27 to 24</td><td>CSIGnDLS [3:0]</td><td>Specifies data length:<br>0: Data length is 16 bits<br>1: Data length is 1 bit<br>2: Data length is 2 bits<br>...<br>15: Data length is 15 bits<br><br>Caution Data lengths of less than 7 bits is only allowed in combination with the extended data length function.<br>It is forbidden to transmit two consecutive data with a data length of less than 7 bits.</td></tr>
<tr><td>18</td><td>CSIGnDIR</td><td>Selects the serial data direction:<br>0: Data is sent/received with MSB first<br>1: Data is sent/received with LSB first</td></tr>
</table>

**Table 19-19 CSIGnCFG0 register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>16</td><td>CSIGnDAP</td><td>Data phase selection bit<br>The control bits CSIGnDAP and CSIGnCTL1.CSIGnCKR determine the CSIGTSCK clock edge<br>• to shift serial data out via CSIGTSO<br>• to sample serial input data at CSIGTSI.<br>The following diagram shows the relation between CSIGTSCK, CSIGTSO and the sample point of CSIGTSI, depending on CSIGnCTL1.CSIGnCKR and CSIGnDAP:
<table>
<tr><th>CSIGn CKR</th><th>CSIGn DAP</th><th>Clock and data phase selection</th></tr>
<tr><td>0</td><td>0</td><td> 
</td></tr>
<tr><td>0</td><td>1</td><td> 
</td></tr>
<tr><td>1</td><td>0</td><td> 
</td></tr>
<tr><td>1</td><td>1</td><td> 
</td></tr>
</table></td></tr>
</table>

### (8) CSIGnTX0W - CSIG transmission register 0 for word access

This register stores the transmission data. It has to be used when the extended data length function is enabled (CSIGnCTL1.SCIGnEDLE = 1).

**Access** This register can be read/written in 32-bit units.

**Address** <CSIGn_base> + $1004_H$

**Initial Value** $0000\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | CSIGnEDL | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSIGnTX[15:0] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 19-20 CSIGnTX0W register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 29 | CSIGnEDL | Specifies the extended data length configuration:<br>0: Normal operation<br>1: Extended data length activated<br>The associated data is transmitted as 16-bit data.<br>This bit can only be set if CSIGnCTL1.CSIGnEDLE = 1. It is automatically cleared if CSIGnCTL1.CSIGnEDLE is cleared. |
| 15 to 0 | CSIGn TX[15:0] | Data to be transmitted |

### (9) CSIGnTX0H - CSIG transmission register 0 for half word access

This register stores the transmission data. It can be used when the Extended Data Length function is disabled (CSIGnCTL1.CSIGnEDLE = 0).

**Access** This register can be read/written in 16-bit units.

**Address** <CSIGn_base> + $1008_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSIGnTX[15:0] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 19-21 CSIGnTX0H register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | CSIGn TX[15:0] | Data to be transmitted |

### (10) CSIGnRX0 - CSIG reception register 0

This register stores the received data.

**Access** This register can be read in 16-bit units.

**Address** <CSIGn_base> + $100C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSIGn RX[15:0] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 19-22 CSIGnRX0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | CSIGn RX[15:0] | Received data<br>This register is initialized when CSIGnCTL0.CSIGnPWR changes from 0 to 1 or from 1 to 0. |

### (11) CSIGnEMU - CSIG emulation register

This register controls whether the CSIGn can be stopped during emulation, for instance upon a breakpoint hit.

**Access** This register can be read/written in 8-bit units.

**Address** <CSIGn_base> + $18_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CSIGn SVSDIS | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 19-23 CSIGnEMU register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | CSIGn SVSDIS | Emulation control<br>0: CSIGn can be stopped during emulation<br>1: CSIGn continuous operating during emulation |

## 19.5 Operating Procedure Example

In the following a transmit/receive example in master mode in combination with a DMA is described.

The following instructions are based on the assumption that:

- Transmission data length is 8 bits (CSIGnCFG0.CSIGnDLS[3:0] = $1000_B$)
- MSB is transmitted first (CSIGnCFG0.CSIGnDIR = 0)
- CSIGTIC interrupt at the end of the transfer (CSIGnCTL1.CSIGnSLIT = 0)
- Normal clock and data phase (CSIGnCTL1.CSIGnCKR = 0, CSIGnCFG0.CSIGnDAP = 0)
- The number of data packets is 10 (0 to 9)

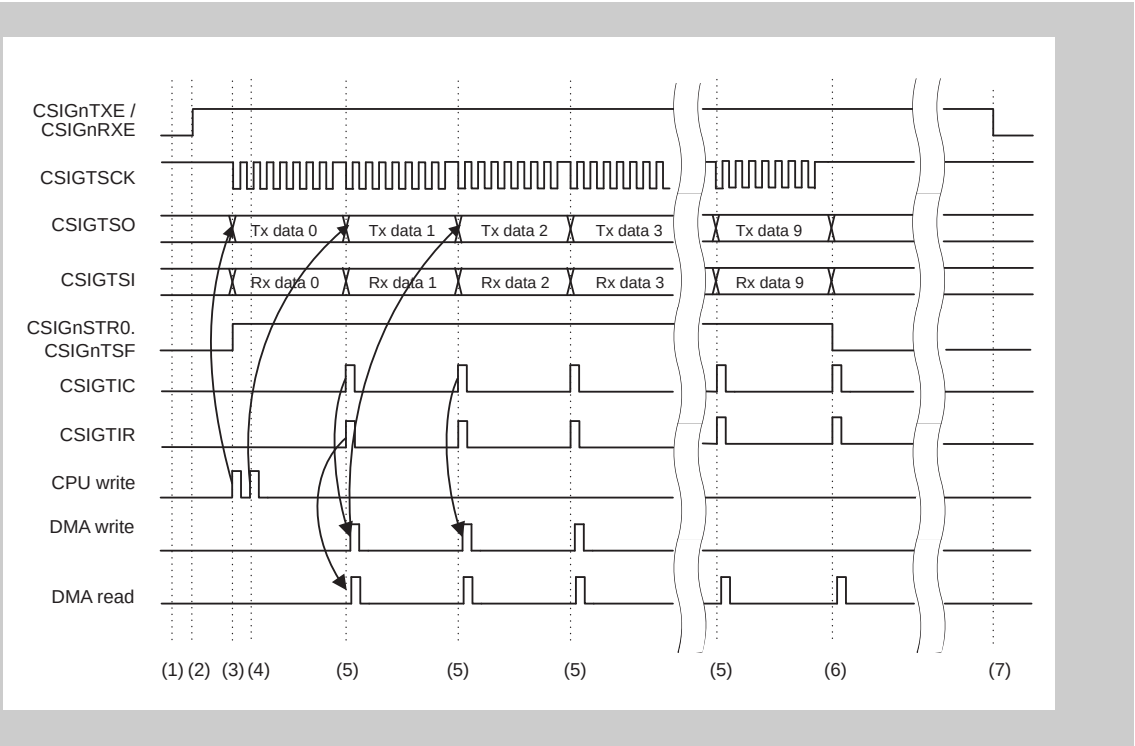



**Figure 19-27 Communication in master mode**

Procedure:

1. Configure the communication protocol in register CSIGnCFG0.
2. In the CSIGnCTL0 register, set bits CSIGnPWR = 1 (enable the clock), CSIGnTXE = 1 (enable transmission), CSIGnRXE = 1 (enable reception). The data output CSIGTSO is enabled.
3. Write the first data packet to be sent to the transmission register CSIGnTX0H. Transmission starts automatically when the first data is available.
4. Write the second data to CSIGnTX0H. Writing the second packet immediately after the first one avoids unnecessary delays between the packets.
5. After every packet that has been transmitted, the interrupts CSIGTIC and CSIGTIR are generated. CSIGTIC indicates that the next packet can be

written to CSIGnTX0H. CSIGTIR indicates that the reception register CSIGnRX0 must be read.
In this example, CPU write and DMA write are equivalent.

6. No more write action is required after completion of packet 8. Packet 9 (the last packet) has been written in advance.
However, the reception register CSIGnRX0 must be read after completion of packets 8 and 9.

7. To finally disable the transmit/receive operation, clear CSIGnCTL0.CSIGnTXE and CSIGnCTL0.CSIGnRXE.

# Chapter 20 $I^2C$ Interface (IICB)

This chapter contains a generic description of the $I^2C$ Interface (IICB).

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 20.1 V850E2/Fx4-G IICB Features

**Instances**

This microcontroller has following number of instances of the $I^2C$ Interface.

**Table 20-1 Instances of IICB**

| $I^2C$ Interface | |
|---|---|
| Instances | 1 |
| Names | IICB0 |

**Instances index n**

Throughout this chapter, the individual instances of a $I^2C$ Interface are identified by the index “n” (n = 0), for example, IICBnCTL0 for the IICBn control register 0.

**Register addresses**

All IICBn register addresses are given as address offsets from the individual base address <IICBn_base>.
The base address <IICBn_base> of each IICBn is listed in the following table:

**Table 20-2 Register base addresses <IICBn_base>**

| IICBn instance | <IICBn_base> address |
|---|---|
| IICB0 | FF82 $0000_H$ |

**Clock supply** All I2C Interfaces provide one clock input:

**Table 20-3 IICB clock supply**

| IICBn instance | Clock | Connected to |
|---|---|---|
| IICB0 | PCLK | Clock Controller CKSCLK_028 |

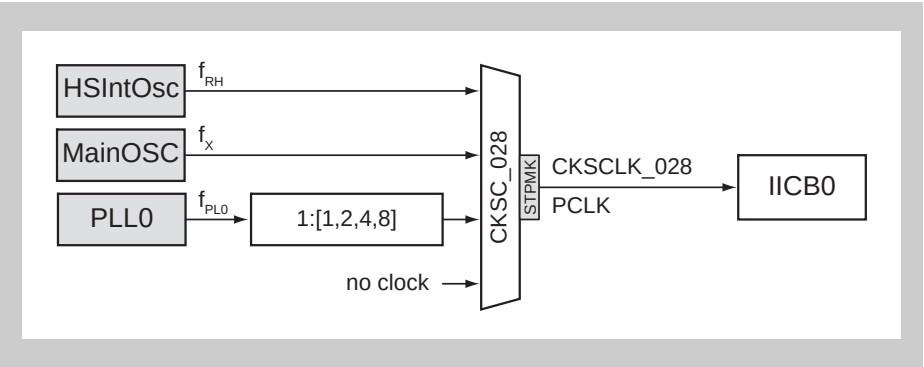



**Figure 20-1 IICB clock supply**

**Interrupts and DMA** The I2C Interfaces can generate following interrupt and DMA requests:

**Table 20-4 IICB interrupt and DMA requests**

| IICBn signals | Function | Connected to |
|---|---|---|
| **IICB0:** | | |
| IICBTIA | Data transmission/reception interrupt | Interrupt Controller INTIICB0IA<br>DMA Controller trigger 43 |

**IICB H/W reset** The I2C Interfaces and their registers are initialized by the following reset signal:

**Table 20-5 IICBn reset signal**

| IICBn | Reset signal |
|---|---|
| IICBn | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

**I/O signals** The I/O signals of the I2C Interfaces are listed in the following table:

**Table 20-6 IICB I/O signals**

| IICBn signals | Function | Connected to |
|---|---|---|
| **IICB0:** | | |
| SCL | IICB0 clock signal | Port IICB0SCL |
| SDA | IICB0 data/address signal | Port IICB0SDA |

## 20.2 I²C Interface Port Settings

The I²C interface function requires to define the ports for the IICBnSCL and IICBnSDA signals as input and open drain output pins simultaneously.

In the following table the port configuration register bits are listed to be set up properly for port m of the port group n, if it shall be used for I²C interface signals IICBnSCL or IICBnSDA.

The bold printed values in grey cells differ from the initial register values after reset release and thus have to be changed.

**Table 20-7 I²C Interface port settings**

| Register | | Value to set | Initial value | Function |
|---|---|---|---|---|
| **Port function configuration registers:** | | | | |
| PMCn_m | = | **1** | 0 | alternative mode |
| PIPCn_m | = | 0 | 0 | S/W I/O control |
| PMn_m | = | **0** | 1 | output mode |
| PIBCn_m | = | x | 0 | overruled by PMCn_m = 1 |
| PFCn_m, PFCEn_m | = | **1 or 0** | 0 | select correct alternative function |
| **Port data input/output registers:** | | | | |
| PBDCn_m | = | **1** | 0 | bi-directional I/O |
| **Configuration of electrical characteristics registers:** | | | | |
| PUn_m | = | x | 0 | no pull-up resistor, overruled by PMn_m = 0 |
| PDn_m | = | x | 0 | no pull-down resistor, overruled by PMn_m = 0 |
| PODCn_m | = | **1** | 0 | open drain output |
| PISn_m, PISEn_m | = | **1 or 0** | 0 | input characteristic can be selected based on application signal quality |

## 20.3 Functional Overview

| | |
|---|---|
| **Operating mode** | Standard mode (SCL clock frequency: 100 kHz max.)<br>Fast mode (SCL clock frequency: 400 kHz max.) |
| **Transfer mode** | Single transfer mode |
| **Pin configuration** | SCLn: Serial clock pin<br>SDAn: Serial transmit/receive data pin |
| **Interrupt request signal** | Data transmit/receive interrupt request signal (IICBTIAn) |
| **Communication data length** | 8 bits |
| **Multimaster support** | Multiple masters can control the bus simultaneously. |
| **SCLn level width** | The high-level width and low-level width of the serial clock signal (SCLn) can be changed. |
| **Automatic detection** | The start and stop conditions can be detected automatically. |

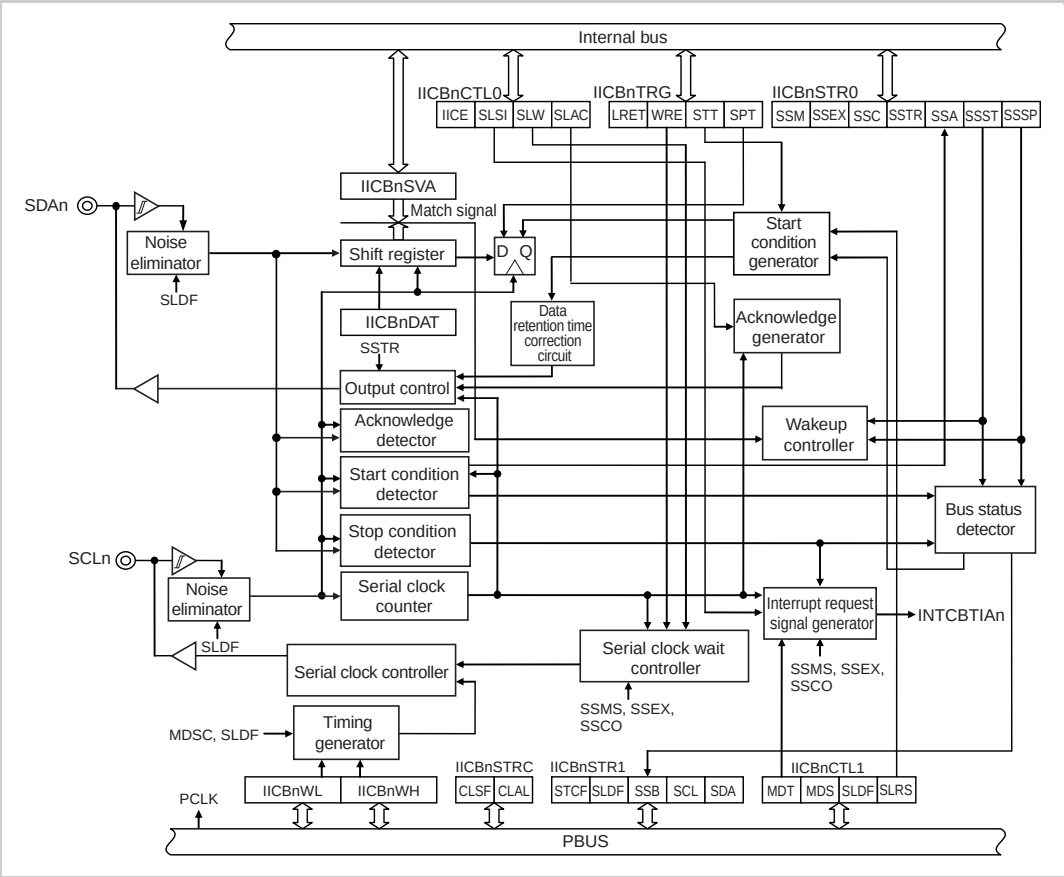



**Figure 20-2 Block diagram of IICBn**

# 20.4 I2C Bus Mode Functions

## 20.4.1 Pin configuration

The serial clock pin (SCLn) and serial data bus pin (SDAn) are configured as follows.

SCLn... This pin is used for serial clock input and output.
This pin is an N-ch open-drain output for both master and slave devices. Input is Schmitt input.

SDAn... This pin is used for serial data input and output.
This pin is an N-ch open-drain output for both master and slave devices. Input is Schmitt input.

Because the outputs of the serial clock line and serial data bus line are N-ch open-drain outputs, an external pull-up resistor must be connected to these lines.

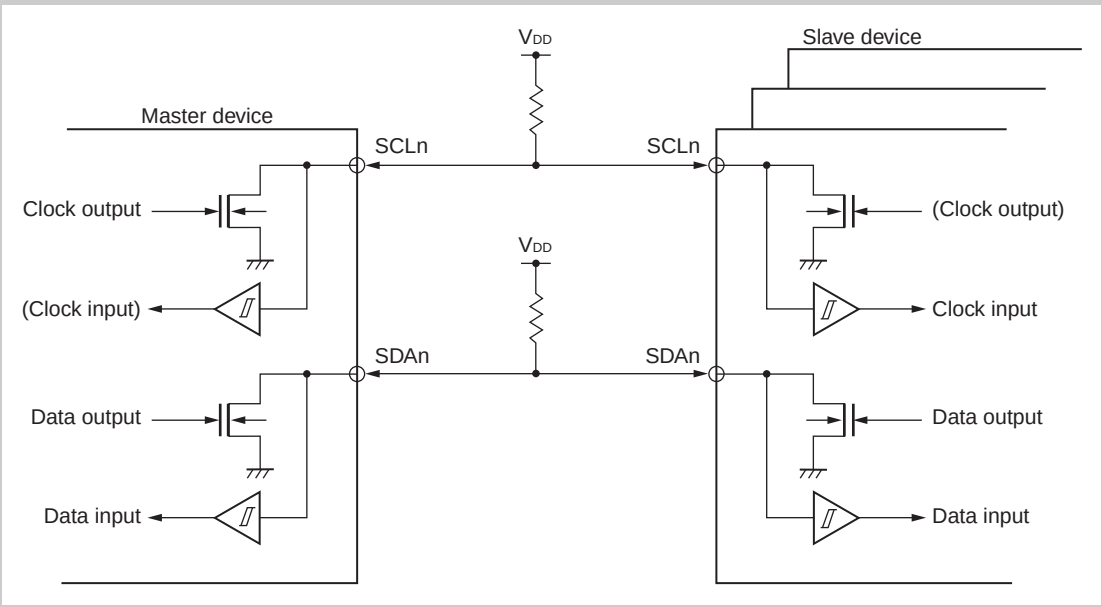



**Figure 20-3 Pin configuration diagram**

## 20.5 I2C Bus Definition

This section describes the I2C bus's serial data communication format and the signals used by the I2C bus.

*Figure 20-4 "I2C bus serial data transfer timing"* shows the transfer timing for the "start condition", "address", "transfer direction specification", "data", and "stop condition", which are output onto the I2C bus's serial data bus.

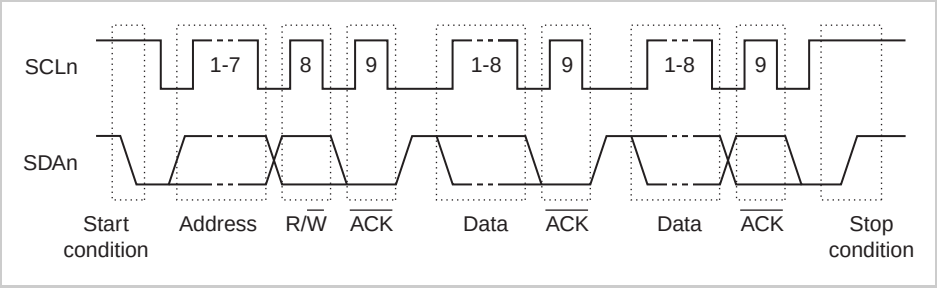



**Figure 20-4 I2C bus serial data transfer timing**

The start condition, slave address, and stop condition are output by the master device.

$\overline{\text{ACK}}$ can be output by either the master or slave device. (Normally, it is output by the device that receives 8-bit data.)

The serial clock signal (SCLn) is continuously output by the master device. In the slave device, the low-level period of the SCLn signal can be extended to insert a wait.

### 20.5.1 Start Condition

The start condition is met if the SDAn signal level changes from high to low while the SCLn signal is high. The start condition is output when the master device starts serial data transfer to a slave device. When the IICBn is in the slave mode, it detects the start condition.

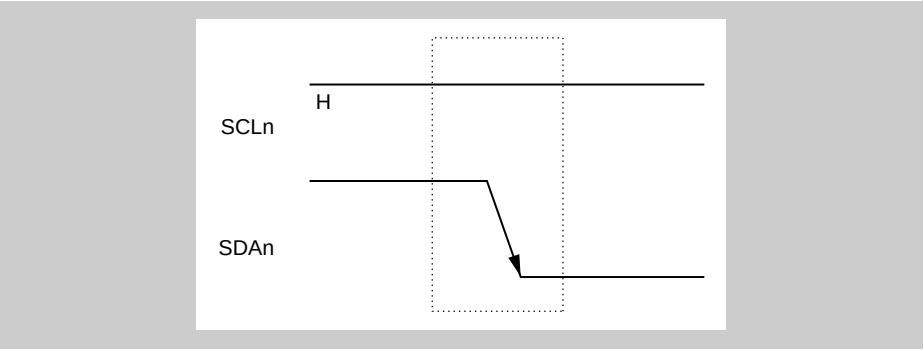



**Figure 20-5 Start condition**

### 20.5.2 Addresses

The 7 bits of data following the start condition are defined as an address.

An address is a 7-bit data segment that is output in order to select one of the slave devices that are connected to the master device via the bus lines. Therefore, each slave device connected via the bus lines must have a unique address.

The slave device checks whether the 7-bit data matches its own address. If they match, that slave device is selected as the communication destination and communicates with the master device until the master device outputs another start condition or a stop condition.

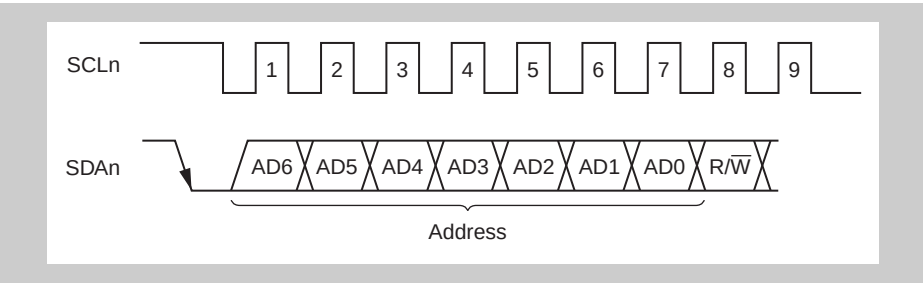



**Figure 20-6 Addresses**

### 20.5.3 Extension code

When the higher 4 bits of the address are 0000 or 1111, these bits are called extension code. *Table 20-8 "Extension code bit definitions"* lists the bit definitions of extension code.

**Table 20-8 Extension code bit definitions**

| Slave address | R/W bit | Description |
|---|---|---|
| 0000 000 | 0 | General call address |
| 0000 000 | 1 | Start byte |
| 0000 001 | × | CBUS address |
| 0000 010 | × | Address reserved for different bus format |
| 0000 011 | × | Reserved for future use |
| 0000 1xx | × | HS mode master code[a] |
| 1111 0xx | × | 10-bit slave address specification |
| 1111 1xx | × | Reserved for future use |

a) The HS mode cannot be used with this IICB module.

### 20.5.4 Transfer direction specification

After the 7-bit address data, the master device transmits 1 bit that specifies the transfer direction.

If this transfer direction specification bit is 0, it indicates that the master device transmits data to a slave device. If this bit is 1, it indicates that the master device receives data from a slave device.

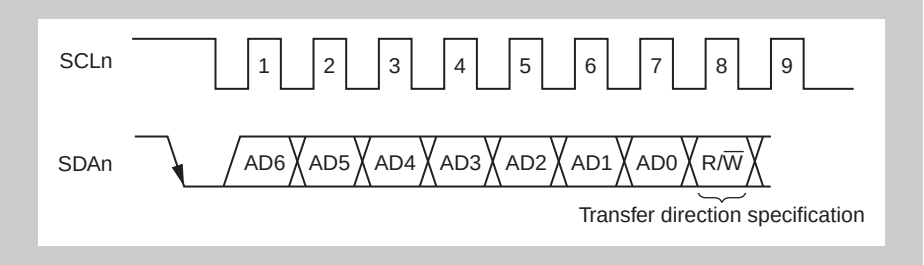



**Figure 20-7 Transfer direction specification**

### 20.5.5 Acknowledge (ACK)

The 1-bit data after the transfer direction bit (R/W) and the 1-bit data after the 8-bit data during address transfer are defined as an acknowledge signal (ACK). ACK is used to check the serial data status of the transmitting and receiving devices.

The receiving device returns ACK after receiving 8-bit data.

The transmitting device normally receives ACK after transmitting 8-bit data. If the transmitting device receives ACK from the receiving device, it continues processing assuming that the transmitted data is normally received.

If the master device is the receiving device and receives the final data, it does not return ACK and outputs a stop condition. If the slave device is the receiving device and does not return ACK, the master device outputs either a stop condition or a restart condition and then stops the current transmission. Failure to return ACK may be caused by the following factors:

- The transmitted data has not been received normally.
- The final data has been received.
- The receiving device (slave) does not exist for the specified address.

ACK is output when the SDAn line of the receiving device changes to low level at the 9th clock (normal reception).

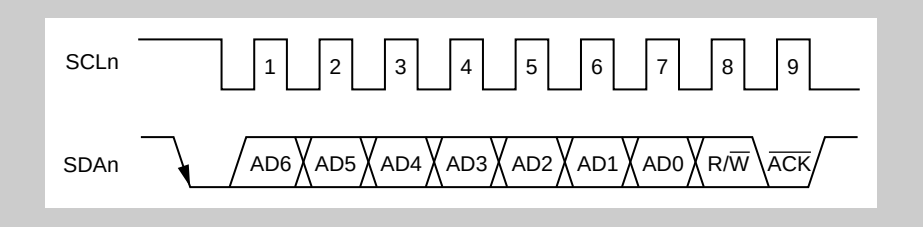



**Figure 20-8 Acknowledge (ACK)**

### 20.5.6 Data

Except for a 9-bit data string (consisting of a 7-bit address, 1-bit R/$\overline{W}$, and 1-bit $\overline{ACK}$) transferred after the start condition, the bits other than $\overline{ACK}$ are defined as data.

If a 10-bit address is specified using an extension code, the 8-bit data that is transferred after the address is used as the second address.

### 20.5.7 Stop condition

A stop condition is met if the SDAn signal level changes from low to high while the SCLn signal is high.
The stop condition is output when serial data transfer from the master device to the slave device has been completed.

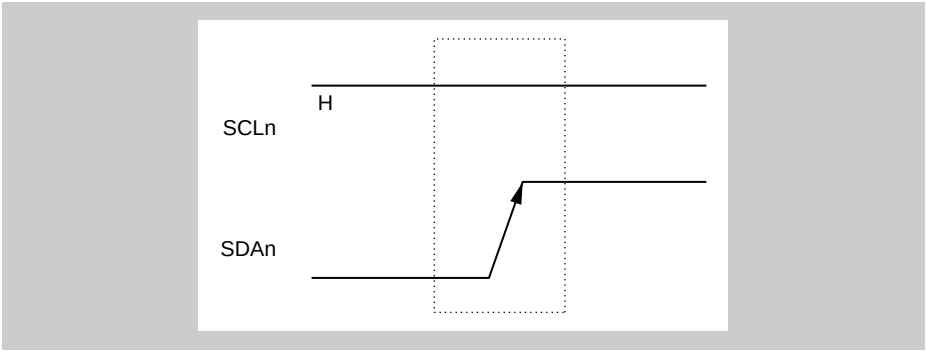



**Figure 20-9 Stop condition**

### 20.5.8 Wait state

A wait state is used to report to the communication destination that the IICBn (master or slave) is preparing to transmit or receive data.

The wait state is reported to the communication destination by setting the SCLn signal to low. The next data transfer cannot start until both the master and slave devices exit the wait state.

(a) When a wait at the 9th clock is specified for the master and a wait at the 8th clock for the slave (master: transmission, slave: reception)

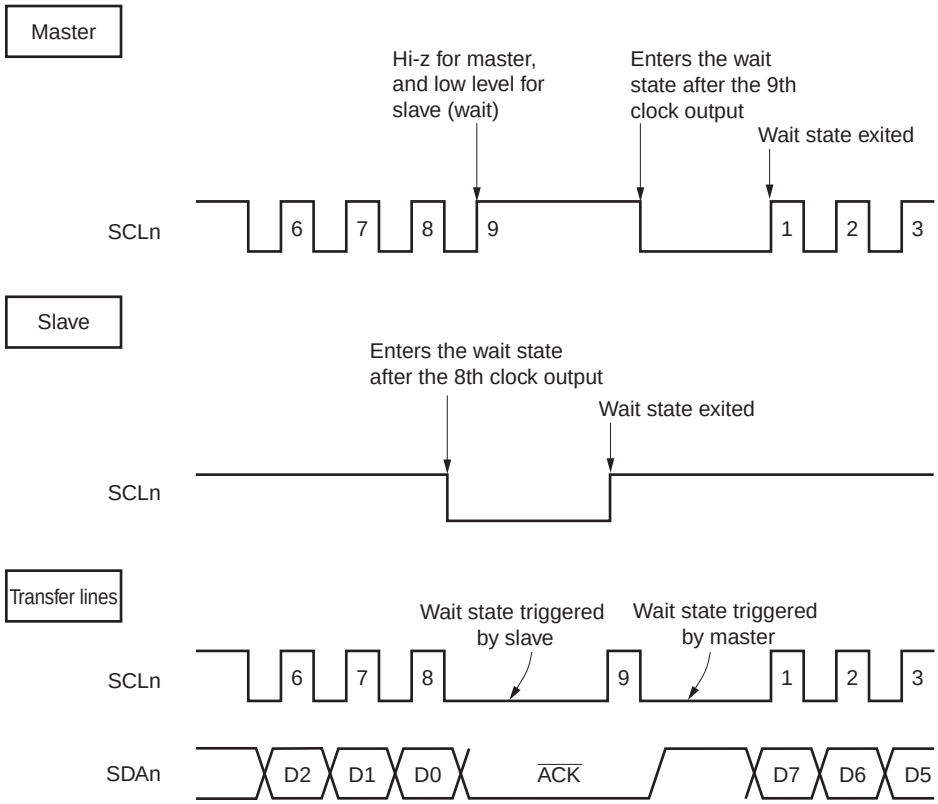



**Figure 20-10 Wait state (1/2)**

(b) When a wait at the 9th clock is specified for both the master and slave
(master: transmission, slave: reception)

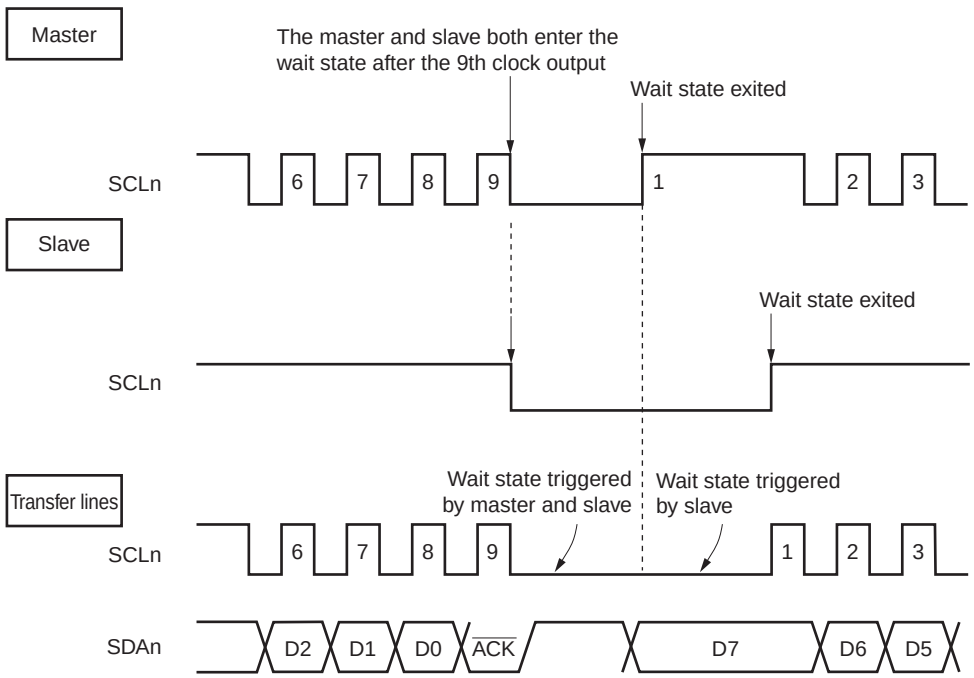



**Figure 20-10 Wait state (2/2)**

### 20.5.9 Arbitration

When several master devices simultaneously output a start condition, communication with the master devices continues until the data differs, while adjusting the clocks. An example where two masters simultaneously output a start condition and arbitration is conducted is described below.

This example assumes that one master outputs the SDAn line high (master 1) and the other master outputs the SDAn line low (master 2) while the SCLn line is low.
In this case, the communication with master 2 is prioritized, and communication is not authorized for master 1.

This kind of operation is called arbitration, and the state in which communication is not authorized is called arbitration loss. The master that lost arbitration releases the bus by setting both the SCLn and SDAn line to high impedance.

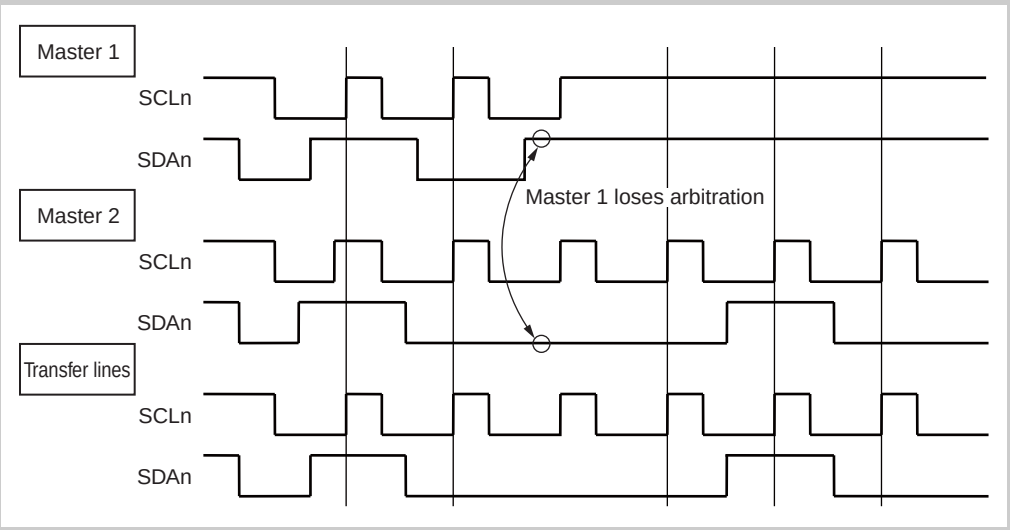



**Figure 20-11 Arbitration timing example**

## 20.6 Registers

**Caution** In this section, the operation when an extension code is received is omitted. For details about the extension code, refer to *20.7.4 "Extension code"*.

### (1) IICBnDAT – IICBn data register

This register is used to transmit and receive transfer data.

**Access** This register can be read/written in 8-bit units.

**Address** <IICBn_base> + $0000_H$

**Initial Value** $0000_H$. This register is also initialized by changing the value of the IICBnCTL0.IICBnIICE bit from 1 to 0 or from 0 to 1.

**Cautions**

1. When the IICBn becomes a master in the single transfer mode, after the IICBnTRG.IICBnSTT bit has been set to 1, writing to the IICBnDAT register is allowed only once to transfer the address and communication direction.
2. When transferring data in the single transfer mode, writing to the IICBnDAT register in communication state other than the wait state is prohibited.
3. When transferring data in the continuous transfer mode, writing to the IICBnDAT register in response to an INTIICBTIAn interrupt request signal is only allowed once.
4. When executing transmission operations in the continuous transfer mode, do not read the IICBnDAT register.
   Similarly, when performing reception operations in the continuous transfer mode, do not write to the IICBnDAT register.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IICBnDAT7 | IICBnDAT6 | IICBnDAT5 | IICBnDAT4 | IICBnDAT3 | IICBnDAT2 | IICBnDAT1 | IICBnDAT0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 20-9 IICBnDAT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | IICBnDAT[7:0] | During reception, these bits hold the received data.<br>During transmission, these bits write the transmit data.<br><br>The prescribed procedure must be followed during access (read, write) to the IICBnDAT register. For the setting sequence, refer to *20.10 "Setting Sequence"*. The IICBn exits the wait state by performing access to the IICBnDAT register.<br>• In single transfer mode<br>- When write access to the IICBnDAT register is performed<br>• In continuous transfer mode<br>- When write access to the IICBnDAT register is performed<br>- When read access to the IICBnDAT register is performed during a wait state for data transfer that is not triggered by NACK signal reception |

### (2) IICBnSVA – IICBn slave address register

This register stores the slave address of the I2C bus.

**Access** This register can be read/written in 8-bit units.

**Address** <IICBn_base> + $0004_H$

**Initial Value** $0000_H$

**Caution** Write access to the IICBnSVA register is prohibited when the value of the IICBnCTL0.IICBnIICE bit is 1.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IICBnSVA7 | IICBnSVA6 | IICBnSVA5 | IICBnSVA4 | IICBnSVA3 | IICBnSVA2 | IICBnSVA1 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R |

**Table 20-10 IICBnSVA register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 1 | IICBnSVA[7:1] | Store the slave address of the I2C bus.<br><br>Address match/address mismatch is judged by comparing the received address and the IICBnSVA register.<br>If the received address matches the IICBnSVA register, the IICBnSTR0.IICBnSSCO bit is set to 1. |

### (3) IICBnCTL0 – IICBn control register 0

This register is used to control the operations of the IICBn.

**Access** This register can be read/written in 8-bit units.

**Address** <IICBn_base> + $0008_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IICBn IICE | 0 | 0 | 0 | 0 | 0 | IICBn SLWT | IICBn SLAC |
| R/W | R | R | R | R | R | R/W | R/W |

**Table 20-11 IICBnCTL0 register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | IICBnIICE | Enables/disables operation of the IICBn.<br>0: Disables operation of IICBn.<br>1: Enables operation of IICBn.<br>Synchronous reset of the following registers is executed when the value of the IICBnIICE bit changes from 1 to 0, or the value of the IICBnIICE bit changes from 0 to 1.<br>• IICBnDAT and IICBnSTR0 registers<br>When IICBnIICE is 0, the SCLn and SDAn pins go into the high impedance state. |
| 1 | IICBnSLWT | Controls a wait and interrupt request output timing.<br>0: The IICBn enters the wait state and an interrupt request is output at the falling edge of the 8th clock during single transfer.<br>1: The IICBn enters the wait state and an interrupt request is output at the falling edge of the 9th clock during single transfer.<br><br>The IICBnSLWT bit controls wait state transition and interrupt request output at the following timing.<br>- 8th and 9th clocks during data transfer<br>For the conditions for transition to the wait state, refer to *20.7.3 "Entering and exiting wait state"*.<br>During address transfer, the conditions for transiting to the wait state and for interrupt request output are as follows, regardless of the setting of the IICBnSLWT bit.<br><br>• Master mode:<br>A data transmit/receive interrupt request signal (INTIICBTIAn) is output and the IICBn enters the wait state upon detection of the falling edge of the 9th clock.<br>• Slave mode:<br>During an address match, the INTIICBTIAn signal is output and the IICBn enters the wait state upon detection of the falling edge of the 9th clock. During address mis-match, the INTIICBTIAn signal is not output and the IICBn does not enter the wait state.<br><br>**Caution** During data transmission, rewriting the IICBnSLWT bit is prohibited. |

**Table 20-11 IICBnCTL0 register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | IICBnSLAC | Controls acknowledge signal output.<br>0: Disables acknowledge signal output.<br>Master:The acknowledge signal is not output during data reception (SDAn = “H”).<br>Slave:The acknowledge signal is not output during data transfer when an address match occurs (SDAn = “H”).<br>1: Enables acknowledge signal output.<br>Master:The acknowledge signal is output during data reception (SDAn = “L”).<br>Slave:The acknowledge signal is output during data transfer when an address match occurs (SDAn =“L”).<br><br>When the IICBn is operating as a slave, in the case of an address match, an acknowledge signal is output during address transfer regardless of the value of the IICBnSLAC bit (SDAn = “L”).<br>also, no acknowledge signal is output (SDAn = “H”) while the IICBn is transmitting data or when it does not participate in communications. |

### (4) IICBnCTL1 – IICBn control register 1

This register controls the operation of the IICBn.

**Access** This register can be read/written in 8-bit units.

**Address** <IICBn_base> + $0020_H$

**Initial Value** $0000_H$

**Caution** Write access to the IICBnCTL1 register is prohibited when the value of the IICBnCTL0.IICBnIICE is 1.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IICBn MDSC | IICBn LGDF2 | IICBn LGDF1 | IICBn LGDF0 | 0 | 0 | IICBn SLSE | 0 |
| R/W | R/W | R/W | R/W | R | R | R/W | R |

**Table 20-12 IICBnCTL1 register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | IICBnMDSC | Specifies the operation mode for the IICBn.<br>0: Standard mode (SCL clock frequency: 100 kbps max.)<br>1: Fast mode (SCL clock frequency: 400 kbps max.) |
| 6 to 4 | IICBnLGDF[2:0] | Specify the digital filter sampling frequency.<br>Note that the digital filter can be used only in the fast mode.<br>000:Does not use digital filter.<br>SCLn and SDAn are used without passing through the digital filter in the IICBn.<br>The digital filter circuit operations are stopped.<br>Other than above:Uses digital filter.<br>SCLn and SDAn are used passing through the digital filter in the IICBn.<br><br>When using a digital filter, set bits IICBnLGDF2 to IICBnLGDF0 as follows. |

| IICBnLGDF2 to IICBnLGDF0 bits | Frequency |
|---|---|
| 001 | Minimum frequency [a] ≤ PCLK ≤ 20 MHz |
| 010 | 20 MHz < PCLK ≤ 40 MHz |
| 011 | 40 MHz < PCLK ≤ 60 MHz |
| 100 | 60 MHz < PCLK ≤ 80 MHz |
| 101 | 80 MHz < PCLK ≤ 100 MHz |
| 110, 111 | Setting prohibited |

a) A list of the minimum frequencies by setting is shown below.

| Operation mode (IICBnMDSC) | When digital filter used (IICBnLGDF bits = 000) | When digital filter used (IICBnLGDF bits ≠ 000) |
|---|---|---|
| Standard mode (0) | 1.0 MHz | Use prohibited |
| Fast mode (1) | 3.5 MHz | 4.0 MHz |

**Table 20-12 IICBnCTL1 register contents (2/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | IICBnSLSE | Enables/disables start condition output in the initial communication state.<br>0: Disables start condition output in the initial communication state.<br>1: Enables start condition output in the initial communication state.<br><br>If the IICBnSLSE bit is set to 1, a start condition can be output by setting the IICBnTRG.IICBnSTT bit to 1 in the initial communication state (from when the IICBnCTL0.IICBnIICE bit is set to 1 until detection of a stop condition).<br>The IICBnSLSE bit is automatically cleared to 0 upon detection of a start condition (even without a 0 write operation).<br><br>**Caution** Clear the IICBnSLSE bit to 0 when participating in communications after other communications have started.<br>When other communications are being performed, if the IICBnSTT has been set to 1 with the IICBnSLSE bit set to 1, the other communications may be damaged. |

### (5) IICBnWL – IICBn low level width setting register

This register is used to set the low level width of the serial clock register (SCLn).

**Access** This register can be read/written in 16-bit units.

**Address** <IICBn_base> + $0024_H$

**Initial Value** $03FF_H$

**Caution** Write access to the IICBnWL register is prohibited when the value of the IICBnCTL0.IICBnIICE bit is 1.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | IICBnWL9 | IICBnWL8 |
| R | R | R | R | R | R | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IICBnWL7 | IICBnWL6 | IICBnWL5 | IICBnWL4 | IICBnWL3 | IICBnWL2 | IICBnWL1 | IICBnWL0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 20-13 IICBnWL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 9 to 0 | IICBnWL[9:0] | Specify the $t_{LOW}$ period (low level width of the SCLn clock) of the $I^2C$ bus specification. The value of the IICBnWL register is used to determine the serial output timing of other $I^2C$ bus specifications. For the serial output timing setting conditions, refer to *a "Setting transfer clock by using IICBnWL and IICBnWH registers" on page 1091* . |

**(a) Setting transfer clock by using IICBnWL and IICBnWH registers**

The various timings in compliance with the I²C bus specifications can be set by setting the IICBnWL register and IICBnWH register.

- Setting transfer clock on master side

$$\text{Transfer clock (Hz)} = \frac{\text{PCLK}}{\text{IICBnWL} + \text{IICBnWH} + \text{PCLK}\ (t_R + t_F)}$$

At this time, the optimal setting values of IICBnWL and IICBnWH are as follows.
(The fractional parts of all setting values are rounded up.)

– When the fast mode

$$\text{IICBnWL} = \frac{0.52}{\text{Transfer clock}} \times \text{PCLK}$$

$$\text{IICBnWH} = \left(\frac{0.48}{\text{Transfer clock}} - t_R - t_F\right) \times \text{PCLK}$$

– When the normal mode

$$\text{IICBnWL} = \frac{0.47}{\text{Transfer clock}} \times \text{PCLK}$$

$$\text{IICBnWH} = \left(\frac{0.53}{\text{Transfer clock}} - t_R - t_F\right) \times \text{PCLK}$$

**Caution** The data hold time must be 0.9 µs or less in the fast mode and 3.45 µs or less in the standard mode.

**Note** The data hold time is determined by the IICBWL register setting as follows:

Data hold time = IICBnWL.IICBnWL[9:2] / PCLK

- Setting IICBnWL and IICBnWH on slave side
  (The fractional parts of all setting values are rounded up.)

  – When the fast mode

    IICBnWL = 1.3 µs × PCLK

    IICBnWH = (1.2 µs – $t_R$ – $t_F$) × PCLK

  – When the normal mode

    IICBnWL = 4.7 µs × PCLK

    IICBnWH = (5.3 µs – $t_R$ – $t_F$) × PCLK

**Note** IICBnWL: IICBn low-level width setting register
IICBnWH: IICBn high-level width setting register
$t_F$: SDAn and SCLn signal falling times
$t_R$: SDAn and SCLn signal rising times
PCLK: Frequency of the clock supplied to the IICBn
$f_{CLK}$: SCL clock frequency

### (6) IICBnWH – IICBn high-level width setting register

This register is used to set the high level width of the serial clock signal (SCLn).

**Access** This register can be read/written in 16-bit units.

**Address** <IICBn_base> + $0028_H$

**Initial Value** $03FF_H$

**Caution** Write access to the IICBnWH register is prohibited when the value of the IICBnCTL0.IICBnIICE bit is 1.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | IICBnWH9 | IICBnWH8 |
| R | R | R | R | R | R | R/W | R/W |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IICBnWH7 | IICBnWH6 | IICBnWH5 | IICBnWH4 | IICBnWH3 | IICBnWH2 | IICBnWH1 | IICBnWH0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 20-14 IICBnWH register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 9 to 0 | IICBnWH[9:0] | Specify the $t_{HIGH}$ period (high level width of the SCLn clock) of the $I^2C$ bus specification. The value of the IICBnWH register is used to determine the serial output timing of other $I^2C$ bus specifications. For the serial output timing setting conditions, refer to *a "Setting transfer clock by using IICBnWL and IICBnWH registers" on page 1091* . |

### (7) IICBnTRG – IICBn trigger register

This register is used to set the IICBn trigger.

**Access** This register can be read/written in 8-bit units.

**Address** <IICBn_base> + $000C_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | IICBn LRET | IICBn WRET | IICBn STT | IICBn SPT |
| R | R | R | R | R/W | R/W | R/W | R/W |

**Table 20-15 IICBnTRG register contents (1/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 3 | IICBnLRET | Communication exit trigger bit<br>0: Normal operation<br>1: The IICBn exits the current communication and enters the wait state. This bit is automatically cleared to 0 following execution.<br><br>The following occurs when IICBnLRET is 1.<br>- SCLn and SDAn each go into high impedance (communication wait state).<br>- Bits IICBnSSMS, IICBnSSDR, IICBnSSWT, IICBnSSEX, IICBnSSC0, IICBnSSTR, IICBnSSAC, and IICBnSSST of the IICBnSTR0 register are cleared to 0.<br>- When IICBnSTT = 1 (start condition output preparation) or IICBnSPT =1 (stop condition output preparation) has been set, output of a start condition or stop condition is stopped.<br><br>**Caution** If IICBnLRET is set to 1 during master operation (IICBnSTR0.IICBnSSMS = 1), the bus is released. Since serial clock output stops, problems occur during communication on the slave side. |
| 2 | IICBnWRET | This is the trigger bit for exiting the wait state.<br>0: Does not exit the wait state.<br>1: Exits the wait state and resumes communication. This bit is automatically cleared following execution.<br><br>If the IICBn have exited the wait state by setting the IICBnWRET bit to 1 during the wait state triggered by the falling edge of the 9th clock, the IICBnSTR0.IICBnSSTR bit is cleared to 0 and SDAn goes into high impedance (this enables the external master to output a stop condition or start condition.)<br>If the IICBn is not in the wait state (IICBnSTR0.IICBnSSWT = 0), setting this bit to 1 has no meaning.<br>There are other conditions for exiting the wait state in addition to the setting of this bit. For details, refer to *20.7.3 "Entering and exiting wait state"*. |

**Table 20-15 IICBnTRG register contents (2/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | IICBnSTT | Start condition trigger bit<br>0: Does not output a start condition.<br>1: Outputs a start condition (This bit is automatically cleared to 0 after it has been set to 1.<br><br>The IICBnSTT bit can be set to 1 under the following conditions:<br>[1] IICBnSTR0.IICBnSSMS bit = Master state (1)<br>• During wait state triggered by the falling edge of the 9th clock (both address transfer and data transfer)<br>• During data reception, only after clearing the ACKEn bit to 0 to report the end of reception to the slave<br><br>After exiting the wait state that was triggered by the falling edge of the 9th clock, and upon detection of the falling edge of the 9th clock in all other cases, SDAn and SCLn are set to high level after the $t_{LOW}$ (low-level width of SCLn clock) period, and when SDAn is set to the low level after waiting for $t_{SU:STA}$ of the $I^2C$ bus specification (setup time of start/restart condition), a start condition is output.<br>Then, SCLn is set to low level after the $t_{HD:STA}$ time (hold time) of the $I^2C$ bus specification has elapsed.<br><br>[2] Slave state or communication wait state (IICBnSTR0.IICBnSSMS = 0)<br>• IICBnSTR0.IICBnSSBS bit = 0 (bus release state)<br>Outputs a start condition.<br>Following verification of the $t_{BUF}$ (bus free time (between stop/start conditions) of the $I^2C$ specification (if this time has not elapsed, following lapse, and if this time has elapsed, immediately), a start condition is output when SDAn is changed from high level to low level while SCLn is high level. (At this time, SCLn outputs a high level.)<br>Then, SCLn is set to low level after the $t_{HD: STA}$ time of the $I^2C$ bus specification has elapsed.<br>• IICBnSSBS bit = 1 (bus communication state)<br>This status indicates that communication is performed on the bus while the IICBn is not operating as a master.<br>The IICBnSTR0.IICBnSTCF bit is set to 1 and a start condition is not output.<br><br>**Caution** [2] shows the operations according to the value of the IICBnSSBS bit when the IICBnSTT bit is 0.<br>Even if the IICBnSTT bit is set to 1 after checking the value of the IICBnSSBS bit through register read, the value of IICBnSSBS may differ from its value when it was checked. |

**Table 20-15 IICBnTRG register contents (3/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | IICBnSTT | The output processing of the start condition is started by setting the IICBnSTT bit to 1, but upon detection of the following states, output processing of the start condition is stopped and the start condition is not output.<br>- When 0 is written to the IICBnCTL0.IICBnIICE bit<br>- When 1 is written to the IICBnLRET bit<br>- Upon detection of arbitration loss<br><br>**Caution** When start in the initial communication state is enabled (IICBnCTL1.IICB0SLSE bit = 1), the start condition is output regardless of the bus status when the IICBnSTT bit is set to 1. If other communications are performed at that time, they may be damaged.<br><br>**Note** Setting the IICBnSTT bit at the same time as the IICBnSPT bit is prohibited. |
| 0 | IICBnSPT | Stop condition trigger bit<br>0: Does not output a stop condition.<br>1: Outputs a stop condition (This bit is automatically cleared after it has been set to 1).<br><br>The IICBnSPT bit can be set to 1 under the following conditions while the IICBn is performing communication as a master.<br>• Wait state triggered by the falling edge of the 9th clock (both address transfer and data transfer)<br>• During data reception, only after clearing the ACKEn bit to 0 to report the end of reception to the slave<br><br>A stop condition can be output with the following procedure.<br>(If the IICBn is in the wait state, after exiting the wait state) SCLn is released when SDAn has output a low level, and SCLn = high level, SDAn is low level are waited for. Then, following the lapse of the $t_{SU:STO}$ time, a stop condition is output by setting SDAn to high level.<br><br>The output processing of the stop condition is started by setting the IICBnSPT bit to 1, but upon detection of the following states, output processing of the stop condition is stopped and the stop condition is not output.<br>- When 0 is written to the IICBnIICE bit<br>- When 1 is written to the IICBnLRET bit<br>- Upon detection of a stop condition<br>- Upon detection of arbitration loss<br><br>**Caution** Setting the IICBnSPT bit to 1 is prohibited during slave operation (IICBnSTR0.IICBnSSMS bit = 0)<br><br>**Note** Setting the IICBnSPT bit to 1 at the same time as the IICBnSTT bit is prohibited. |

### (8) IICBnSTR0 – IICBn status register 0

This register indicates the statuses of the IICBn and the bus.

**Access** This register can be read only in 16-bit units. However, when IICBnIICE is 0, this register can also be write accessed.

**Address** <IICBn_base> + $0010_H$

**Initial Value** $0000_H$. This register is also initialized by changing the value of the IICBnCTL0.IICBnIICE bit from 1 to 0 or from 0 to 1.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| IICBn SSMS | 0 | IICBn SSDR | IICBn SSWT | IICBn SSEX | IICBn SSCO | IICBn SSTR | IICBn SSAC |
| R | R | R | R | R | R | R | R |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0[a] | IICBn SSBS | IICBn SSST | IICBn SSSP | 0 | 0 | IICBn STCF | IICBn ALDF |
| R | R | R | R | R | R | R | R |

a) Bit 7 may change to 1, but has no particular meaning.

**Table 20-16 IICBnSTR0 register contents (1/5)**

| Bit position | Bit name | Function |
|---|---|---|
| 15 | IICBnSSMS | Master state check flag: Indicates that the IICBn is operating as a master.<br>1: Indicates that the IICBn is operating as a master.<br><br>Setting condition: Upon detection of a start condition after 1 is written to the IICBnTRG.IICBnSTT bit<br>Clearing conditions: • When 1 is written to the IICBnTRG.IICBnLRET bit<br>• Upon detection of a stop condition<br>• Upon detection of arbitration loss<br>If a setting condition coincides with a clearing condition, the clearing condition takes priority. |
| 13 | IICBnSSDR | IICBnDAT register status flag<br>1: Indicates that data in the IICBnDAT register remains unprocessed.<br>During reception operation:Received data remains unread in the IICBnDAT register.<br>During transmission operation:Data written to the IICBnDAT register has not been transferred to the IICBnDAT register.<br><br>Setting condition:<br>• When the IICBnDAT register is written during address transfer and data transfer while the IICBnSSWT bit is 0<br>(Note that the IICBnSSDR bit is not set to 1 if address data is written to the IICBnDAT register while the IICBn is operating as a master, because the address data is directly transferred to the IICBnDAT register in this event.)<br>• At the falling edge of the 9th clock after an address match with a slave<br>• When IICBnCTL0.IICBnSLWT = 0, at the falling edge of the 8th clock during data reception<br>• When IICBnCTL0.IICBnSLWT = 1, at the falling edge of the 9th clock during data reception |

Table 20-16 IICBnSTR0 register contents (2/5)

| Bit position | Bit name | Function |
|---|---|---|
| 13 | IICBnSSDR | Clearing conditions:<br>• Clearing conditions given priority over setting conditions<br>- When 1 is written to the IICBnLRET bit<br>- Upon detection of arbitration loss<br>- At the falling edge of the 9th clock during address transfer while the IICBn is operating as a master<br>- When IICBnCTL0.IICBnSLWT = 0, at the falling edge of the 8th clock during data transmission<br>- When IICBnCTL0.IICBnSLWT = 1, at the falling edge of the 9th clock during data transmission<br>• Clearing condition for which setting conditions are given priority<br>- When the IICBnDAT register is read while the IICBnDAT register does not have any received data that must be transferred to the IICBnDAT register |
| 12 | IICBnSSWT | Wait state flag<br>1: Indicates that the IICBn is in the wait state.<br><br>Setting condition:<br><Common to master/slave><br>• During data transfer, upon detection of the falling edge of the 8th clock with IICBnSLWT = 0<br>• During data transfer, upon detection of the falling edge of the 9th clock with IICBnSLWT = 1<br><Master><br>• When the IICBn becomes a master (IICBnSSMS = 1) after 1 is written to the IICBnSTT bit, and the falling edge of the first SCLn is detected without the IICBnDAT register being written<br>• Upon detection of the falling edge of the 9th clock during address transfer<br><Slave><br>• Upon detection of the falling edge of the 9th clock during address transfer when an address match occurred<br><br>During address transfer period, operating as master><br>- When the IICBn becomes a master (IICBnSSMS = 1) after 1 is written to the IICBnSTT bit, and the first falling edge is detected without the IICBnDAT register being written<br>- Upon NACK detection<br>(However, only if 1 has not been written to IICBnTRG.IICBnSTT or IICBnTRG.IICBnSPT)<br><During address transfer period, operating as slave><br>- Upon detection of the falling edge of the 9th clock while IICBnSSTR bit is 0 during address transfer when an address match occurred<br>- Upon NACK detection |

**Table 20-16 IICBnSTR0 register contents (3/5)**

| Bit position | Bit name | Function |
|---|---|---|
| 12 | IICBnSSWT | Clearing conditions:<br>• Clearing conditions given priority over setting conditions<br>- When 1 is written to the IICBnLRET bit<br>- When 1 is written to the IICBnSPT bit while the IICBn is operating as a master in the continuous transfer mode<br>- When the IICBnDAT register is written while the IICBn is performing transmission in the continuous transfer mode<br>- During the wait state triggered by the falling edge of the 8th clock, when the IICBnDAT register is read while reception is performed in the continuous transfer mode<br>- During the wait state triggered by the falling edge of the 9th clock, when the IICBnDAT register is read while the IICBn is performing reception in the continuous transfer mode and an acknowledge signal ($\overline{ACK}$) has been received<br>• Clearing condition for which setting conditions are given priority<br>- When 1 is written to the IICBnTRG.IICBnWRET bit<br>- When 1 is written to the IICBnSTT bit while the IICBn is operating as a master in the single transfer mode<br>- When 1 is written to the IICBnSPT bit while the IICBn is operating as a master in the single transfer mode<br>- When the IICBnDAT register is written while the IICBn is performing reception in the single transfer mode<br><br>**Caution** If the IICBn exits the wait state that was triggered by the falling edge of the 9th clock by writing 1 to the IICBnWRET bit, the IICBnSSTR bit is cleared to 0 and the bus is released (both SCLn and SDAn go into high impedance). |
| 11 | IICBnSSEX | Expansion code reception detection flag<br>1: Indicates that an expansion code has been received.<br><br>Setting condition: Upon detection of the falling edge of the 8th clock while transferring received address data whose higher 4 bits are either 0000 or 1111<br>Clearing conditions: • When 1 is written to the IICBnLRET bit<br>• Upon detection of a stop condition<br>• Upon detection of a start condition<br><br>**Caution** When the expansion codes match, the processing after the interrupt differs according to the ensuing data, and therefore is dependent on software processing. |
| 10 | IICBnSSCO | Address match detection flag<br>1: Indicates that an address that matches the IICBnSVA register has been detected.<br><br>Setting condition: Upon detection of the falling edge of the 8th clock while transferring a received address that matches the IICBnSVA register<br>Clearing conditions: • When 1 is written to the IICBnLRET bit<br>• Upon detection of a stop condition<br>• Upon detection of a start condition |

**Table 20-16 IICBnSTR0 register contents (4/5)**

| Bit position | Bit name | Function |
|---|---|---|
| 9 | IICBnSSTR | Transmission status detection flag<br>1: Indicates that data is being transmitted to the serial data bus.<br><br>Setting condition:<br><Master><br>- Upon detection of a start condition after 1 is written to the IICBnSTT bit<br><Slave><br>- Upon detection of the falling edge of the 8th clock following reception of 1 to R/$\overline{W}$ bit during address transfer when an address match occurred<br>Clearing conditions:<br><Common to master/slave><br>- When 1 is written to the IICBnLRET bit<br>- Upon detection of a stop condition<br>- When 1 is written to the IICBnWRET bit during the wait state triggered by the falling edge of the 9th clock<br><Master><br>- Upon detection of the falling edge of the 8th clock following reception of 1 to R/$\overline{W}$ bit during address transfer<br>- Upon detection of arbitration loss<br><Slave><br>- Upon detection of a start condition |
| 8 | IICBnSSAC | Acknowledge ($\overline{ACK}$) detection flag<br>1:Indicates that an acknowledge signal has been detected.<br><br>Setting condition: Upon detection of the falling edge of SCLn when a low level has been received at the $\overline{ACK}$ bit during participation in communications<br>Clearing conditions: • When 1 is written to the IICBnLRET bit<br>• Upon detection of the rising edge of SCLn<br><br>**Caution** The value of the IICBnSSAC bit changes regardless of whether or not an interrupt has occurred. |
| 6 | IICBnSSBS | IICBn bus status flag<br>0: Bus released state (initial communication state when IICBnCTL1.IICBnSLSE = 1)<br>1: Bus communication state (initial communication state when IICBnCTL1.IICBnSLSE = 0)<br><br>Setting condition: • Upon detection of a start condition<br>• When 1 is written to the IICBnCTL0.IICBnIICE bit when IICBnSLSE = 0<br>Clearing conditions: Upon detection of a stop condition<br><br>**Note** The IICBnSSBS bit operates whether or not the IICBn is participating in communications. |
| 5 | IICBnSSST | Start condition detection flag<br>1: Indicates that a start condition has been detected.<br><br>Setting condition: Upon detection of a start condition<br>Clearing conditions: • When 1 is written to the IICBnLRET bit<br>• Upon detection of a stop condition<br>• Upon detection of the rising edge of SCLn following the end of address transfer<br><br>**Note** The IICBnSSST bit operates whether or not the IICBn is participating in communications. |

**Table 20-16 IICBnSTR0 register contents (5/5)**

| Bit position | Bit name | Function |
|---|---|---|
| 4 | IICBnSSSP | Stop condition detection flag<br>1: Indicates that a stop condition has been detected.<br><br>Setting condition: Upon detection of a stop condition<br>Clearing conditions: Upon detection of the falling edge of the first SCLn following start condition detection<br><br>**Note** The IICBnSSSP bit operates whether or not the IICBn is participating in communications. |
| 1 | IICBnSTCF | IICBnSTT bit clear flag<br>1: Indicates that the IICBnSTT bit has been cleared because start condition output failed.<br><br>Setting condition: When 1 is written to the IICBnSTT bit during bus communication when the IICBn is not operating as a master<br><br>**Caution** Even if the bus is released in the external bus state, this bit is set to 1 when 1 is written to the IICBnSTT bit, unless the IICBn recognizes the bus release state (IICBnSSBS = 1).<br><br>Clearing condition: When 1 is written to the IICBnSTRC.IICBnCLSF bit |
| 0 | IICBnALDF | Arbitration loss detection flag<br>1: Indicates that an arbitration loss has been detected.<br><br>Setting condition: Upon detection of arbitration loss<br>Clearing condition: When 1 is written to the IICBnSTRC.IICBnCLAF bit<br><br>If a setting condition coincides with a clearing condition, the setting condition takes priority.<br>Upon detection of arbitration loss, the IICBnSSMS and IICBnSSTR bits are cleared to 0. (SCLn and SDAn become high level and the bus is released.)<br><br>**Caution** When the IICBnALDF bit is set to 1 due to arbitration loss, the INTIICBTIA interrupt request signal may be output.<br>After confirming that the IICBnALDF bit has been set to 1 with an interrupt request signal, clear the IICBnALDF bit with the IICBnSTRC.IICBnCLAF bit. |

### (9) IICBnSTR1 – IICBn status register 1

This register indicates the status of the serial bus.

**Access** This register is read-only, in 8-bit units.

**Address** <IICBn_base> + $0014_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | IICBn SSCL | IICBn SSDA |
| R | R | R | R | R | R | R | R |

**Table 20-17 IICBnSTR1 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | IICBnSSCL | Indicates the level of the SCLn pin (input).<br>0: Low level<br>1: High level |
| 0 | IICBnSSDA | Indicates the level of the SDAn pin (input).<br>0: Low level<br>1: High level |

### (10) IICBnSTRC – IICBn status clear register

This register clears the IICBnSTCF and IICBnALDF bits of the IICBnSTR0 register.

**Access** This register can be read/written in 8-bit units.

**Address** <IICBn_base> + $0018_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | IICBnCLSF | IICBnCLAF |
| R | R | R | R | R | R | R/W | R/W |

**Table 20-18 IICBnSTRC register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 1 | IICBnCLSF | Clears the IICBnSTR0.IICBnSTCF bit.<br>1: Clears the IICBnSTCF bit.<br><br>**Note** If the IICBnCLSF bit is read after setting data, 0 is returned. |
| 0 | IICBnCLAF | Clears the IICBnSTR0.IICBnALDF bit.<br>1: Clears the IICBnALDF bit.<br><br>**Caution** If writing 1 to the IICBnCLAF bit and the setting condition of the IICBnALDF bit occur at the same time, the setting condition of the IICBnALDF takes priority.<br><br>**Note** If the IICBnCLAF bit is read after data setting, 0 is returned. |

### (11) IICBnEMU - IICBn emulation register

This register controls whether the IICBn can be stopped during emulation, for instance upon a breakpoint hit.

**Access** This register can be read/written in 8-bit units.

**Address** <IICBn_base> + $34_H$

**Initial Value** $0000_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IICBn SVSDIS | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R | R | R | R | R | R | R |

**Table 20-19 IICBnEMU register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | IICBn SVSDIS | Emulation control<br>0: IICBn can be stopped during emulation<br>1: IICBn continuous operating during emulation |

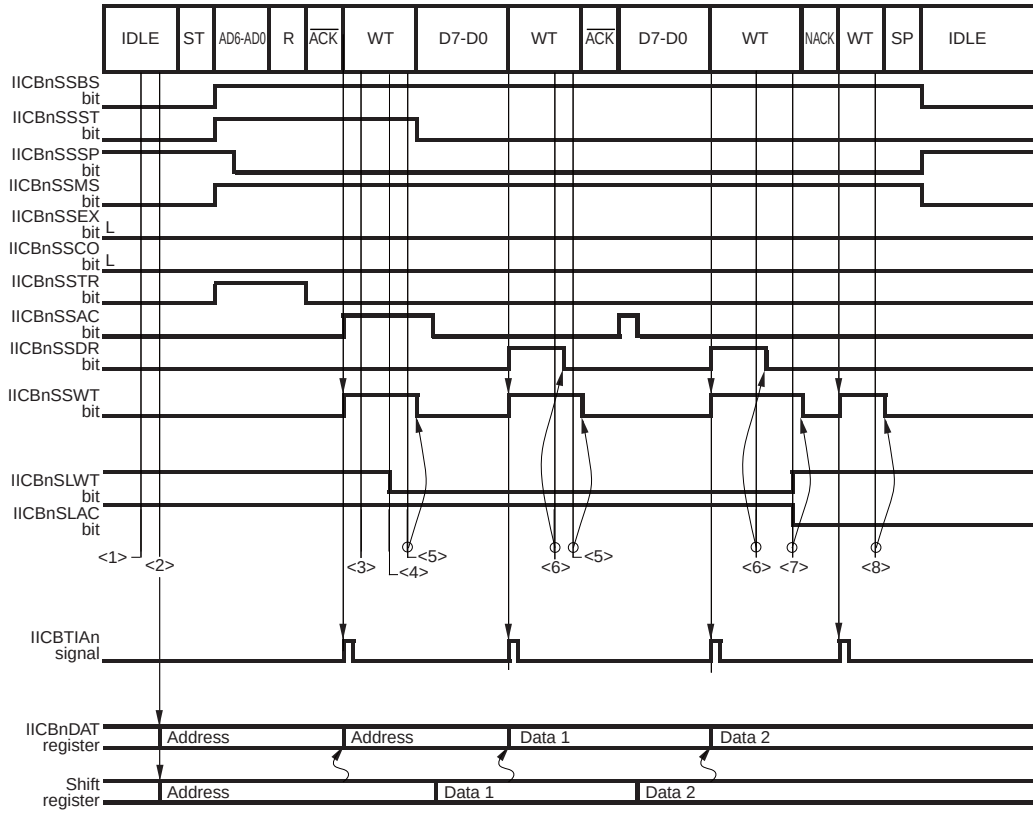
## 20.7 Operation

### 20.7.1 Single transfer mode

In the single transfer mode, a data transmit/receive interrupt request signal is output at the timing specified using the IICBnCTL0.IICBnSLW bit to make the IICBn enter the wait state, and transmit/receive data processing is performed during this wait state.

The various processing operations are described below.

### (1) Example of communication in single transfer mode (master reception)




<1> Start condition output
Set the IICBnTRG.IICBnSTT bit (to 1).

<2> Address and transfer direction specification output
Set the address of the slave device and the transfer direction as 8 bits into the IICBnDAT register.

<3> Acknowledge result check
Check the acknowledge result by reading the IICBnSTR0.IICBnSSAC bit using the IICBTIAn interrupt.

<4> Wait timing setting
During data reception, clear the IICBnCTL0.IICBnSLWT bit (to 0) so that the IICBn enters the wait state at the falling edge of the 8th clock.

<5> Data reception
Exit the wait state by setting the IICBnTRG.IICBnWRET bit (to 1) to start reception.

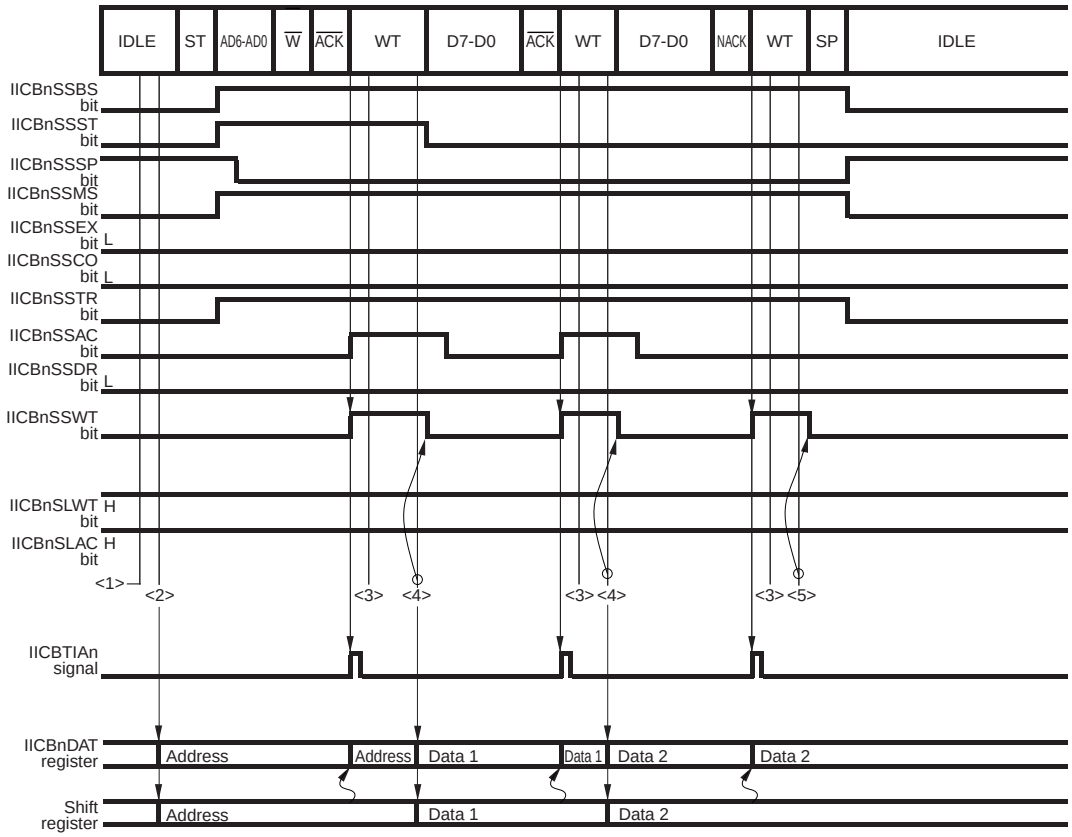
<6> Receive data load
Read the receive data from the IICBnDAT register using the IICBTIAn interrupt.

<7> Data reception completion processing

- Set the IICBnCTL0.IICBnSLWT bit to 1 and the IICBnSLAC bit to 0.
- Next, exit the wait state by setting the IICBnTRG.IICBnWRET bit (to 1). The end of the data is notified to the transmitting device without outputting ACK.

<8> Stop condition output
Set the IICBnTRG.IICBnSPT bit (to 1).

### (2) Example of communication in single transfer mode (master transmission)




**Note** During data transmission, set the IICBnCTL0.IICBnSLWT bit (to 1) so that the IICBn enters the wait state at the falling edge of the 9th clock.

<1> Start condition output
Set the IICBnTRG.IICBnSTT bit (to 1).

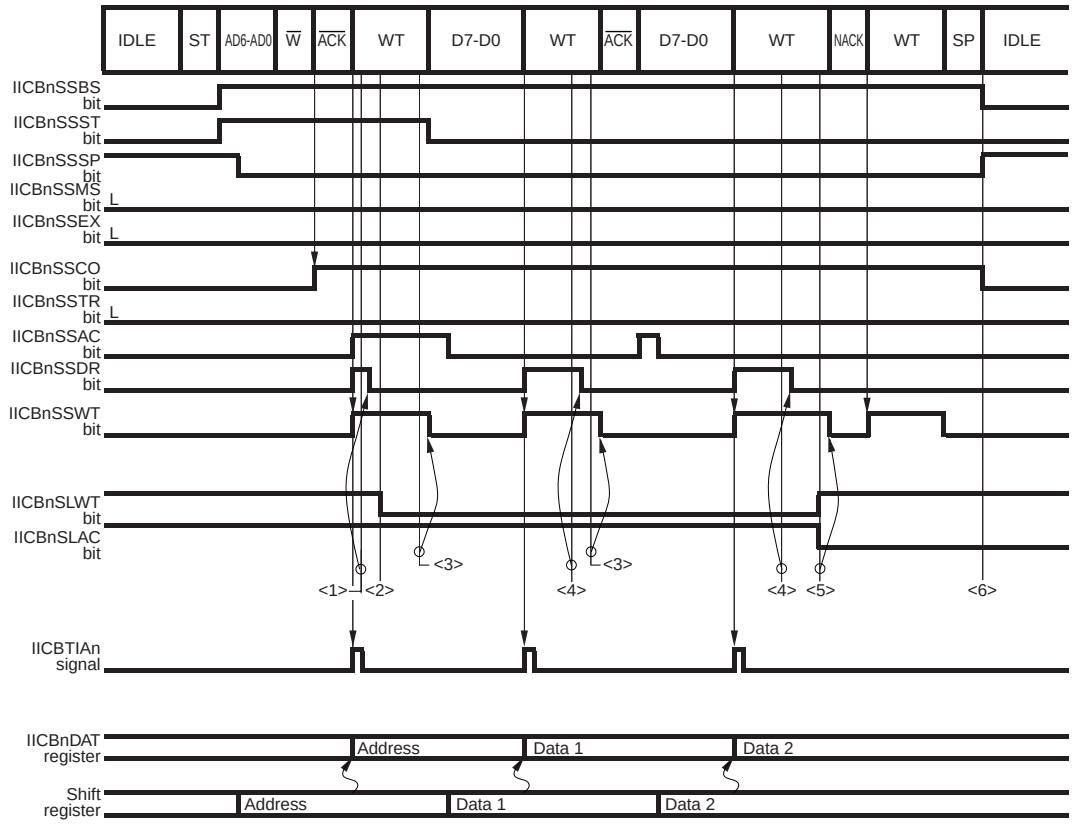
<2> Address and transfer direction specification output
Set the address of the slave device and the transfer direction as 8 bits into the IICBnDAT register.

<3> Acknowledge result check
Check the acknowledge result by reading the IICBnSTR0.IICBnSSAC bit using the IICBTIAn interrupt.

<4> Data transmission
Exit the wait state by setting the transmit data into the IICBnDAT register to start transmission.

<5> Stop condition output
Set the IICBnTRG.IICBnSPT bit (to 1).

### (3) Example of communication in single transfer mode (slave reception)




<1> Operation mode check in slave mode

- Check the operation mode using the IICBTIAn interrupt.
- Check the address transfer, address match, and reception operation with the IICBnSTR0.IICBnSSST, IICBnSSCO, and IICBnSSTR bits.
- Read the IICBnDAT register (empty read).

<2> Wait timing setting
During data reception, clear the IICBnCTL0.IICBnSLWT bit (to 0) so that the IICBn enters the wait state at the falling edge of the 8th clock.

<3> Data reception
Exit the wait state by setting the IICBnTRG.IICBnWRET bit (to 1) to start reception.

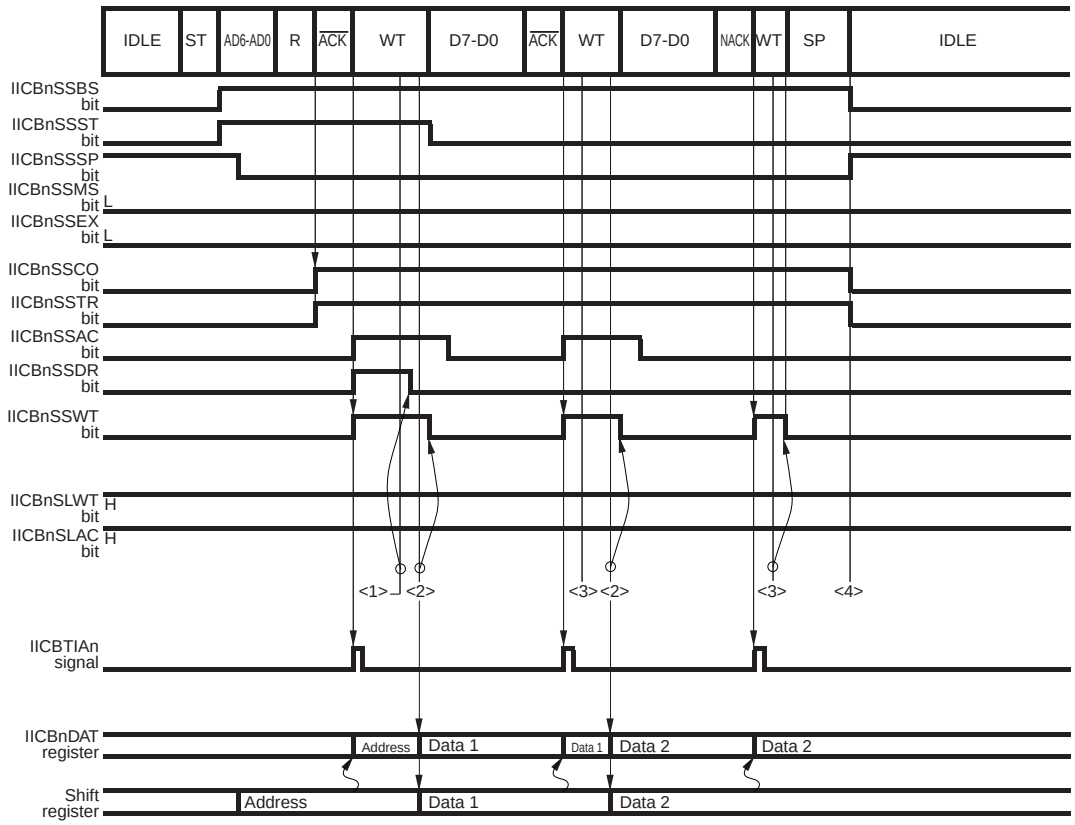
<4> Receive data load
Read the receive data from the IICBnDAT register using the IICBTIAn interrupt.

<5> Data reception completion processing

- Set the IICBnCTL0.IICBnSLWT bit to 1 and the IICBnSLAC bit to 0.
- Next, exit the wait state by setting the IICBnTRG.IICBnWRET bit (to 1). The end of the data is notified to the transmitting device without outputting $\overline{\text{ACK}}$.

<6> Stop condition detection.

### (4) Example of communication in single transfer mode (slave transmission)




**Note** During data transmission, set the IICBnCTL0.IICBnSLWT bit (to 1) so that the IICBn enters the wait state at the falling edge of the 9th clock.

<1> Operation mode check in slave mode

- Check the operation mode using the IICBTIAn interrupt.
- Check the address transfer, address match, and reception operation with the IICBnSTR0.IICBnSSST, IICBnSSCO, and IICBnSSTR bits.
- Read the IICBnDAT register (empty read).

<2> Data transmission
Exit the wait state by setting the transmit data into the IICBnDAT register to start transmission.

<3> Acknowledge result check
Check the acknowledge result by reading the IICBnSTR0.IICBnSSAC bit using the IICBTIAn interrupt.
If $\overline{\text{ACK}}$ is not output, the transmission is judged to have been completed, and the IICBn exits the wait state by setting the IICBnTRG.IICBnWRET bit (to 1).

<4> Stop condition detection.

### 20.7.2 Arbitration

When the IICBn operates as the master device and loses arbitration, it enters the slave standby state by setting both SCLn and SDAn to high level upon detection of the arbitration loss, and then the IICBnSTR0.IICBnALDF bit is set (to 1).

**(1) Status upon occurrence of arbitration**

The statuses upon occurrence of arbitration during master device operation (IICBnSTR0.IICBnSSMS bit = 1) are listed below.

[8] Address transmission

[9] R/$\overline{W}$ bit transmission of address transfer

[10] Extension code transmission

[11] R/$\overline{W}$ bit transmission of extension code transfer

[12] Data transmission

[13] $\overline{ACK}$ bit transmission after data reception

[14] Start condition detection during address transfer or data transfer

[15] Stop condition detection during address transfer or data transfer

[16] The SDAn signal is low when the IICBn is attempting to output a restart condition

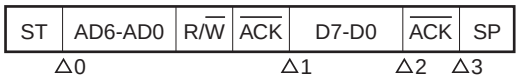
[17] The SDAn signal is low when the IICBn is attempting to output a stop condition

[18] The falling edge of the SCLn signal is detected when the IICBn is attempting to output a restart condition

### 20.7.3 Entering and exiting wait state

The IICBn enters the wait state at the following timings.

**Table 20-20 Wait state transit timings**




| Timing | Description | Refer to: |
|---|---|---|
| Δ0 | Upon detection of the first falling edge of the SCLn, following detection of start condition as the master device | *1 "Wait state at falling edge of first SCLn after IICBn became master" on page 1112* |
| Δ1 | Upon detection of the falling edge of the 9th SCLn during address transfer after the start condition | *2 "Wait state upon detection of the falling edge of the 9th SCLn during address transfer after the start condition" on page 1113* |
| Δ2 | Upon detection of the falling edge of the 8th SCLn during data transfer | *3 "Wait state upon detection of the falling edge of the 8th SCLn during data transfer" on page 1113* |
| Δ3 | Upon detection of the falling edge of the 9th SCLn during data transfer | *4 "Wait state upon detection of the falling edge of the 9th SCLn during data transfer" on page 1114* |

**Note**
ST: Start condition
AD6 to AD0: Address
R/W: Transfer direction specification
ACK: Acknowledge
D7 to D0: Data
SP: Stop condition

The method to exit the wait state differs according to the wait state.

Exit the wait state by applying the appropriate method for each of the four wait states as described below.

**(1) Wait state at falling edge of first SCLn after IICBn became master**

Δ0 indicates the wait state when the data to be transferred has not been written (to the IICBnDAT register) when the falling edge of the first SCLn after the IICBn became the master is detected, after 1 was written to the IICBnTRG.IICBnSTT bit.

**(a) Wait state transit condition**

The IICBn enters the wait state if data is not written to the IICBnDAT register in the period from when the IICBnSTT bit becomes 1 until the Δ0 timing, upon detection of the first falling edge of SCLn after the IICBn became master, after 1 was written to the IICBnSTT bit.

The valid times to write to the IICBnDAT register for each of these cases are shown in *Figure 20-12 "Valid times to write to IICBnDAT register"*.

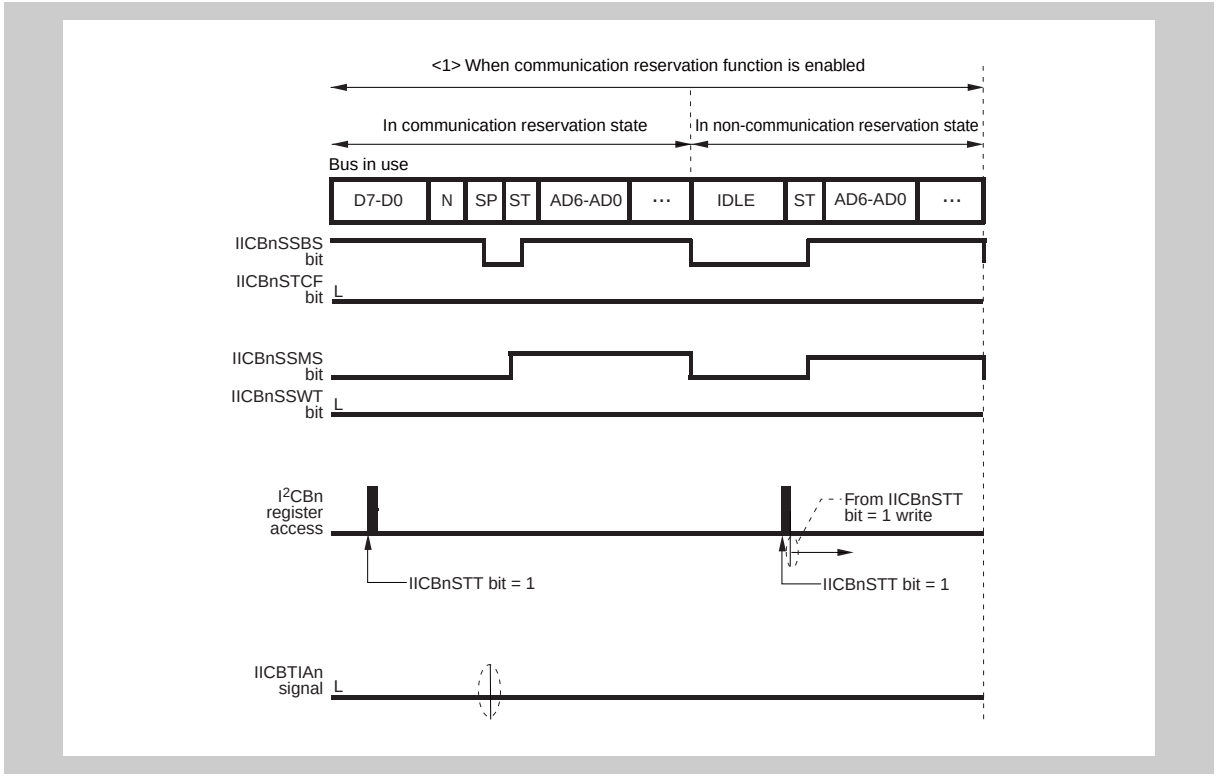



**Figure 20-12 Valid times to write to IICBnDAT register**

**Caution** The communication reservation function is disabled (<2> in the above figure) while the IICBnSTR0.IICBnSTCF bit is 0.
When the IICBnSTCF bit becomes 1, setting from IICBnSTCF bit = 1 write is required again.

**(b) Wait state exit conditions**

Exit the wait state by writing to the IICBnDAT register.

**(2) Wait state upon detection of the falling edge of the 9th SCLn during address transfer after the start condition**

Δ1 indicates the wait state entered upon completion of address transfer.

**(a) Wait state transit condition**

In the single transfer mode, the IICBn always enters the wait state while it operates as the master.

While the IICBn operates as a slave, it enters the wait state upon an address match, or upon extension code detection while the IICBnSLWT bit is 1.

**(b) Wait state exit conditions**

Exit the wait state by writing to the IICBnDAT register during transmission, or by writing 1 to the IICBnWRET bit during reception. When the IICBn operates as the master and the IICBnSTR0.IICBnSSAC bit is 0 or the IICBn is at the transmission side, the wait state can be exited by writing 1 to the IICBnTRG.IICBnSTT or the IICBnTRG.IICBnSPT bit.

**(3) Wait state upon detection of the falling edge of the 8th SCLn during data transfer**

Δ2 indicates the wait state entered upon detection of the falling edge of the 8th SCLn during data transfer.

**(a) Wait state transit condition**

When the IICBn participates in communications and the IICBnCTL0.IICBnSLWT bit is 0, the IICBn enters the wait state if the falling edge of the 8th SCLn is detected.

**(b) Wait state exit conditions**

Exit the wait state by writing to the IICBnDAT register during transmission, or by writing 1 to the IICBnTRG.IICBnWRET bit during reception.

**(4) Wait state upon detection of the falling edge of the 9th SCLn during data transfer**

Δ3 indicates the wait state entered upon detection of the falling edge of the 9th SCLn during data transfer.

**(a) Wait state transit condition**

When the IICBn participates in communications and the IICBnCTL0.IICBnSLWT bit is 1, the IICBn enters the wait state if the falling edge of the 9th SCLn is detected.

**(b) Wait state exit conditions**

The wait state exit conditions are listed for each transfer mode in *Table 20-21 "Wait state exit conditions"*.

**Table 20-21 Wait state exit conditions**

| Master/ slave | Transfer mode | Transfer direction | IICBnSTR0. IICBnSSAC bit | Exit conditions |
|---|---|---|---|---|
| Master | Single transfer mode | Reception | 0 | IICBnTRG.IICBnSTT bit = 1 or IICBnTRG.IICBnSPT bit = 1 |
| | | | 1 | IICBnTRG.IICBnLRET bit = 1 |
| | | Transmission | 0 | IICBnSTT bit = 1 or IICBnSPT bit = 1 |
| | | | 1 | Write to IICBnDAT register or IICBnSTT bit = 1 or IICBnSPT bit = 1 |
| Slave | Single transfer mode | Reception | - | IICBnWRET bit = 1 |
| | | Transmission | 0 | IICBnWRET bit = 1 |
| | | | 1 | Write to IICBnDAT register[a] |

a) Condition for exiting the wait state that was entered when no transmit data has been written to the data register.

### 20.7.4 Extension code

The processing when the extension code is received differs according to the data after the extension code and thus must be executed through the user's software.

Therefore, the operation differs from that during normal slave address reception. These differences are described below.

(1) When the upper 4 bits of the received address are 0000 or 1111, the extension code reception flag (IICBnSTR0.IICBnSSEX bit) is set to 1 to indicate that an extension code has been received. The IICBn enters the wait state (IICBnTRG.IICBnSSWT = 1). The IICBnSTR0.IICBnSSDR bit is then set (to 1).

(2) During address transfer, the acknowledge output can be controlled by setting the IICBnCTL0.IICBnLAC bit. (Note that an acknowledge is always output upon an address match, regardless of the setting of this bit, during address transfer for normal slave address reception.)

(3) During transmission while the IICBnCTL0.IICBnSLWT bit is 0, exit the wait state by writing to the IICBnDAT register. During transmission while the IICBnSLWT bit is 1, or during reception, exit the wait state by writing 1 to the IICBnTRG.IICBnWRET bit.

(4) At the falling edge of the 9th clock, if the IICBnSLWT bit is 1, the interrupt request signal (IICBTIAn) is output and the IICBn enters the wait state (IICBnTRG.IICBnSSWT = 1). If the IICBnCTL0.IICBnSLWT bit is 0, the interrupt request signal (IICBTIAn) is not output and the IICBn does not enter the wait state.

(5) If the IICBn receives an extension code, it participates in communications even if the addresses do not match.

For example, to avoid operating the IICBn as a slave device after receiving an extension code, set the IICBnTRG.IICBnLRET bit to 1. The IICBn enters the standby state for the next communication.

## 20.8 Interrupt Request Signals

**Caution** In this section, the operation when an extension code is received is omitted. For details about the extension code, refer to *20.7.4 "Extension code"*.

The IICBn has the data transmit/receive interrupt request signal (IICBTIAn), which is a pulse of one PCLK clock width. The interrupt request signal is explained below.

### 20.8.1 Single transfer mode

The interrupt request signal timing in the single transfer mode is described below.
During the single transfer mode, for the IICBTIAn interrupt request signal, whether to output an interrupt is judged based on the IICBn state when the falling edge of SCLn is detected during the bus cycle. Note, however, that whether to output an interrupt is judged based on the IICBn state when a stop condition is detected at the ∆4 timing.

**Table 20-22 Interrupt request signal output timing (single transfer mode)**


| Output timing | Description | Refer to: |
|---|---|---|
| ∆1 | Upon detection of the falling edge of the 9th SCLn during address transfer | *1 “Interrupt request signal output conditions and output interrupt request signal during address transfer” on page 1118* |
| ∆2 | Upon detection of the falling edge of the 8th SCLn during data transfer | *2 “Interrupt request signal output conditions and interrupt request signals output during data transfer” on page 1119* |
| ∆3 | Upon detection of the falling edge of the 9th SCLn during data transfer | *2 “Interrupt request signal output conditions and interrupt request signals output during data transfer” on page 1119* |

**Note**
ST: Start condition
AD6 to AD0: Address
R/W: Transfer direction specification
ACK: Acknowledge
D7 to D0: Data
SP: Stop condition

### (1) Interrupt request signal output conditions and output interrupt request signal during address transfer

Δ1 in *Table 20-22 "Interrupt request signal output timing (single transfer mode)"* indicates the interrupt request signal output timing during address transfer. *Table 20-23 "Interrupt request signal output conditions and interrupt request signal output during address transfer (single transfer mode)"* lists the interrupt request signal output conditions and the interrupt request signal output (IICBTIAn) at the Δ1 timing.

**Table 20-23 Interrupt request signal output conditions and interrupt request signal output during address transfer (single transfer mode)**

| IICBn SSMS | IICBn ALDF | IICBn SLWT | IICBn SSCO | Δ1 | | Remark |
|---|---|---|---|---|---|---|
| | | | | Interrupt | Wait | |
| 1 | 0 | × | × | IICBTIAn | Wait | - |
| 1 | 1 | × | × | This state does not exist. | | - |
| 0 | 0 | × | 1 | IICBTIAn | Wait | - |
| 0 | 1 | × | 0 | - | - | After arbitration loss, non-participation in communications |
| 0 | 1 | × | 1 | IICBTIAn | Wait | - |

**Note** ×: don't care

### (2) Interrupt request signal output conditions and interrupt request signals output during data transfer

Δ2 and Δ3 in *Table 20-22 "Interrupt request signal output timing (single transfer mode)"* indicate the interrupt request signal output timings during data transfer. The interrupt request signal output timing at Δ2 or Δ3 is determined by the setting of the IICBnCTL0.IICBnSLWT bit. *Table 20-24 "Interrupt request signal output conditions and interrupt request signals output during address transfer (single transfer mode)"* lists the interrupt request signal output conditions and the interrupt request signal (IICBTIAn) output at the Δ2 and Δ3 timings.

**Table 20-24 Interrupt request signal output conditions and interrupt request signals output during address transfer (single transfer mode)**

| IICBn SSMS | IICBn ALDF | IICBn SLWT | IICBn SSCO | Δ2 | | Δ3 | | Remark |
|---|---|---|---|---|---|---|---|---|
| | | | | Interrupt | Wait | Interrupt | Wait | |
| 1 | 0 | 0 | × | IICBTIAn | Wait | - | - | - |
| 1 | 0 | 1 | × | - | - | IICBTIAn | Wait | - |
| 1 | 1 | × | × | This state does not exist. | | | | - |
| 0 | 0 | × | 0 | - | - | - | - | Non-participation in communications |
| 0 | 0 | 0 | 1 | IICBTIAn | Wait | - | - | - |
| 0 | 0 | 1 | 1 | - | - | IICBTIAn | Wait | - |
| 0 | 1 | 0 | 0 | - | - | - | - | Non-participation in communications after arbitration loss |
| 0 | 1 | 1 | 0 | - | - | - | - | Non-participation in communications after arbitration loss |
| 0 | 1 | 0 | 1 | IICBTIAn | Wait | - | - | - |
| 0 | 1 | 1 | 1 | - | - | IICBTIAn | Wait | - |

**Note** ×: don't care

## 20.9 Interrupt Outputs and Statuses

This section describes the statuses of the IICBnSTR0 register during interrupt output by communication flow.

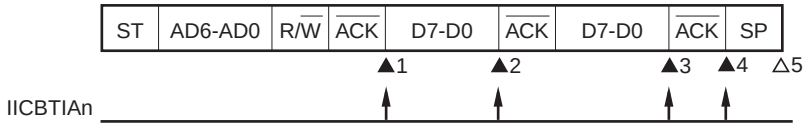
The meanings of the symbols used in the figures are as follows.

| | |
|---|---|
| ST: | Start condition |
| AD6-AD0: | Address |
| R, $\overline{W}$, R/$\overline{W}$: | Transfer direction specification |
| $\overline{ACK}$: | Acknowledge |
| NACK: | Not acknowledge |
| D7-D0: | Data |
| SP: | Stop condition |

### 20.9.1 Single transfer mode (master device operation)

#### (1) Start ~ Address ~ Data ~ Data ~ Stop (normal transmission/reception)

<1> When IICBnCTL0.IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 1-0100X1 0110--00B

▲2: IICBnSTR0 register = 1-0100X0 0100--00B

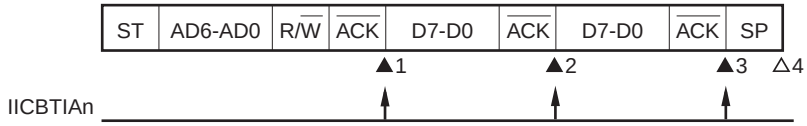
▲3: IICBnSTR0 register = 1-0100X0 0100--00B (IICBnSLWT bit = 1)

▲4: IICBnSTR0 register = 1-0100XX 0100--00B (IICBnTRG.IICBnSPT bit = 1)

△5: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 1-0100X1 0110--00B

▲2: IICBnSTR0 register = 1-0100X0 0100--00B

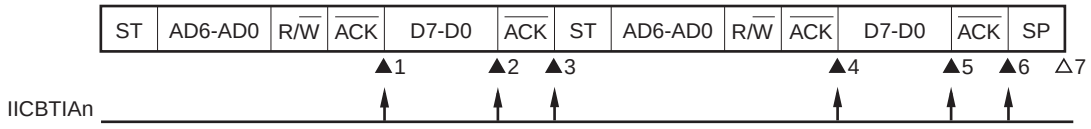
▲3: IICBnSTR0 register = 1-0100XX 0100--00B (IICBnSPT bit = 1)

△4: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

**(2) Start ~ Address ~ Data ~ Start ~ Address ~ Data ~ Stop (restart)**

<1> When IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 1-0100X1 0110--00B

▲2: IICBnSTR0 register = 1-0100X0 0100--00B (IICBnSLWT bit = 1)

▲3: IICBnSTR0 register = 1-0100XX 0100--00B (IICBnTRG.IICBnSTT bit = 1, IICBnSLWT bit = 0)

▲4: IICBnSTR0 register = 1-0100X1 0110--00B

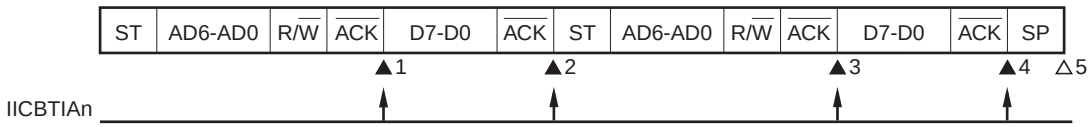
▲5: IICBnSTR0 register = 1-0100X0 0100--00B (IICBnSLWT bit = 1)

▲6: IICBnSTR0 register = 1-0100XX 0100--00B (IICBnSPT bit = 1)

△7: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 1-0100X1 0110--00B

▲2: IICBnSTR0 register = 1-0100XX 0100--00B (IICBnSTT bit = 1)

▲3: IICBnSTR0 register = 1-0100X1 0110--00B

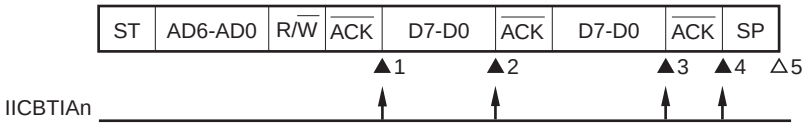
▲4: IICBnSTR0 register = 1-0100XX 0100--00B (IICBnSPT bit = 1)

△5: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

### (3) Start ~ ExCode ~ Data ~ Data ~ Stop (extension code transmission)

<1> When IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 1-0110X1 0110--00B

▲2: IICBnSTR0 register = 1-0110X0 0100--00B

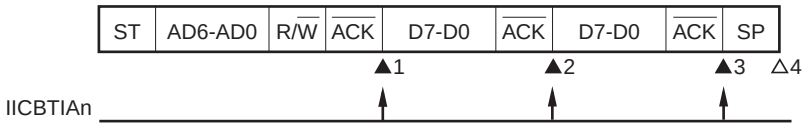
▲3: IICBnSTR0 register = 1-0110X0 0100--00B (IICBnSLWT bit = 1)

▲4: IICBnSTR0 register = 1-0110XX 0100--00B (IICBnSPT bit = 1)

△5: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 1-0110X1 0110--00B

▲2: IICBnSTR0 register = 1-0110X1 0100--00B

▲3: IICBnSTR0 register = 1-0110XX 0100--00B (IICBnSPT bit = 1)

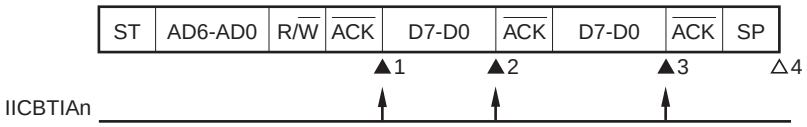
△4: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

### 20.9.2 Single transfer mode (slave device operation: during slave address reception (IICBnSTR0.IICBnSSC0 bit = 1))

**(1) Start ~ Address ~ Data ~ Data ~ Stop**

<1> When IICBnCTL0.IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 0-0101X1 0110--00B

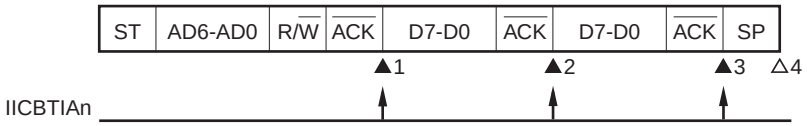
▲2: IICBnSTR0 register = 0-0101X0 0100--00B

▲3: IICBnSTR0 register = 0-0101X0 0100--00B

△4: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 0-0101X1 0110--00B

▲2: IICBnSTR0 register = 0-0101X1 0100--00B

▲3: IICBnSTR0 register = 0-0101XX 0100--00B

△4: IICBnSTR0 register = 0-000000 0001--00B

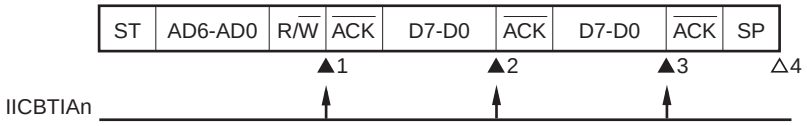
**Note** ▲ Always output
- Undefined
X don't care

### 20.9.3 Single transfer mode (slave device operation: during extension code reception (IICBnSTR0.IICBnSSEX bit = 1))

The IICBn always participates in communications when it receives an extension code.

**(1) Start ~ Code ~ Data ~ Data ~ Stop**

<1> When IICBnCTL0.IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 0-0110X0 0110--00B

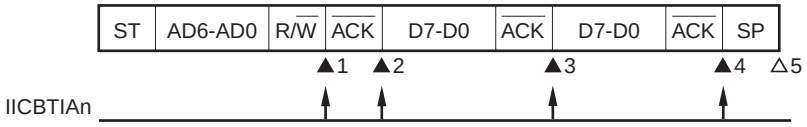
▲2: IICBnSTR0 register = 0-0110X0 0100--00B

▲3: IICBnSTR0 register = 0-0110X0 0100--00B

△4: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
\- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 0-0110X0 0110--00B

▲2: IICBnSTR0 register = 0-0110X1 0110--00B

▲3: IICBnSTR0 register = 0-0110X0 0100--00B

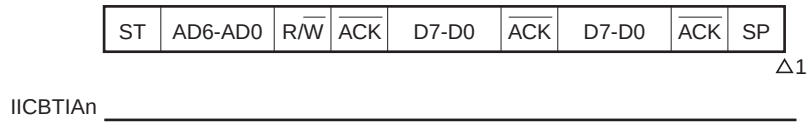
▲4: IICBnSTR0 register = 0-0110XX 0100--00B

△5: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
\- Undefined
X don't care

### 20.9.4 Single transfer mode (non-participation in communications)

#### (1) Start ~ Code ~ Data ~ Data ~ Stop




△1: IICBnSTR0 register = 0-000000 0001--00B

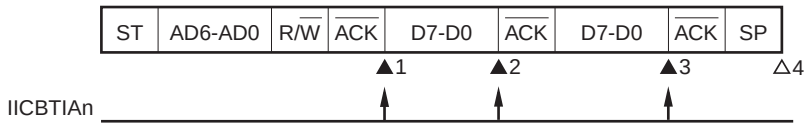
**Note** - Undefined

### 20.9.5 Single transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1): operation as slave after arbitration loss)

When the IICBn operates as a master in a multi-master system, read the IICSn.MSTSn bit to check the arbitration result each time the INTIICn interrupt is output.

**(1) Address match after arbitration loss**

<1> When IICBnCTL0.IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 0-0101X1 0110--01B (IICBnSTRC.IICBnCLAF bit = 1)

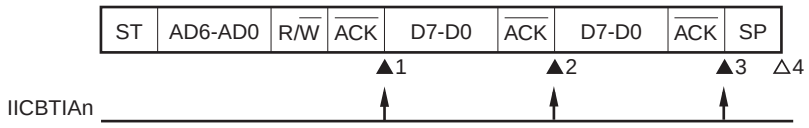
▲2: IICBnSTR0 register = 0-0101X0 0100--00B

▲3: IICBnSTR0 register = 0-0101X0 0100--00B

△4: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don’t care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 0-0101X1 0110--01B (IICBnCLAF bit = 1)

▲2: IICBnSTR0 register = 0-0101X1 0100--00B

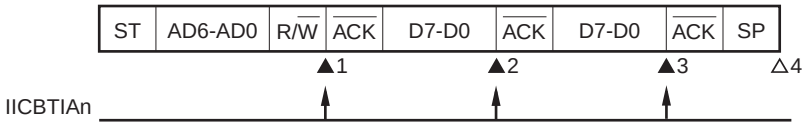
▲3: IICBnSTR0 register = 0-0101XX 0100--00B

△4: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don’t care

### (2) Upon extension code detection after arbitration loss

<1> When IICBnCTL0.IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 0-0110X0 0110--01B (IICBnCLAF bit = 1)

▲2: IICBnSTR0 register = 0-0110X0 0100--00B

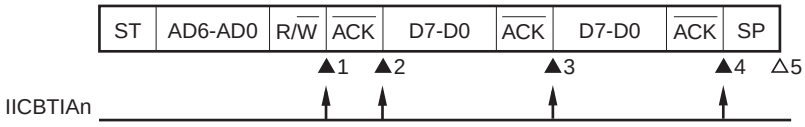
▲3: IICBnSTR0 register = 0-0110X0 0100--00B

△4: IICBnSTR0 register = 0-000000 0001--00B

**Notes 1.** ▲ Always output
\- Undefined
X don't care

**2.**n = 0 to 5

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 0-0110X0 0110--01B (IICBnCLAF bit = 1)

▲2: IICBnSTR0 register = 0-0110X1 0110--00B

▲3: IICBnSTR0 register = 0-0110X0 0100--00B

▲4: IICBnSTR0 register = 0-0110XX 0100--00B

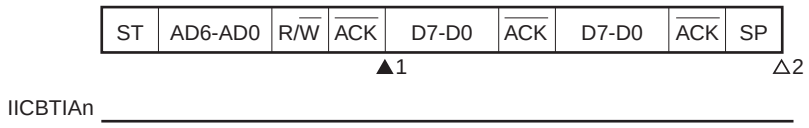
△5: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
\- Undefined
X don't care

### 20.9.6 Single transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1): non-participation in communications after arbitration loss)

When the IICBn is used as a master in a multi-master system, read the IICSn.MSTSn bit to check the arbitration result each time the INTIICn interrupt is output.

**(1) Arbitration loss during transmission of slave address**



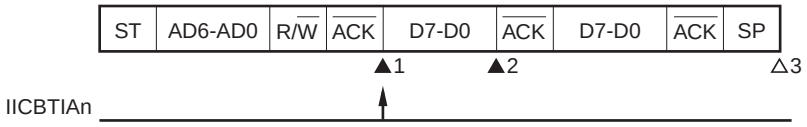

▲1: IICBnSTR0 register = 0-0000X1 0110--01B (IICBnSTRC.IICBnCLAF bit = 1)

△2: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don’t care

### (2) Arbitration loss during data transfer

<1> When IICBnSLWT bit is 0



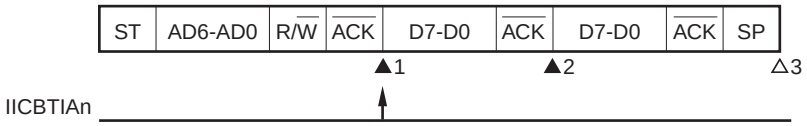

▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 0-0000X0 0100--01B (IICBnCLAF bit = 1)

△3: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 1-1000X1 0110--00B

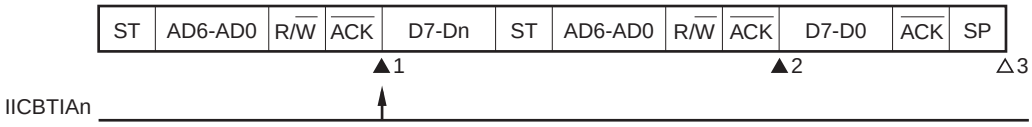
▲2: IICBnSTR0 register = 0-0000X0 0100--01B (IICBnCLAF bit = 1)

△3: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

### (3) Arbitration loss for the restart condition during data transfer

<1> When IICBnSLWT bit is 1 (extension code mismatch, address mismatch)



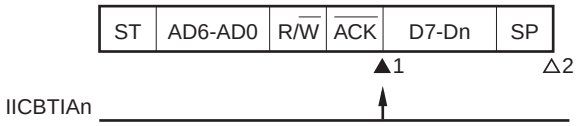

▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 0-0000X0 0100--01B (IICBnCLAF bit = 1)

△3: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

### (4) Arbitration loss for the stop condition during data transfer



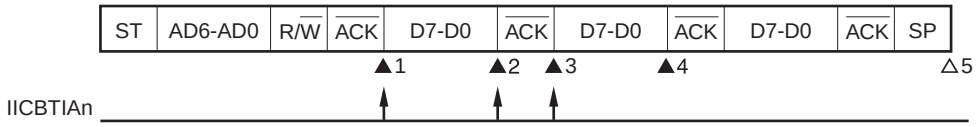

▲1: IICBnSTR0 register = 1-1000X1 0110--00B

△2: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

**(5) Arbitration loss because the SDAn signal is low level when attempting to output restart condition**

<1> When IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 1-1000X0 0100--00B (IICBnSLWT bit = 1)

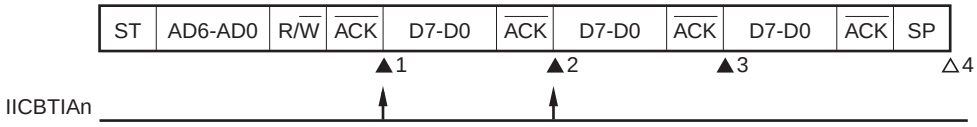
▲3: IICBnSTR0 register = 1-1000XX 0100--00B (IICBnSLWT bit = 0, IICBnTRG.IICBnSTT bit = 1)

▲4: IICBnSTR0 register = 0-0000X0 0100--01B (IICBnCLAF bit = 1)

△5: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 1-1000XX 0100--00B (IICBnSLWT bit = 0, IICBnSTT bit = 1)

▲3: IICBnSTR0 register = 0-0000X0 0100--01B (IICBnCLAF bit = 1)

△4: IICBnSTR0 register = 0-000000 0001--00B

**Note** ▲ Always output
- Undefined
X don't care

### (6) Arbitration loss for the stop condition when attempting to output restart condition

<1> When IICBnSLWT bit is 0


▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 1-1000X0 0100--00B (IICBnSLWT bit = 0)

▲3: IICBnSTR0 register = 1-0000XX 0100--00B (IICBnSTT bit = 1)

△4: IICBnSTR0 register = 0-000000 0001--01B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1


▲1: IICBnSTR0 register = 1-1000X1 0110--00B

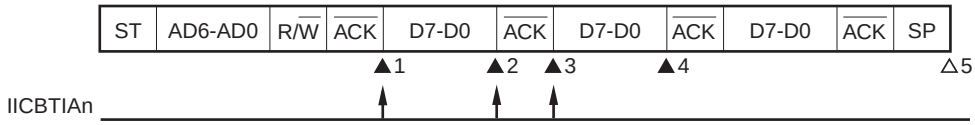
▲2: IICBnSTR0 register = 1-0000XX 0100--00B (IICBnSTT bit = 1)

△3: IICBnSTR0 register = 0-000000 0001--01B

**Note** ▲ Always output
- Undefined
X don't care

### (7) Arbitration loss because the SDAn signal is low level when attempting to output stop condition

<1> When IICBnSLWT bit is 0




▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 1-1000X0 0100--00B (IICBnSLWT bit = 1)

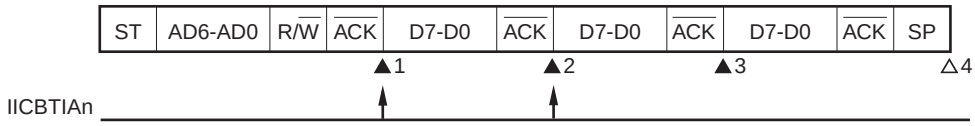
▲3: IICBnSTR0 register = 1-1000XX 0100--00B (IICBnSLWT bit = 0, ICBnTRG.IICBnSPT bit = 1)

▲4: IICBnSTR0 register = 0-0000XX 0100--01B (IICBnCLAF bit = 1)

△5: IICBnSTR0 register = 0-000000 0001--01B

**Note** ▲ Always output
- Undefined
X don't care

<2> When IICBnSLWT bit is 1




▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 1-1000XX 0100--00B (IICBnSPT bit = 1)

▲3: IICBnSTR0 register = 0-0000XX 0100--01B (IICBnCLAF bit = 1)

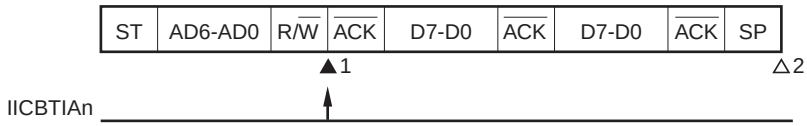
△4: IICBnSTR0 register = 0-000000 0001--01B

**Note** ▲ Always output
- Undefined
X don't care

### 20.9.7 Single transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1): non-participation in communications after arbitration loss (during extension code transfer))

When the IICBn is used as a master in a multi-master system, read the IICSn.MSTSn bit to check the arbitration result each time the INTIICn interrupt is output.

**(1) Arbitration loss during extension code transfer**

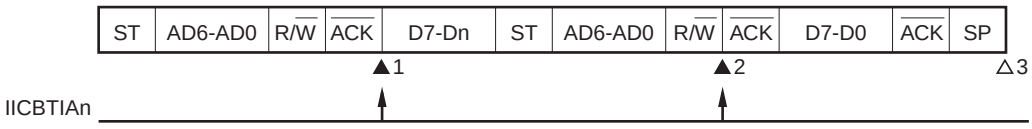



▲1: IICBnSTR0 register = 0-1100X0 0110--01B (IICBnSTRC.IICBnCLAF bit = 1, IICBnTRG.IICBnLRET bit = 1)

△2: IICBnSTR0 register = 0-000000 0001--01B

**Note** ▲ Always output
- Undefined
X don't care

**(2) Arbitration loss for the restart condition during data transfer (extension code match)**




▲1: IICBnSTR0 register = 1-1000X1 0110--00B

▲2: IICBnSTR0 register = 0-1100X0 0100--01B (IICBnCLAF bit = 1, IICBnLRET bit = 1)

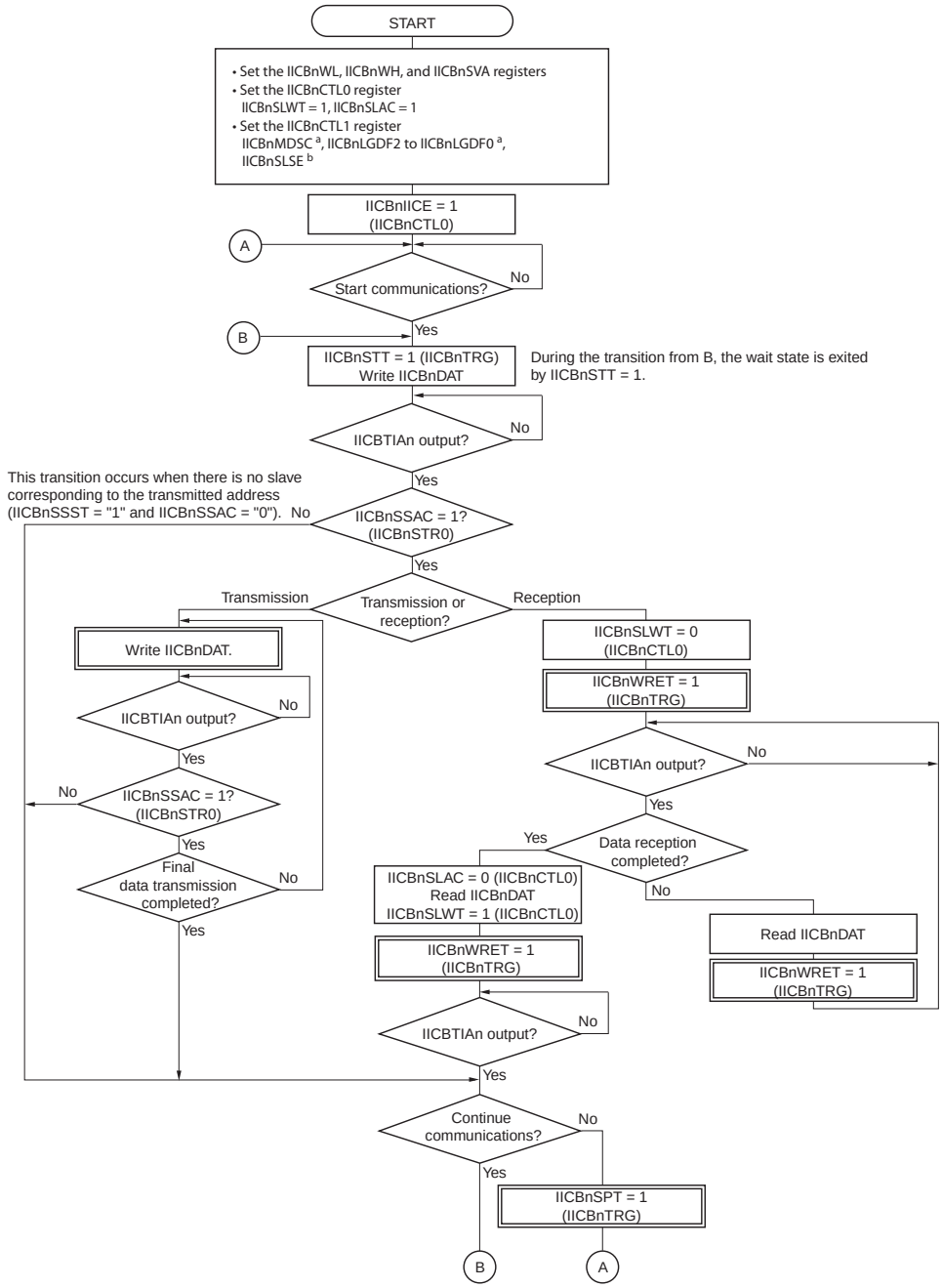
△3: IICBnSTR0 register = 0-000000 0001--01B

**Note** ▲ Always output
- Undefined
X don't care

# 20.10 Setting Sequence

## 20.10.1 Single master environment

### (1) Master operate setting sequence during single transfer mode

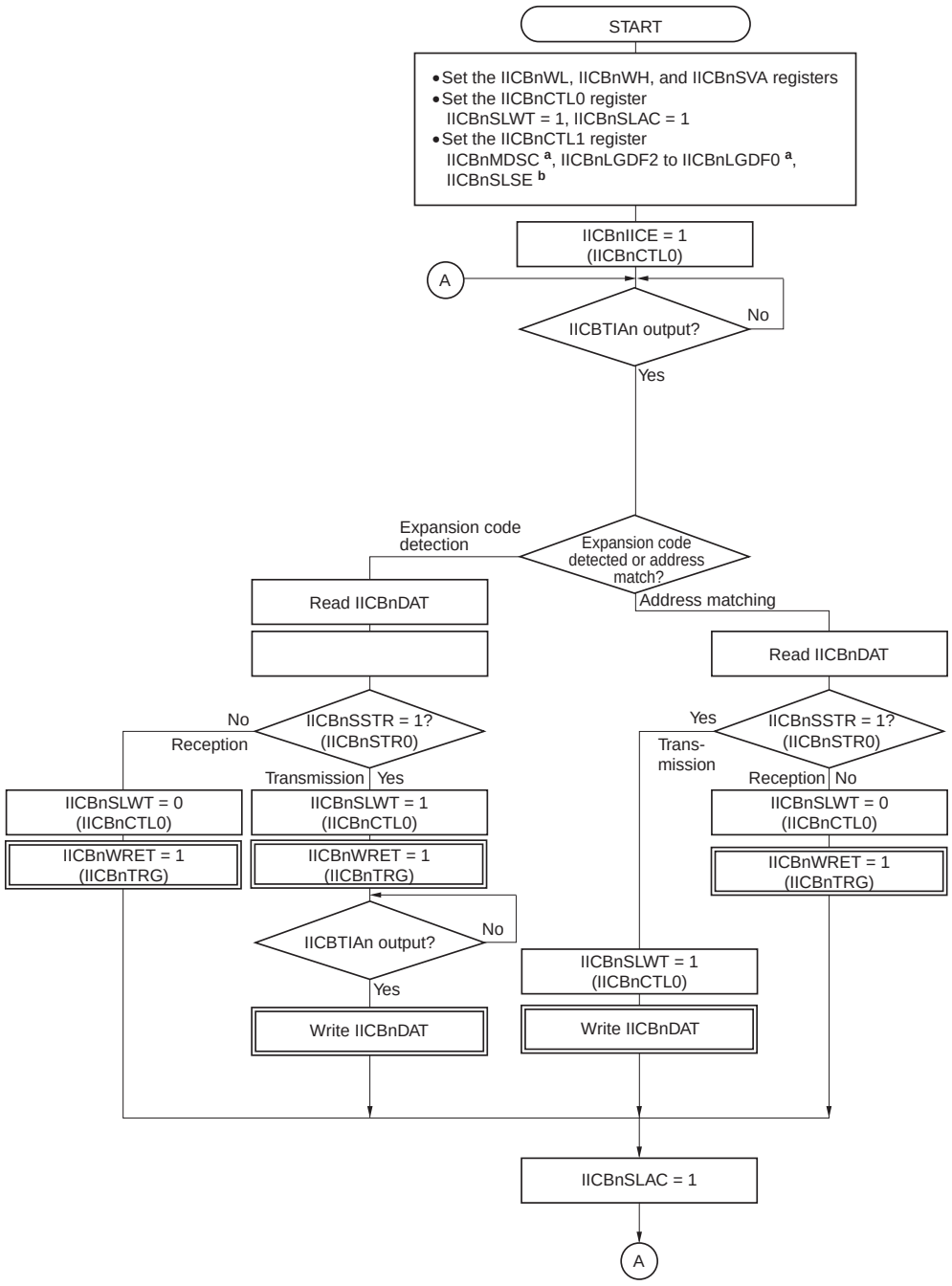



a) Enter settings according to the environment.
b) Set 1 to enable output of a start condition in the initial communication state. Otherwise, set 0.

**Note** The double-boxed items indicate processing that exits the wait state.

**Figure 20-13 Master operate setting sequence during single transfer mode (single master environment)**

### (2) Slave operate setting sequence during single transfer mode




a) Enter settings according to the environment.

**Note** The double-boxed items indicate processing that exits the wait state.

**Figure 20-14 Slave operate setting sequence during single transfer mode (single master environment)**

# Chapter 21 Key Return Function (KR)

This chapter contains a generic description of the Key Return Function.

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 21.1 V850E2/Fx4-G KR Features

**Instances**

This microcontroller has following number of instances of the Key Return Function.

**Table 21-1 Instances of KR**

| Key Return Function | |
|---|---|
| Instance | 1 |
| Name | KR0 |

**Instances index n**

Throughout this chapter, the individual instances of a Key Return Function is identified by the index “n” (n = 0), for example KRnM for the Key Return mode register.

**Register addresses**

All KRn register addresses are given as address offsets to the individual base address <KRn_base>.
The base address <KRn_base> of each KRn is listed in the following table:

**Table 21-2 Register base addresses <KRn_base>**

| KRn instance | <KRn_base> address |
|---|---|
| KR0 | FF82 $B000_H$ |

**Clock supply** All Key Return Functions provide one clock input:

**Table 21-3 KRn clock supply**

| KRn instance | KRn clock | Connected to |
|---|---|---|
| KR0 | PCLK | Clock Controller CKSCLK_A02 |

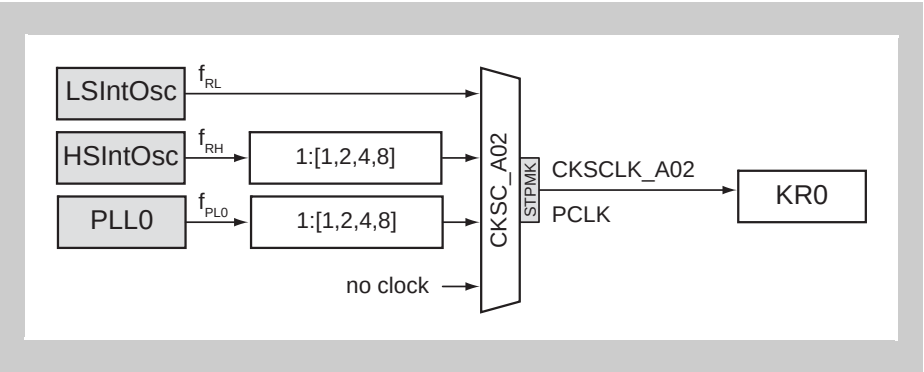



**Figure 21-1 KR clock supply**

**Interrupts** The Key Return Function can generate the following interrupt requests:

**Table 21-4 KRn interrupt requests**

| KRn signals | Function | Connected to |
|---|---|---|
| KR0TIKR | Communication status interrupt | Interrupt Controller INTKR0[a] |

a) This interrupt can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

**KRH/W reset** The Key Return Function and their registers are initialized by the following reset signal:

**Table 21-5 KRn reset signal**

| KRn | Reset signal |
|---|---|
| KR0 | • Reset Controller SYSRES |

**I/O signals** The I/O signals of the Key Return Function are listed in the following table.

**Table 21-6 KRn I/O signals**

| KRn signal | Function | Connected to |
|---|---|---|
| KR0TPKR7 to KR0TPKR0 | Key input signals | Ports KR0[a] to KR7[a] |

a) These input signals are passed through a noise filter, refer to the section *"Port Filters"* in the chapter *"Port Functions"*.

## 21.2 Functional Overview

**Features summary** The Key Return Function has the following feature:

A key interrupt request signal (KRnTIKR) can be generated by inputting a low level to any of the eight key input pins (KRnTPKR0 to KRnTPKR7).

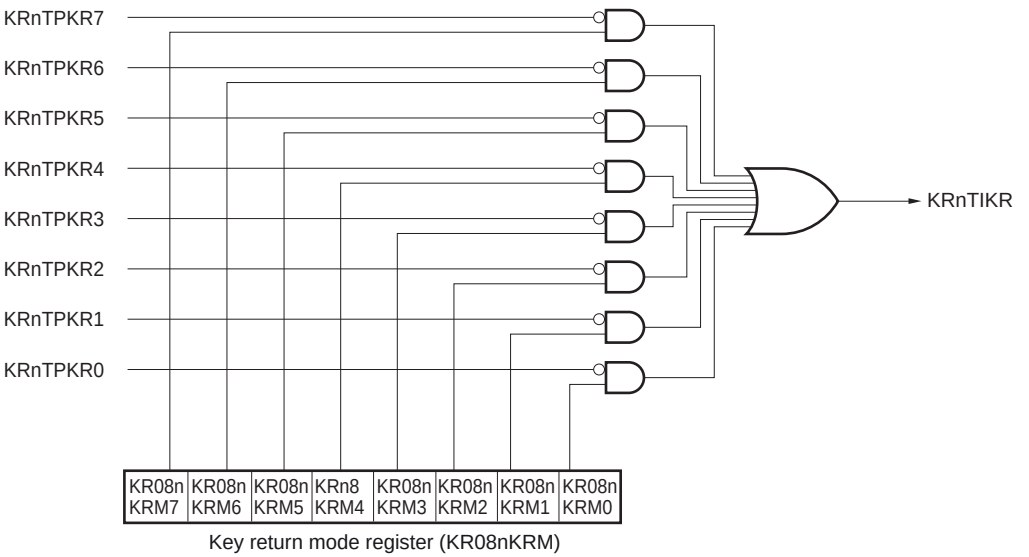



**Figure 21-2 Block diagram of the Key Return Function**

## 21.3 Functional Description

### 21.3.1 Interrupt request KRnTIKR

The interrupt request KRnTIKR is generated when a low level is input to key input pin KRnTPKR[7:0] while input to pin KRnTPKR[7:0] is enabled (KR08nM.KR08nKRM[7:0] = 1).

The following figure shows the interrupt request generation:

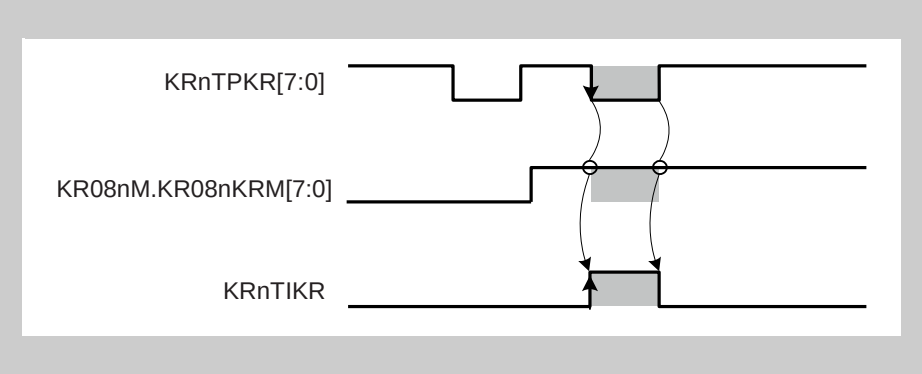



**Figure 21-3 Interrupt request generation**

**Cautions**

1. If a low level is input to any key input KRnTPKR[7:0], KRnTIKR is not generated again, even if another key input changes from high to low level.
2. An unintended key interrupt request KRnTIKR may occur under the following conditions:
   - KR08nM.KR08nKRM[7:0] = 1 and the key input KRnTPKR[7:0] level changes from high to low.
   - The key input KRnTPKR[7:0] level is low and KR08nM.KR08nKRM[7:0] is changed from 0 to 1.

   Thus mask (i.e. disable) KRnTIKR in the Interrupt Controller before changing KR08nM.KR08nKRM[7:0] from 0 to 1 or vice versa.

## 21.4 Registers

This section contains a description of all registers of the Key Return Function.

### 21.4.1 Key Return Function registers overview

The Key Return Function is controlled and operated by the following registers:

**Table 21-7 Key Return Function registers overview**

| Register name | Shortcut | Address |
|---|---|---|
| Key return mode register | KR08nM | <KRn_base> |

**<KRn_base>** <KRn_base> of the KRn are defined in the first section of this chapter under the key word “Register addresses”.

### 21.4.2 Key Return Function registers details

**(1) KR08nM - Key return mode register**

This register enables/disables the key input signal detection.

**Access** This register can be read/written in 8-bit units.

**Address** <KRn_base>

**Initial Value** $00_H$. This register is cleared by any reset.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| KR08n KRM7 | KR08n KRM6 | KR08n KRM5 | KR08n KRM4 | KR08n KRM3 | KR08n KRM2 | KR08n KRM1 | KR08n KRM0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 21-8 KR08nM register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | KR08n KRMm | Enables/disables the key input signal detection<br>0: Disabled<br>1: Enabled |

# Chapter 22 A/D Converter (ADAA)

This chapter contains a generic description of the A/D Converter.

The first section describes all properties specific to the V850E2/Fx4-G, such as instances, register base addresses, input/output signal names, etc.
The subsequent sections describe the features that apply to all implementations.

## 22.1 V850E2/Fx4-G ADAA Features

**Instances**

This microcontroller has following number of instances of the A/D Converter.

**Table 22-1 Instances of ADAA**

| A/D Converter | |
|---|---|
| Instance | 1 |
| Name | ADAA0 |

**Instances index n**

Throughout this chapter, the individual instances of the ADAA are identified by “n” (n = 0), for example ADAAn, or ADAAnCTL0 for the control register 0 of ADAAn.

**Channel group index i**

The A/D Converter has 3 A/D conversion channel groups, abbreviated with CG. Throughout this chapter, the individual channel group is identified by the index “i” (i = 0 to 2), for example, ADAAnIOCi for the CGi interrupt controller register.

**Channel index m**

Each A/D Converter has several A/D conversion channels. Throughout this chapter, the individual channels of each ADAA are identified by the index “m”, for example, ADAAnCmCR for the channel m conversion result register.

**Table 22-2 ADAAn channel select indices m**

| ADAAn instance | V850E2/FF4-G | V850E2/FG4-G |
|---|---|---|
| ADAA0 | m = 3 to 13 | m = 0 to 15 |

**Register addresses**

All ADAA register addresses are given as address offsets to the individual base address <ADAAn_base>.
The base address <ADAAn_base> of each ADAAn is listed in the following table:

**Table 22-3 Register base addresses <ADAAn_base>**

| ADAAn instance | <ADAAn_base> address |
|---|---|
| ADAA0 | $FF81\ D000_H$ |

**Clock supply**

All A/D Converters provide one clock input:

**Table 22-4 ADAAn clock supply**

| ADAAn instance | ADAAn clock | Connected to |
|---|---|---|
| ADAA0 | PCLK | Clock Controller CKSCLK_012 |

**Interrupts and DMA**

The A/D Converters can generate the following interrupt and DMA requests:

**Table 22-5 ADAAn interrupt and DMA requests**

| ADAAn signals | Function | Connected to |
|---|---|---|
| **ADAA0:** | | |
| INTADAA0T0 | End of conversion CG0 | Interrupt Controller INTADAA0I0<br>DMA Controller trigger 30 |
| INTADAA0T1 | End of conversion CG1 | Interrupt Controller INTADAA0I1<br>DMA Controller trigger 31 |
| INTADAA0T2 | End of conversion CG2 | Interrupt Controller INTADAA0I2<br>DMA Controller trigger 32 |
| INTADAA0LLT | Last conversion | Interrupt Controller INTADAA0LLT[a]<br>DMA Controller trigger 33 |
| INTADAA0ERR | Error interrupt | Interrupt Controller INTADAA0ERR[a] |

a) These interrupts can be used as a wake-up source from any stand-by mode. Refer to the chapter *"Stand-by Controller (STBC)"* for details.

**ADAA H/W reset**

The A/D Converters and their registers are initialized by the following reset signal:

**Table 22-6 ADAAn reset signal**

| ADAAn | Reset signal |
|---|---|
| ADAA0 | • Reset Controller SYSRES<br>• Stand-by Controller DPSTPWU (wake-up from DEEPSTOP mode) |

**I/O signals** The I/O signals of the A/D Converters are listed in the following table.

**Table 22-7 ADAAn I/O signals**

| ADAAn signal | Function | Connected to |
|---|---|---|
| **ADAA0:** | | |
| ADAA0I0 to ADAA0I15 | Analog input channels 0 to 15 | Pins ADAA0I0 to ADAA0I15 |
| ADAA0TTRGi | H/W trigger for CGi | H/W trigger expansion i, refer to the section *"H/W Trigger Expansion"* below |
| AVREFP | Positive reference voltage | Power supply A0VDD |
| AVREFM | Negative reference voltage | Power supply A0VSS |

**ADAA ports** If the ports used as ADAAn inputs shall be used as port inputs, set

- ADAA0CTL1.ADAA0GPS = 1 to use port groups P10 in port mode

## 22.2 H/W Trigger Expansion

All A/D Converter H/W trigger inputs ADAAnTTRGi are equipped with a H/W trigger expansion module that enables up to 16 signals ADAAnTTINi[15:00] to trigger an A/D conversion process.

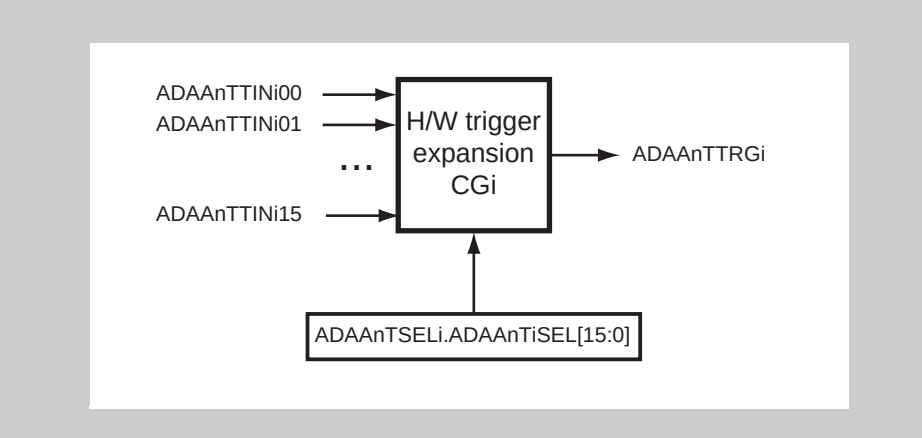



**Figure 22-1 H/W trigger expansion module**

### 22.2.1 ADAAn H/W trigger selection

The selection of active H/W triggers is made via the ADAAn register ADAAnTSELi.ADAAnTiSEL[15:0] bits.

**Caution** It is only allowed to select a single H/W trigger for each ADAAn H/W trigger input ADAAnTTRGi.
Thus do not set more than one bit of each ADAAnTSELi register to “1” at the same time.

### 22.2.2 ADAAn H/W trigger edge selection

The A/D conversion process of channel group i is started with the rising edge of the H/W triggers ADAAnTTRGi.
The rising edge has to be selection via the ADAAnCTL1 register:

- ADAAnCTL1.ADAAnTiETS0 = 0: no edge is selected,
  thus ADAAnTTRGi does not trigger an A/D conversion
- ADAAnCTL1.ADAAnTiETS0 = 1: rising edge is selected,
  thus rising edge of ADAAnTTRGi triggers an A/D conversion

#### (1) External A/D converter H/W triggers

**ADAAnTRG[2:0]** The rising edge of the external ADAAn H/W trigger signals ADAAnTRG[2:0] can start and A/D conversion, provided ADAAnCTL1.ADAAnTiETS0 = 1.

**INTPx** In case an external interrupt signal INTPx is used to start an A/D conversion, the edge of INTPx to trigger A/D conversion can be selected by the port filters:

**Table 22-8 ADAAn H/W trigger edge selection for external interrupts INTPx**

| Control setup | Rising edge | Falling edge | Both edges |
|---|---|---|---|
| ADAAnCTL1.ADAAnTiETS0 = | 1 | | |
| FCLAnCTLm.FCLAnINTLm = | 0 | | |
| FCLAnCTLm.FCLAnINTFm = | 0 | 1 | 1 |
| FCLAnCTLm.FCLAnINTRm = | 1 | 0 | 1 |

**Note** The index “x” stands for the number of the external interrupt signal.
The index “m” denotes the number of the port control register, assigned to INTPx.

For further details about the port filters refer to the *“Port filters”* section in the chapter “Port Functions”.

### 22.2.3 ADAAn H/W trigger tables

The available ADAAn H/W triggers comprises

- external H/W triggers
- external interupts
- output signals of Timer Array Units B
- output signals of Timer Array Units J

**Table 22-9 ADAA0 H/W trigger signals (1/2)**

| ADAA0 channel group | Trigger input signals | | | ADAA0 trigger signal |
|---|---|---|---|---|
| | Name | Control bit | Connected to | |
| CG0 | ADAA0TTIN000 | ADAA0TSEL0.ADAA0T0SEL[00] | Port ADAA0TRG0[a] | ADAA0TTRG0 |
| | ADAA0TTIN001 | ADAA0TSEL0.ADAA0T0SEL[01] | Port INTP2 | |
| | ADAA0TTIN002 | ADAA0TSEL0.ADAA0T0SEL[02] | Port INTP5 | |
| | ADAA0TTIN003 | ADAA0TSEL0.ADAA0T0SEL[03] | TAUB0 INTTAUB0I15 | |
| | ADAA0TTIN004 | ADAA0TSEL0.ADAA0T0SEL[04] | not connected | |
| | ADAA0TTIN005 | ADAA0TSEL0.ADAA0T0SEL[05] | not connected | |
| | ADAA0TTIN006 | ADAA0TSEL0.ADAA0T0SEL[06] | not connected | |
| | ADAA0TTIN007 | ADAA0TSEL0.ADAA0T0SEL[07] | not connected | |
| | ADAA0TTIN008 | ADAA0TSEL0.ADAA0T0SEL[08] | TAUJ0 INTTAUJ0I3 | |
| | ADAA0TTIN009 | ADAA0TSEL0.ADAA0T0SEL[09] | not connected | |
| | ADAA0TTIN010 | ADAA0TSEL0.ADAA0T0SEL[10] | not connected | |
| | ADAA0TTIN011 | ADAA0TSEL0.ADAA0T0SEL[11] | not connected | |
| | ADAA0TTIN012 | ADAA0TSEL0.ADAA0T0SEL[12] | not connected | |
| | ADAA0TTIN013 | ADAA0TSEL0.ADAA0T0SEL[13] | not connected | |
| | ADAA0TTIN014 | ADAA0TSEL0.ADAA0T0SEL[14] | not connected | |
| | ADAA0TTIN015 | ADAA0TSEL0.ADAA0T0SEL[15] | not connected | |
| CG1 | ADAA0TTIN100 | ADAA0TSEL1.ADAA0T1SEL[00] | Port ADAA0TRG1[a] | ADAA0TTRG1 |
| | ADAA0TTIN101 | ADAA0TSEL1.ADAA0T1SEL[01] | Port INTP1 | |
| | ADAA0TTIN102 | ADAA0TSEL1.ADAA0T1SEL[02] | Port INTP4 | |
| | ADAA0TTIN103 | ADAA0TSEL1.ADAA0T1SEL[03] | TAUB0 INTTAUB0I15 | |
| | ADAA0TTIN104 | ADAA0TSEL1.ADAA0T1SEL[04] | not connected | |
| | ADAA0TTIN105 | ADAA0TSEL1.ADAA0T1SEL[05] | not connected | |
| | ADAA0TTIN106 | ADAA0TSEL1.ADAA0T1SEL[06] | not connected | |
| | ADAA0TTIN107 | ADAA0TSEL1.ADAA0T1SEL[07] | not connected | |
| | ADAA0TTIN108 | ADAA0TSEL1.ADAA0T1SEL[08] | TAUJ0 INTTAUJ0I3 | |
| | ADAA0TTIN109 | ADAA0TSEL1.ADAA0T1SEL[09] | not connected | |
| | ADAA0TTIN110 | ADAA0TSEL1.ADAA0T1SEL[10] | not connected | |
| | ADAA0TTIN111 | ADAA0TSEL1.ADAA0T1SEL[11] | not connected | |
| | ADAA0TTIN112 | ADAA0TSEL1.ADAA0T1SEL[12] | not connected | |
| | ADAA0TTIN113 | ADAA0TSEL1.ADAA0T1SEL[13] | not connected | |
| | ADAA0TTIN114 | ADAA0TSEL1.ADAA0T1SEL[14] | not connected | |
| | ADAA0TTIN115 | ADAA0TSEL1.ADAA0T1SEL[15] | not connected | |

**Table 22-9 ADAA0 H/W trigger signals (2/2)**

| ADAA0 channel group | Trigger input signals | | | ADAA0 trigger signal |
|---|---|---|---|---|
| | Name | Control bit | Connected to | |
| CG2 | ADAA0TTIN200 | ADAA0TSEL2.ADAA0T2SEL[00] | Port ADAA0TRG2[a] | ADAA0TTRG2 |
| | ADAA0TTIN201 | ADAA0TSEL2.ADAA0T2SEL[01] | Port INTP0 | |
| | ADAA0TTIN202 | ADAA0TSEL2.ADAA0T2SEL[02] | Port INTP3 | |
| | ADAA0TTIN203 | ADAA0TSEL2.ADAA0T2SEL[03] | TAUB0 INTTAUB0I15 | |
| | ADAA0TTIN204 | ADAA0TSEL2.ADAA0T2SEL[04] | not connected | |
| | ADAA0TTIN205 | ADAA0TSEL2.ADAA0T2SEL[05] | not connected | |
| | ADAA0TTIN206 | ADAA0TSEL2.ADAA0T2SEL[06] | not connected | |
| | ADAA0TTIN207 | ADAA0TSEL2.ADAA0T2SEL[07] | not connected | |
| | ADAA0TTIN208 | ADAA0TSEL2.ADAA0T2SEL[08] | TAUJ0 INTTAUJ0I3 | |
| | ADAA0TTIN209 | ADAA0TSEL2.ADAA0T2SEL[09] | not connected | |
| | ADAA0TTIN210 | ADAA0TSEL2.ADAA0T2SEL[10] | not connected | |
| | ADAA0TTIN211 | ADAA0TSEL2.ADAA0T2SEL[11] | not connected | |
| | ADAA0TTIN212 | ADAA0TSEL2.ADAA0T2SEL[12] | not connected | |
| | ADAA0TTIN213 | ADAA0TSEL2.ADAA0T2SEL[13] | not connected | |
| | ADAA0TTIN214 | ADAA0TSEL2.ADAA0T2SEL[14] | not connected | |
| | ADAA0TTIN215 | ADAA0TSEL2.ADAA0T2SEL[15] | not connected | |

a) These input signals are passed through a noise filter, refer to the section *“Port Filters”* in the chapter *“Port Functions”*.

## 22.3 Functional Overview

The A/D Converter (ADAA) converts analog input signals into digital values.

**Features summary**

The ADAA has the following features:

- A/D conversion with 10-bit resolution
- A/D conversion based on successive approximation method
- A/D conversion of up to 16 analog input signals
  The number of channels supported by this microcontroller is specified in the first section of this chapter under the key word *"Channel index m"*.
- A/D conversion of up to 3 differently-prioritized groups of channels
- one-shot and continuous A/D conversion modes (continuous conversion mode for channel group CG0 only)
- channel repeat function (repeats conversion of each channel in a channel group 1 to 4 times)
- software and hardware start trigger modes
  The hardware trigger source can be selected from multiple input signals.
- configurable terms for generation of end interrupts
- three types of result check functions
- discharge function for discharging the capacitor prior to executing conversion with a new sampling value
- self-diagnosis function to check correct ADAA operation

The following figure shows the main components of the ADAA.

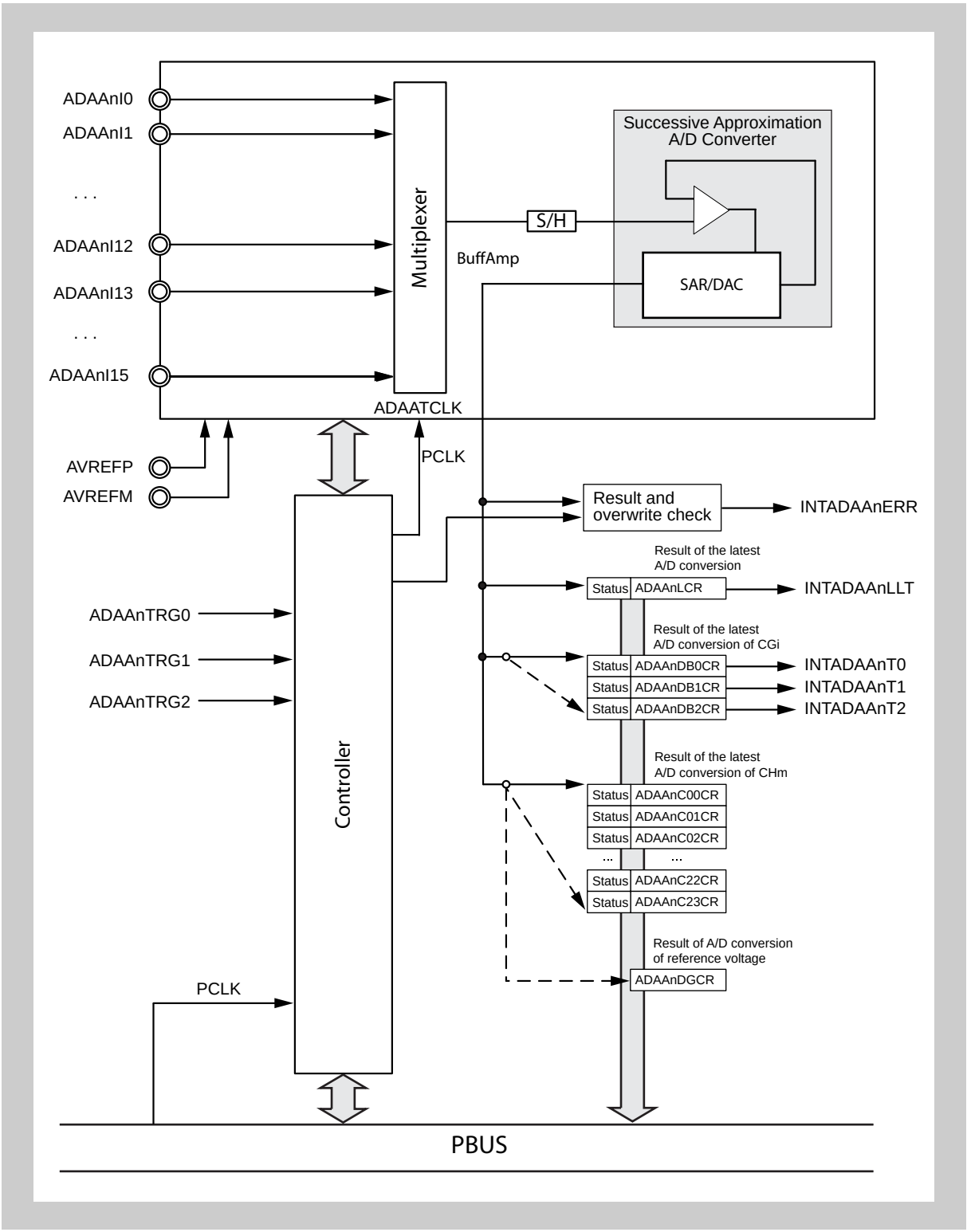



**Figure 22-2 Block diagram of the ADAA**

## 22.4 Cautions

**Caution** Make sure that the voltages input to ADAAnIm do not exceed the rated values. If a voltage higher than AVDD or lower than AVSS is input to a channel, the conversion value of the channel is saturated, and the electrical characteristics of the other channels may also be affected.

For further information concerning the electrical characteristics refer to the Data Sheet.

## 22.5 Functional Description

**A/D Converter resolution**

The A/D Converter A (ADAA) converts up to 16 analog input signals into digital values with a resolution of 10-bit.

**Channels and channel groups**

Each of the input channels can be assigned to one of three channel groups CGi (i = 0 to 2). The list of the input channels assigned to each CG is called a scan list (including diagnostic A/D conversion of CG0). The scan lists can be set up easily with one register. The scan lists can also be changed during operation. All the A/D conversions for a scan list are called scan list conversion.

The ADAA supports up to 3 differently-prioritized channel groups and 2 conversion modes:

- Continuous conversion mode:
  repeats conversion of all input channels of the channel group 0 (CG0) scan list.
- One-shot conversion:
  executes conversion of all input channels of the channel group i (CGi) scan list conversion only once. In one-shot conversion mode, conversion of each channel in the scan list is repeated a selectable number of times (1 to 4 times).

**A/D conversion**

The A/D conversion can be started using either software or hardware as the start trigger.

A multiplexer selects the channel to be converted and the Sample & Hold circuit holds the voltage input.

The A/D Converter uses the successive approximation method.
The successive approximation register (SAR) stores the D/A converter output voltage to be compared with the analog input voltage as a 10-bit digital value.

After each successful conversion the INTADAAnLLT signal is generated.

**A/D conversion result registers**

When the A/D conversion is complete, the contents of the SAR register is stored in three registers allowing the conversion result

- of all channels
- of the last converted channel
- of the last of a certain channel group

to be read continuously.

Depending on the configuration, the ADAA generates a conversion end interrupt after the A/D conversion of certain channels and/or at the end of the A/D conversion of all channels of a channel group.

**Optional result check**

The results of A/D conversions can be checked with the following functions:

- conversion result overwrite check function
- conversion result upper/lower limit compare function

**Optional discharge function**

If required, the internal capacitor of the Sample & Hold circuit can be discharged prior to every conversion.

**Self-diagnosis function**

By means of the A/D conversion circuit self-diagnosis function correct function of the ADAA can be checked.

**Configurable stabilization time**

By setting an arbitrary value to the stabilization counter, the optimum stabilization time can be secured after the power is switched on.

### 22.5.1 Basic Operation

This section describes the basic procedure for an analog to digital conversion. More detailed descriptions are given in the following sections.

1. To optimize the startup time after power on and after standby mode is released, adjust the stabilization time by specifying the stabilization counter ADAAnCNT register.
2. Switch the power of the ADAA on by setting ADAAnCTL1.ADAAnGPS to 1.
3. Before you enable the A/D converter (ADAA must be disabled by ADAAnCTL0.ADAAnCE = 0), configure the ADAA clock, trigger mode, conversion mode, interrupt generation, channel groups, etc. in the following registers:
   – ADAAnCTL1
   – ADAAnCGi
   – ADAAnIOCi
   – ADAAnTSELi
4. If you want to check that the A/D conversion results are within a certain value range, enable the conversion result upper/lower limit compare function for the desired channels (ADAAnCTL2.ADAAnRCKm) and specify the lower and upper limits for the ADAAnLL and ADAAnUL registers.
5. If you want the capacitor of the Sample & Hold circuit to be discharged before sampling a new value, enable the discharge function by writing ADAAnCTL1.ADAAnDISC = 1.
6. Enable the A/D converter by setting ADAAnCTL0.ADAAnCE to 1.
   The A/D converter is ready for A/D conversion after the stabilization time has elapsed after power on or stand-by mode release.
7. Depending on the configured trigger mode, A/D conversion is started by a channel group related start trigger:
   – by a software trigger (ADAAnTRGi.ADAAnSTTi = 1), or
   – by a hardware trigger (input signal ADAAnTTRGi)

   If the A/D conversion starts for multiple CGs, the order of A/D conversion depends on the priority of the CGs.
8. The A/D conversion end interrupt INTADAAnTi is generated when the conversion of the channel set in the ADAAnIOCi register ends.
9. Read the results from the A/D conversion result registers ADAAnLCR, ADAAnDBiCR, and ADAAnCmCR.
10. Monitor the following registers:
    – ADAAnSTR1: To check whether the A/D conversion results have been overwritten before being read.
    – ADAAnSTR0: To check whether the A/D conversion results are within the configured range (only if the conversion result upper/lower limit compare function is enabled).
11. Disable the A/D converter if you want to reconfigure it. To do so, set ADAAnCTL0.ADAAnCE to 0.

### 22.5.2 Clock usage

The ADAA system clock ADAATCLK is the PCLK clock. Sampling and conversion times are selected via ADAAnCTL1.ADAAnFR[3:0].

### 22.5.3 Channels and channel groups

The input channels are configured as channel groups (CG). A scan list can be created for each CG through register settings, and also can be changed during operation. The conversion settings for a CG are applied to all the channels within that group.

The ADAA supports up to three channel groups (CGi, where i = 0 to 2). The channels, i.e. the scan list, of each CGi are specified with the ADAAnCGi register.

**Note** The conversion of a single channel can be performed by assigning only one input channel to a CG.

#### (1) Order of A/D conversion

Upon occurrence of the start trigger for a CG, the channels set to its scan list are converted in ascending order, starting from CH00.

If A/D conversion requests for multiple CGs are pending, the CGs are converted in the following hierarchical order:

CG2 (highest priority) > CG1 > CG0 (lowest priority)

If the start trigger for a CG with a higher priority, or the trigger for ADCHALT mode is set, the current A/D conversion is halted.

One of the following two methods can be selected for halting A/D conversion, according to the setting of ADAAnCTL1.ADAAnTRMi.

- ADAAnCTL1.ADAAnTRMi = 0:
  The A/D conversion of CG is interrupted immediately.

  The A/D conversion of the interrupted channel is repeated after all pending A/D conversions of higher priority CGs have been finished.
- ADAAnCTL1.ADAAnTRMi = 1
  The A/D conversion of the current channel is completed before the higher priority CG is converted.

  When the A/D conversions of *all* higher priority CGs have been finished, the interrupted A/D conversion is continued from the next channel.

ADAAnSTR2.ADAAnST[2:0] indicates the current conversion status.

**Examples** The following figures illustrate the different types of conversion interruption; CH3, CH9, and CH12 are assigned to CG0, CH5 and CH9 are assigned to CG2.

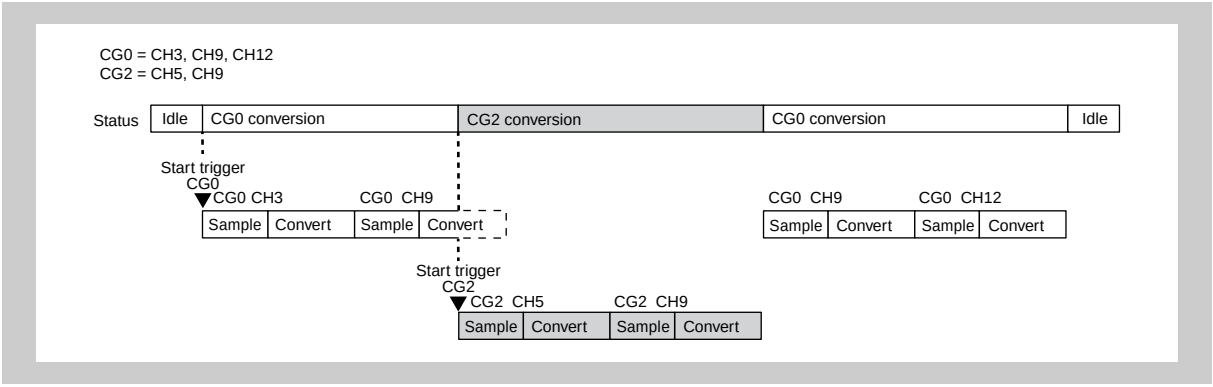



**Figure 22-3 Immediate interruption of A/D conversion of CG0 (ADAAnCTL1.ADAAnTRM0 = 0)**

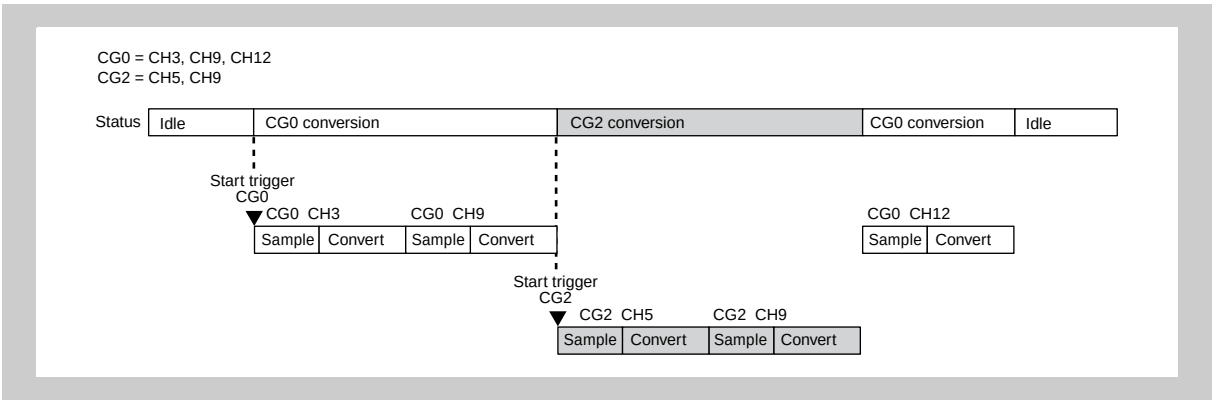



**Figure 22-4 Wait until A/D conversion of current channel has been completed (ADAAnCTL1.ADAAnTRM0 = 1)**

### 22.5.4 A/D conversion modes

The A/D converter provides the following A/D conversion modes.

| Mode | Operation | Channel group |
|---|---|---|
| One-shot conversion mode (ADAAnCTL1.ADAAnMD0 = 0) | Executes CGi scan list conversion only once. The channel of a scan list can be repeatedly converted a specified number of times (1 to 4 times, according to the setting of ADAAnCTL0.SCTi[1:0]). | CG0, CG1, CG2 |
| Continuous conversion mode (ADAAnCTL1.ADAAnMD0 = 1) | Executes scan list conversion repeatedly. | CG0 |

**Notes**
1. A running A/D conversion is interrupted by an A/D conversion request for a higher priority CG and then is automatically continued when all requests of higher priority CGs have been finished (see *1 “Order of A/D conversion” on page 1155* ).
2. CG1 and CG2 operate in one-shot conversion mode regardless of the conversion mode setting in ADAAnCTL1.ADAAnMD0.

#### (1) One-shot conversion mode

In one-shot conversion mode, CGi scan list conversion is executed upon a start trigger. The number of times each channel of a scan list is repeatedly converted, can be specified for each CG from 1 to 4, by specifying ADAAnCTL0.ADAAnSCTi[1:0].

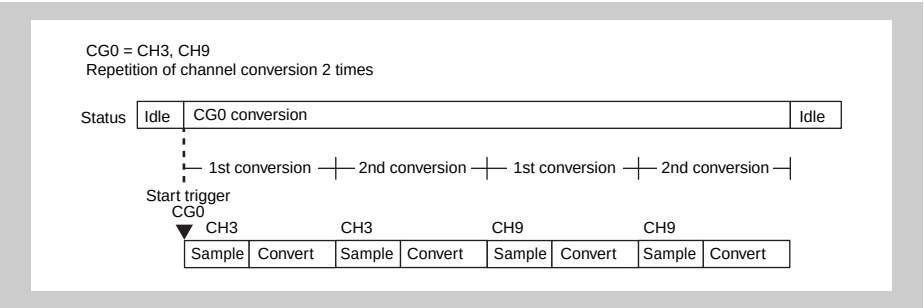



**Figure 22-5 2 times repetition of channel conversion in one-shot conversion mode**

**(a) Start Trigger for the same CG before end of conversion**

The A/D converter can hold one trigger for starting another conversion prior to the end of conversion of the same CG. Therefore, A/D conversion is performed continuously if one or more subsequent start triggers are input (the second and subsequent start triggers are ignored) before the A/D conversion for the CGi started by the first start trigger ends.

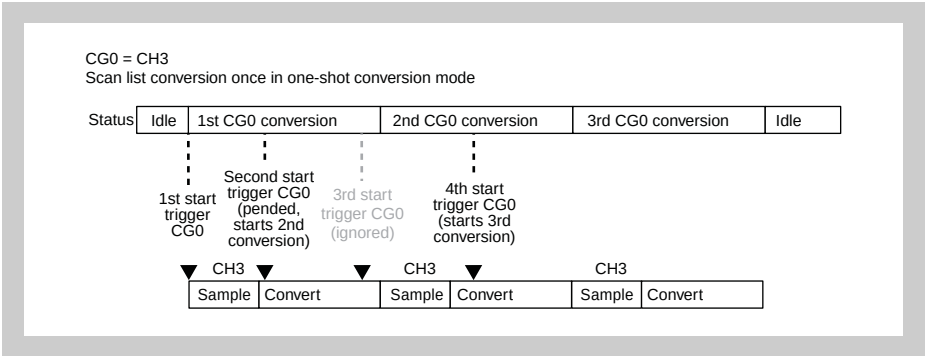



**Figure 22-6 Start Trigger for the same CG before end of conversion**

**(b) Start Trigger for a different CG before end of conversion**

Start triggers for lower priority CGs during conversion of a higher priority CG are ignored, i.e.

- CG0 start trigger is ignored during CG1 or CG2 conversion
- CG1 start trigger is ignored during CG2 conversion

Reversely, a start trigger for a higher priority CG during conversion of a lower priority CG is held.

A start trigger for a lower priority CG that was input before the start of conversion of a high priority CG is held. When no start trigger is generated, a start-before-conversion-end trigger is accepted even during the conversion of a high priority CG.

- CG2 start trigger is held during CG0 or CG1 conversion
- CG1 start trigger is held during CG0 conversion

During conversion, start triggers before end of conversion for a lower priority CG are ignored.
Reversely, a start trigger before end of conversion for a lower priority CG that was input before the start of conversion of a high priority CG is held. When no start trigger is generated, a start-before-conversion-end trigger is accepted even during the conversion of a high priority CG.

Depending on the ADAAnCTL1.ADAAnTRM0 setting, conversion of the higher priority CG is launched immediately ((ADAAnTRM0 = 0) or after completion of the lower priority CG conversion (ADAAnTRM0 = 1).
Refer also to *22.5.3 “Channels and channel groups” on page 1155* .

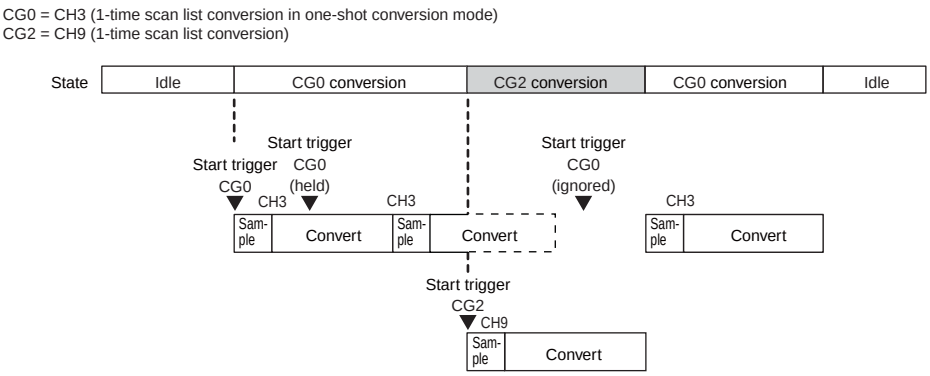



**Figure 22-7 Start Trigger for a different CG before end of conversion (ADAAnCTL1.ADAAnTRM0 = 0)**

### (2) Continuous conversion mode

Continuous conversion mode can be used only for CG0 (ADAAnCTL1.ADAAnMD0 = 1).

In continuous conversion mode, a start trigger causes the channels of CG0 to be sampled and converted repeatedly until a stop trigger is generated or another stop condition occurs (refer to *22.5.6 “Stopping A/D conversion” on page 1162* ).

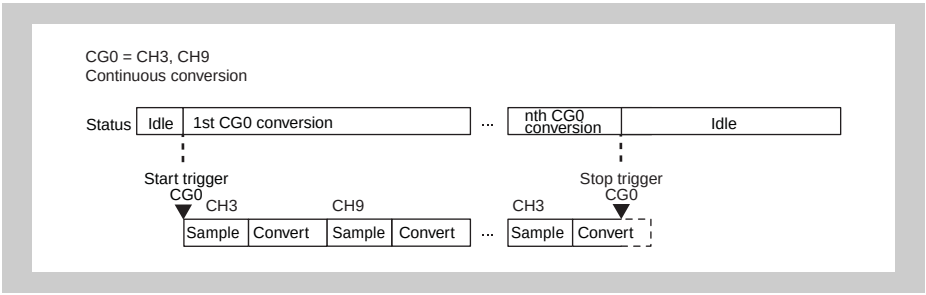



**Figure 22-8 Continuous conversion mode**

**Caution** After a stop trigger, the state becomes idle and sampling/conversion cannot be performed.

**Note** Additional start triggers for CG0 are ignored in continuous conversion mode.

### 22.5.5 Starting A/D conversion (start trigger modes)

A/D conversion can be started by a software or a hardware trigger, as specified by ADAAnCTL1.ADAAnMD1.

If A/D conversion is started for multiple CGs, the conversion order depends on the priority of the CGs (refer to *1 “Order of A/D conversion” on page 1155* ).

**Notes**

1. If no channels are assigned to the CGi scan list (ADAAnCGi = 0000 $0000_H$), start triggers for that CGi are ignored.
2. In one-shot conversion mode, the A/D converter can hold only one start trigger. Additional start triggers are ignored (refer to *Figure 22-6 “Start Trigger for the same CG before end of conversion” on page 1158* ).
3. In continuous conversion mode, additional start triggers before a stop trigger has been generated, are ignored.

#### (1) Software start trigger

The A/D conversion of CGi is started by setting ADAAnTRGi.ADAAnSTTi = 1, provided that the A/D converter is enabled (ADAAnCTL0.ADAAnCE = 1).

**Timing example of software start trigger**

The following figure shows the timing of a software start trigger.

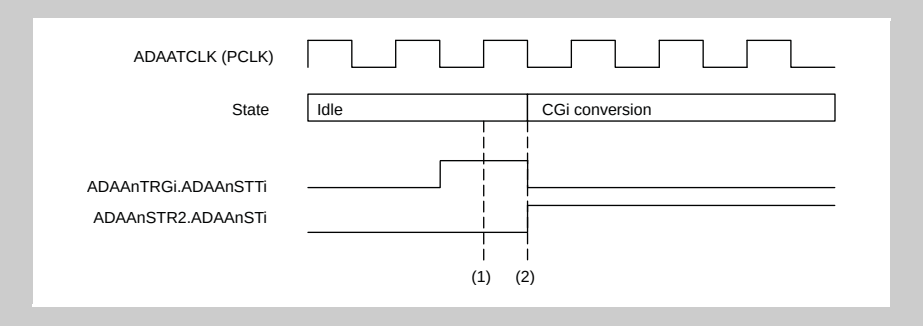



**Figure 22-9 Software start trigger timing example**

1. Software trigger for CGi is written.
2. A/D conversion starts at the next falling edge of ADAATCLK. The status bit ADAAnSTR2.ADAAnSTi is set, indicating that A/D conversion of CGi is running.

**(2) Hardware start trigger**

The A/D conversion of CGi is started upon detection of the valid edge of the ADAAnTTRGi signal, provided that the A/D converter is enabled (ADAAnCTL0.ADAAnCE = 1) and hardware trigger mode is set (ADAAnCTL1.ADAAnMD1 = 1).

The valid edge is specified in ADAAnCTL1.ADAAnTiETS[1:0] for every CG individually.

**H/W trigger expansion**

If this microcontroller supports a hardware trigger expansion, up to 16 hardware trigger sources can be specified for each ADAAnTTRGi signal input. ADAAnTSELi specifies the input signals to be used as the ADAAnTTRGi signal.

**Note** For details about the hardware start trigger function expansion refer to *"H/W trigger expansion"* in the second section of this chapter.

**Timing of hardware start trigger**

The A/D converter starts A/D conversion upon detection of the valid edge of the ADAAnTTRGi signal.

The following figure shows the timing of a hardware start trigger with the following condition:

- Valid edge of ADAAnTTRGi is rising edge (ADAAnCTL1.ADAAnTiETS[1:0] = $01_B$)

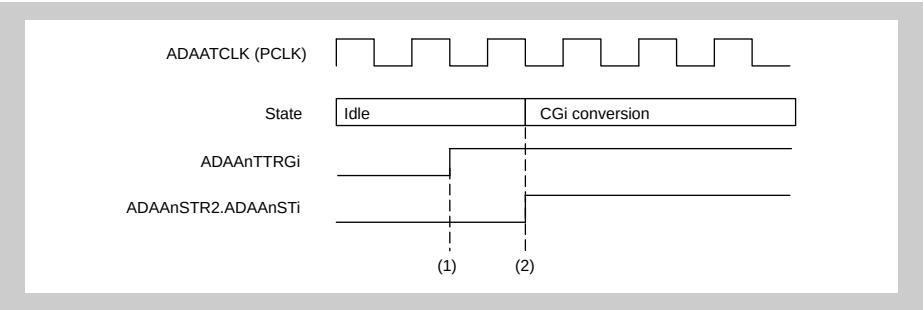



**Figure 22-10 Hardware start trigger timing**

1. Input signal ADAAnTTRGi rises.
2. A/D conversion starts at the next falling edge of ADAATCLK.

### 22.5.6 Stopping A/D conversion

#### (1) Stop trigger

Setting the stop trigger bit for CGi to 1 (ADAAnTRG4+i.ADAAnSPi = 1) stops the A/D conversion for the CGi.

If the stop trigger is generated before the A/D conversion ends, the A/D conversion end interrupt signal INTADAAnTi is not generated and the A/D conversion result register is not updated.

If the start trigger is generated again after the A/D conversion is stopped by a stop trigger, scan list conversion is executed from the beginning.

Follow the stop trigger procedure below when using the hardware start trigger.

1. Stop hardware start trigger generation.
2. Set the stop trigger bit (ADAAnTRG4+i.ADAAnSPi = 1).
3. Check the status of ADAAnSTR2.ADAAnSTi.

If the above procedure is not followed, the hardware start trigger and the stop trigger may conflict with each other and A/D conversion may not stop.

**Timing of stop trigger**

1. Stop trigger for CGi is written.
2. A/D conversion of CGi stops at the next falling edge of ADAATCLK.
   The status bit ADAAnSTR2.ADAAnSTi is cleared, indicating that A/D conversion of CGi is stopped.
   If the digital value of ADAAnIm is already available, the operation is as follows:
   – All A/D conversion result registers are updated.
   – The conversion end interrupt INTADAAnTi is generated as configured in ADAAnIOCi (refer to *22.5.10 “Interrupt generation” on page 1166* ).
   – If configured in ADAAnCTL2, the A/D conversion result is checked to see whether it is in the specified value range (refer to *22.5.12 “Result check functions” on page 1169* ).

   The A/D converter proceeds with pending A/D conversion requests of other CGs - if there are any.

The following figures show the timing of a stop trigger with the following condition:

- The A/D conversion end interrupt INTADAAnTi is generated at the end of the A/D conversion of CGi (ADAAnIOCi = 0000 $0000_H$)

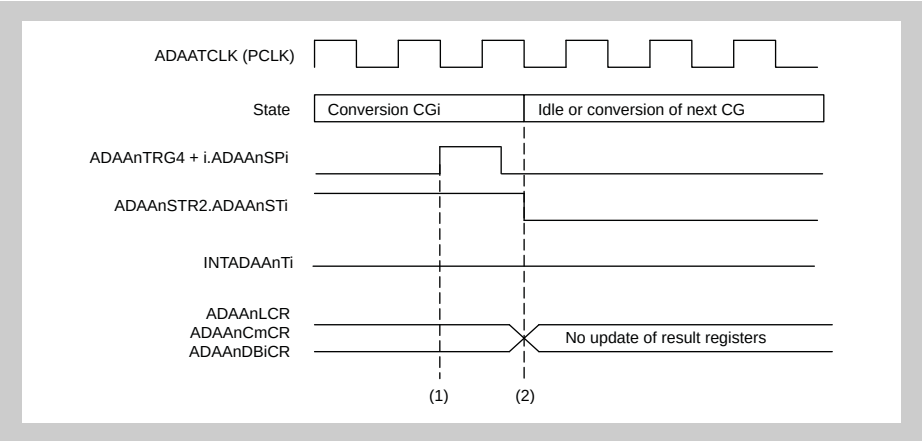



**Figure 22-11 Stop trigger timing, if generated before A/D conversion end**

1. The stop trigger (ADAAnTRG4+i.ADAAnSPi) is set to 1.
2. The status bit (ADAAnSTR2.ADAAnSTi) is cleared.
   The A/D conversion end interrupt INTADAAnTi is not generated and the A/D conversion result register is not updated.

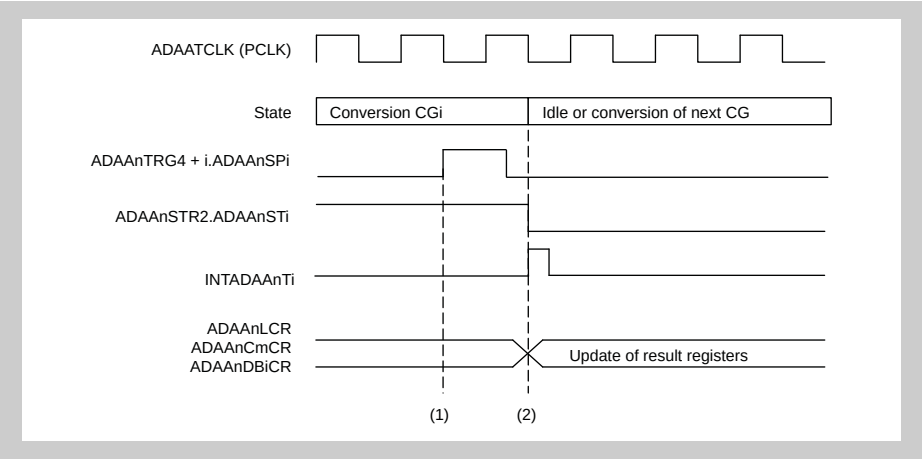



**Figure 22-12 Stop trigger timing, if generated after A/D conversion end**

1. The stop trigger (ADAAnTRG4+i.ADAAnSPi) is set to 1.
2. The status bit (ADAAnSTR2.ADAAnSTi) is cleared.
   The A/D conversion end interrupt (INTADAAnTi) is generated and the A/D conversion result register is updated.

**(2) Other stop conditions**

In addition to the software stop trigger, A/D conversion is stopped when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0).

With ADAAnCTL0.ADAAnCE = 0 conversions are stopped immediately and no A/D conversion results are stored in the conversion result registers.

### 22.5.7 Stand-by mode

Stand-by mode is entered, when the related system stand-by mode becomes active.

The A/D converter is automatically disabled (ADAAnCTL1.ADAAnCE = 0).

To leave stand-by mode:

1. Release related system stand-by mode.
2. Enable the A/D converter by setting ADAAnCTL1.ADAAnCE to 1.

**Note** After stand-by mode is released, a start trigger is accepted but conversion does not start until the stabilization time elapses (stabilization counter ADAAnCNT = $00_H$).
Refer to *22.5.15 “Stabilization control” on page 1173* for details.

### 22.5.8 Pausing and resuming A/D conversion (ADCHALT mode)

The A/D Converter allows the A/D conversion (of all CGs) to be paused/halted. The halted A/D conversion can be continued with a resume trigger.

Procedure:

1. Set ADAAnTRG3 = 1 to go into ADCHALT mode (refer to *1 “Order of A/D conversion” on page 1155* for details about the halt operation).
   – Start triggers are ignored in ADCHALT mode.
   – The internal circuits are stopped and the power consumption is reduced.
   – The analog input pins ADAAnIm can be used for other functions in ADCHALT mode.
2. Set ADAAnTRG7.ADAAnSP3 = 1 to release ADCHALT mode and start A/D conversion.

**Note** ADCHALT has the highest priority and overrules all CGi conversions.

### 22.5.9 Resolution, sampling and conversion times

The total conversion time comprises sampling time and A/D conversion time.

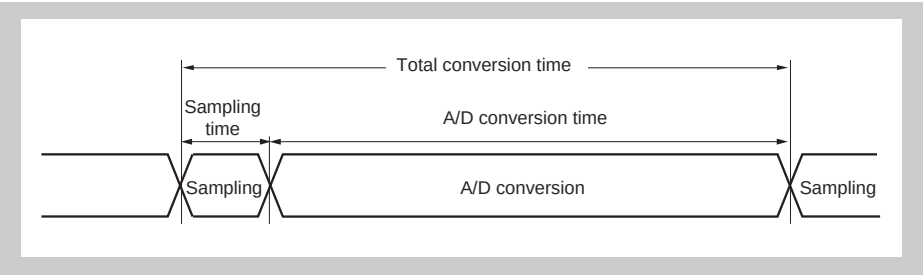



**Figure 22-13 Total conversion time**

The sampling time is the time of connecting an analog input voltage to the Sample & Hold circuit. The A/D conversion time is the time required to obtain one digital value out of an analog input voltage.

While the A/D conversion time takes 25 PCLK clock cycles, the sampling time - and thus the total conversion time - depends on the setting of ADAAnCTL1.ADAAnFR[3:0]:

**Table 22-10 Sampling and conversion times (discharge function disabled)**

| ADAAnCTL1. ADAAnFR[3:0] | A/D conversion time [PCLK clocks] | Sampling time [PCLK clocks] | Total conversion time [PCLK clocks] |
|---|---|---|---|
| $0000_B$ | 25 | 13 | 38 |
| $0001_B$ | 25 | 20 | 45 |
| $0020_B$ | 25 | 26 | 51 |
| $0011_B$ | 25 | 33 | 58 |
| $0100_B$ | 25 | 39 | 64 |
| $0110_B$ | 25 | 52 | 77 |
| $1000_B$ | 25 | 65 | 90 |
| $1010_B$ | 25 | 78 | 103 |
| $1100_B$ | 25 | 91 | 116 |
| $1110_B$ | 25 | 104 | 129 |

**Discharge function**

Using the discharge function (ADAAnCTL1.ADAAnDISC = 1) increases the total conversion time by 7 PCLK cycle.
Refer to *22.5.14 “Discharge function” on page 1173* for details.

## 22.5.10 Interrupt generation

### (1) A/D conversion end interrupt INTADAAnTi

The INTADAAnTi interrupt indicates that the new A/D conversion result is saved to the conversion result register.

An A/D conversion end interrupt is generated upon completion of the A/D conversion for any channel of CGi specified in the ADAAnIOCi register.

If nothing else is configured (ADAAnIOCi = 0000 $0000_H$), INTADAAnTi is generated at the end of the A/D conversion of CGi.

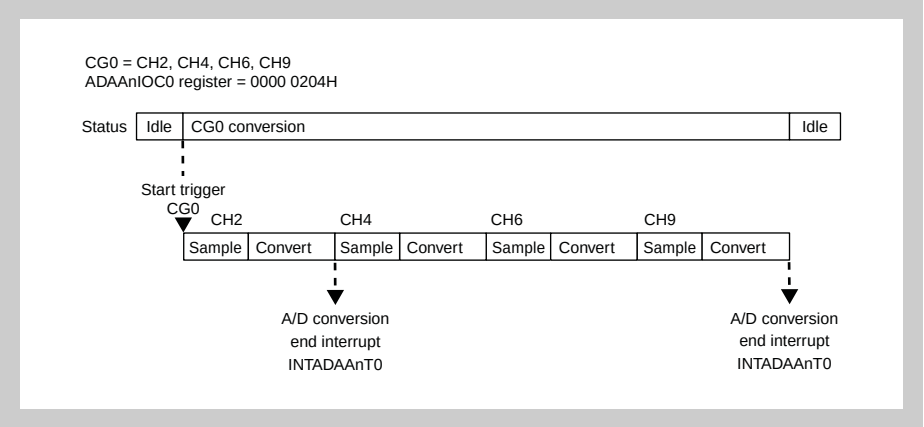



**Figure 22-14 Generation of A/D conversion end interrupt INTADAAnTi**

**Notes**

1. ADAAnIOCi can be written at any time even when the A/D converter is enabled (ADAAnCTL0.ADAAnCE = 1). The new value takes effect after the current A/D conversion of CGi has been completed.
2. ADAAnIOCi is associated with ADAAnCGi and their buffer registers should be updated simultaneously. As the update time depends on writing ADAAnCGi, always write ADAAnIOCi before ADAAnCGi if you want to change the interrupt generation for a CG.

### (2) Error interrupt INTADAAnERR

The interrupt INTADAAnERR is generated in the following cases:

- The result of the A/D conversion of a specified channel is out of the specified range, when the conversion result upper/lower limit compare function is enabled.
- An A/D conversion result in ADAAnLCR, ADAAnDBiCR, or ADAAnCmCR has been overwritten before it was read.
  The generation of the INTADAAnERR error interrupt upon register overwrite can be controlled for each register by setting ADAAnCTL0.ADAAnOEM[4:0].

Refer to *22.5.12 “Result check functions” on page 1169* for details.

### (3) Last conversion interrupt INTADAAnLLT

The interrupt INTADAAnLLT is generated after each successful conversion.

## 22.5.11 Storage of A/D conversion result

### (1) A/D conversion result registers

The A/D conversion result is stored in the following registers:

- ADAAnLCR register
  This register stores the latest A/D conversion result.
- ADAAnDBiCR register
  This register stores the latest A/D conversion result for CGi.
- ADAAnCmCR register
  This register stores the latest A/D conversion results for channel m.

Each register also stores status flags of the A/D conversion result (see *22.5.12 “Result check functions” on page 1169* ).

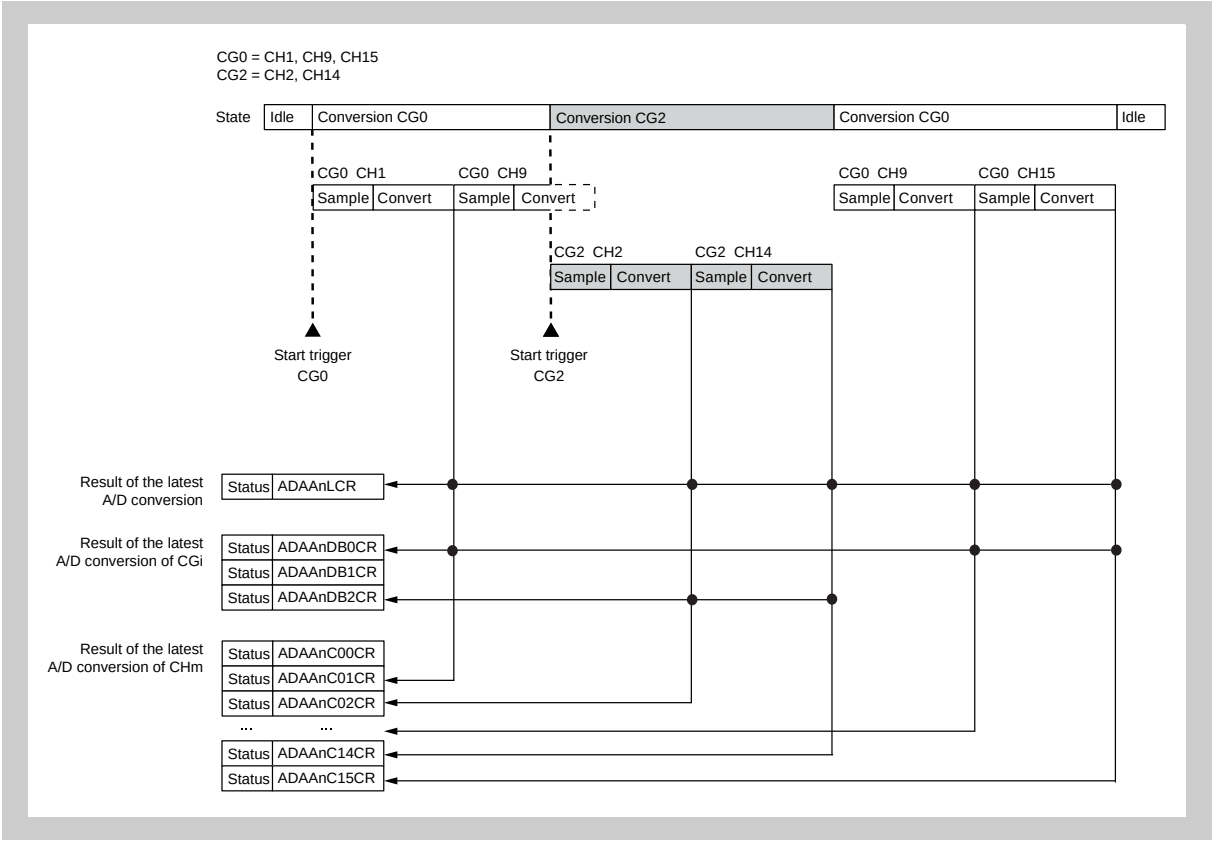



**Figure 22-15 Storage of A/D conversion result**

**(2) Configuration related to A/D conversion result storage**

**Bit position** ADAAnCTL1.ADAAnCRAC specifies whether the 10-bit A/D conversion result is right aligned (ADAAnCRAC = 0) or left aligned (ADAAnCRAC = 1).

**Read & clear function** ADAAnCTL1.ADAAnRCL specifies whether the A/D conversion result ADAAnCmCR is retained after reading it or cleared by reading it.

**(3) Relationship between analog input voltage and A/D conversion result**

The relationship between the analog input voltage input to the analog input pin (ADAAnIm) and the A/D conversion results (the values ADAAnLCR[15:00], ADAAnCmCR[15:00], and ADAAnDBiCR[15:00]) is expressed as follows:

$$\text{A/D conversion result value} = \text{INT}\left(\frac{V_{IAN} - AV_{REFM}}{AV_{REFP} - AV_{REFM}} \times 2^k + 0.5\right)$$

or

$$(\text{A/D conversion result} - 0.5) \times \frac{AV_{REFP} - AV_{REFM}}{2^k} \leq V_{IAN} - AV_{REFMn} < (\text{A/D conversion result} + 0.5) \times \frac{AV_{REFP} - AV_{REFM}}{2^k}$$

| | |
|---|---|
| INT(): | Function that returns the integer part of the value in parentheses |
| $V_{IAN}$: | Analog input voltage |
| $AV_{REFP}$: | AVREFP pin voltage |
| $AV_{REFM}$: | AVREFM pin voltage |
| A/D conversion result: | Values of the ADAAnLCR[15:00], ADAAnCmCR[15:00], and ADAAnDBiCR[15:00] bits |
| k = 10: | Resolution: 10 bit |

The figure below shows the relationship between the analog input voltage and A/D conversion result.

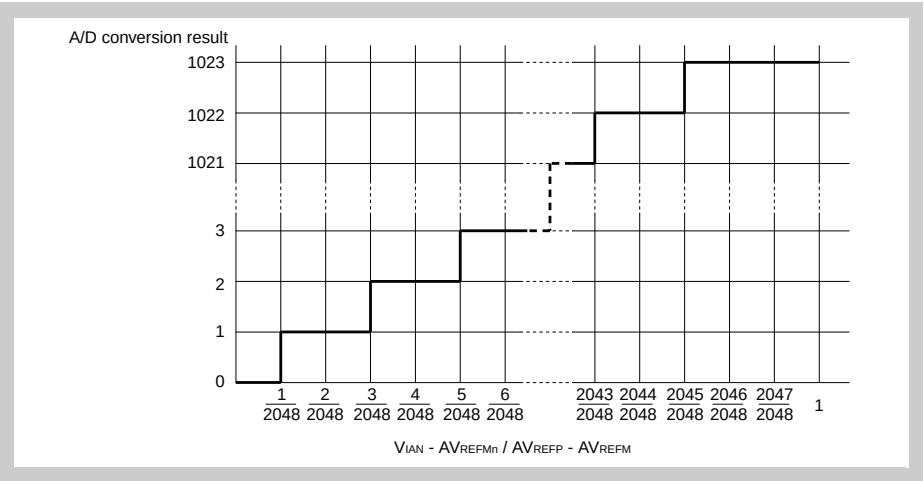



**Figure 22-16 Conversion characteristics of 10-bit A/D converter (ADAAnCTL1.ADAAnCTYP = 1)**

### 22.5.12 Result check functions

ADAA allows checking of the A/D conversion result with the following functions:

- Conversion result overwrite check function
- Conversion result read flag function
- Conversion result upper/lower limit compare function

#### (1) Conversion result overwrite check function

The ADAA allows to check if an A/D conversion result has been overwritten before it was read.

**Error flags**

The A/D conversion result registers have the following overwrite error flags:

- ADAAnLCR.ADAAnLER1
- ADAAnDBiCR.ADAAnDBiER1
- ADAAnCmCR.ADAAnCmER1

Setting of the overwrite error flag indicates, that the conversion result stored in the respective register was overwritten before it is read.

The overwrite flag ADAAnCmCR.ADAAnCmER1 is reflected in ADAAnSTR1.ADAAnOWEm, which holds overwrite flags for all channels.

**Error interrupt**

If the A/D conversion results in the ADAAnLCR, ADAAnDBiCR, and ADAAnCmCR registers are overwritten before they were read, an error interrupt INTADAAnERR is generated.

Generation of an INTADAAnERR interrupt can be masked separately for each for each conversion result register type, so that the interrupt does not become effective:

**Table 22-11 Overwrite error interrupt masking**

| Result register | Mask bit in ADAAnCTL0 |
|---|---|
| ADAAnLCR | ADAAnOEM4 |
| ADAAnDB2CR | ADAAnOME3 |
| ADAAnDB1CR | ADAAnOEM2 |
| ADAAnDB0CR | ADAAnOEM1 |
| ADAAnCmCR | ADAAnOEM0 |

If a mask bit is set to 1, the interrupt is not effective.

**Note** Interrupt generation for conversion result registers that are not to be read shall be masked by setting the appropriate ADAAnCTL0.ADAAnOEM[4:0] = 1.

### (2) Conversion result read flag function

An update status flag indicates whether the A/D conversion result in the conversion result register has already been read or is a new one.

**Status flags**

The update status flags are provided in the A/D conversion result registers:

- ADAAnLCR.ADAAnLUR
- ADAAnDBiCR.ADAAnDBiUR
- ADAAnCmCR.ADAAnCmUR

If an update status flag is set to 1, the A/D conversion result is new.

The update status flags are cleared after they are read.

### (3) Conversion result upper/lower limit compare function

The ADAA can check whether an A/D conversion results lies within a configurable value range.

The result check function can be enabled for every channel individually in ADAAnCTL2.

When enabled, the A/D conversion result ADAAnCmCR is compared with the specified lower limit (ADAAnLL) and upper limit (ADAAnUL), each time the channel results is converted respectively updated.

**Error flags**

If the A/D conversion result of a specified channel is either lower than the lower limit value ADAAnLL or higher than the upper limit value ADAAnUL, the ADAAnSTR0.ADAAnRCE[15:00] error flag corresponding to that channel is set.

ADAAnSTR0 indicates the error status of the latest A/D conversion upper/lower limit check for every channel. By use of this register it is possible to evaluating which A/D conversion results are outside the specified range.

The result check error flag is also mirrored by the A/D conversion result registers ADAAnLCR, ADAAnDBiCR, and ADAAnCmCR.

The result check error flags ADAAnSTR0.ADAAnRCE[15:00] are reflected in ADAAnCmCR.ADAAnCmER0 of the respective channel.

**Error interrupt**

If the A/D conversion result for the specified channel is out of the setting range, the INTADAAnERR error interrupt is generated.

### 22.5.13 Self-diagnosis function

The self-diagnosis function can be used to verify that the ADAA works properly.

The diagnosis of the A/D conversion circuit can be performed during normal A/D conversion. Following the end of A/D conversion of CG0, the reference voltage signal ADDIAGOUT is converted. If this diagnosis A/D conversion result differs greatly from the expected value, a hardware anomaly or a malfunction may occur.

The diagnostic A/D conversion is enabled by ADAAnCG0.ADAAnDIAG = 1.

**Note** The A/D conversion circuit diagnosis is available for CG0 only.

The diagnostic A/D conversion is started after the A/D conversion of the last channel of CG0 has been completed:

- The A/D conversion results of CG0 are stored in the “normal” A/D conversion result registers (refer to *1 “A/D conversion result registers” on page 1167* ).
- The result of the diagnostic A/D conversion is stored in ADAAnDGCR.

**Diagnosis procedure**

1. Switch the power of the ADAA on by setting ADAAnCTL1.ADAAnGPS = 1.
2. Configure CG0 and the A/D conversion as follows:
   – Ensure that ADAAnCG0.ADAAnDIAG is set to 1 to enable the diagnostic A/D conversion of the reference voltage.
   For example, write 8000 $0007_H$ to first convert the analog input voltages of CH0, CH1, and CH2, and than the reference voltage ADDIAGOUT for diagnostic purposes.
   – Set ADAAnI0C0.ADAAnCG0IDG = 1 to generate the A/D conversion end interrupt INTADAAnT0 on finishing the diagnostic A/D conversion.
3. Specify the reference voltage signal ADDIAGOUT via ADAAnDGCTL0.ADAAnPSEL.
   For example, set ADAAnDGCTL0.ADAAnPSEL = 1 to apply AVDD as the reference voltage ADDIAGOUT.
4. Enable the ADAA by setting ADAAnCTL0.ADAAnCE = 1.
5. Generate software or hardware start triggers to start the A/D conversion.
6. When the A/D conversion end interrupt INTADAAnT0 is generated, read the A/D conversion results of the diagnostic A/D conversion in ADAAnDGCR.

**ADDIAGOUT change** ADAAnDGCTL0.ADAAnPSEL can be written - and thus the reference voltage can be changed - even during A/D conversion, as shown in the diagram below.

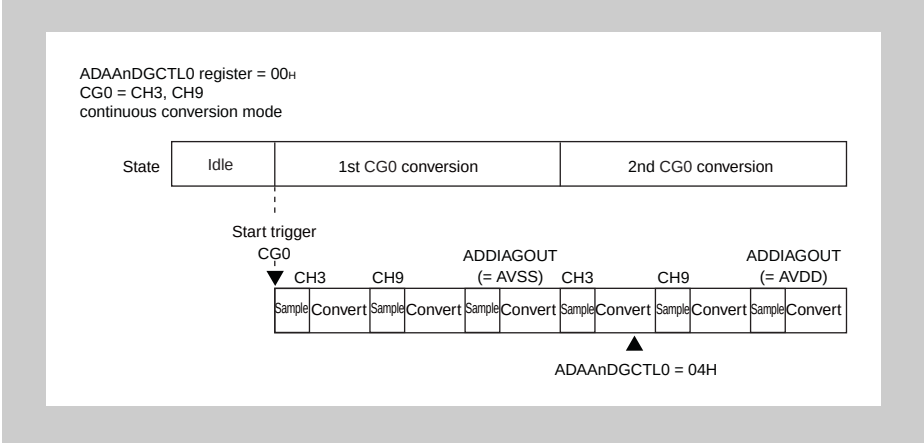



**Figure 22-17 ADDIAGOUT change during A/D conversion**

The value set with ADAAnDGCTL0.ADAAnPSEL is applied upon completion of the conversion of the current channel. Therefore, set the reference voltage of the next diagnosis A/D conversion before that diagnosis A/D conversion starts.

### 22.5.14 Discharge function

If required, the internal capacitor of the common Sample & Hold circuit can be discharged prior to every conversion. This ensures that the capacitor is always empty before the new sample value is stored.

**Note** Using the discharge function increases the total conversion time by 7 PCLK clock cycles (see *22.5.9 “Resolution, sampling and conversion times” on page 1165* ).

**Configuration** The discharge function is enabled by setting ADAAnCTL1.ADAAnDISC to 1.

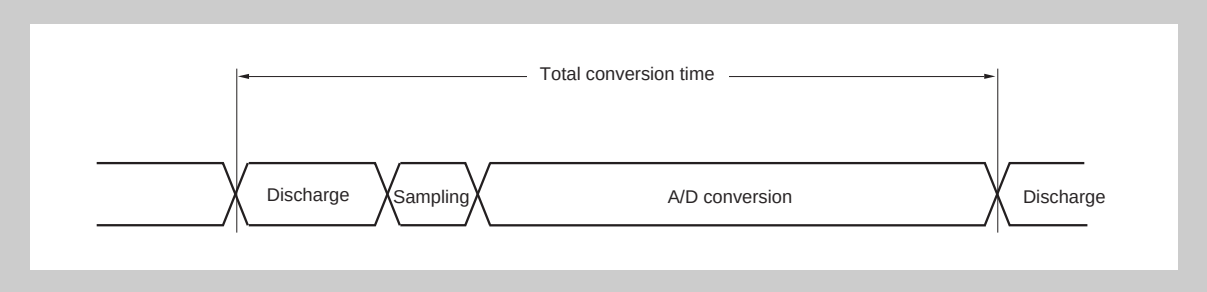



**Figure 22-18 Timing when discharge function is enabled**

### 22.5.15 Stabilization control

The A/D converter needs time for stabilization purposes in the following cases:

- The A/D converter has been switched on (ADAAnCTL1.ADAAnGPS = 1)
- The stand-by mode has been terminated.

A start trigger is accepted during the stabilization time, but the conversion does not start before the stabilization time has elapsed.

In order to secure the minimum stabilization time, the stabilization time counter ADAAnCNT must be set (refer to the Data Sheet).

## 22.6 Registers

This section contains a description of all the registers of the ADAA.

### 22.6.1 ADAA registers overview

The ADAA is controlled and operated by the registers in the following table.

- Where there is one register per channel, this is indicated by an “m”. The number “m” of available channels are specified in the first section of this chapter under the key word “Channel index m”.
- Where there is one register per CG, this is indicated by an “i” (i = 0 to 2).

**Table 22-12 ADAA registers overview (1/2)**

| Register name | Shortcut | Address |
|---|---|---|
| **Control registers** | | |
| A/D converter mode control register 0 | ADAAnCTL0 | <ADAAn_base> + $100_H$ |
| A/D converter mode control register 1 | ADAAnCTL1 | <ADAAn_base> + $104_H$ |
| A/D converter CG register i | ADAAnCGi | <ADAAn_base> + i x $4_H$ |
| A/D converter interrupt control register i | ADAAnIOCi | <ADAAn_base> + $C_H$ + i × $4_H$ |
| A/D converter trigger select control register i | ADAAnTSELi | <ADAAn_base> + $108_H$ + i × $4_H$ |
| A/D converter stabilization counter | ADAAnCNT | <ADAAn_base> + $114_H$ |
| **Conversion status registers** | | |
| A/D converter overwrite error flag register | ADAAnSTR1 | <ADAAn_base> + $28_H$ |
| ADAAnSTR1 flag clear register | ADAAnSTC1 | <ADAAn_base> + $34_H$ |
| A/D converter status flag register 2 | ADAAnSTR2 | <ADAAn_base> + $2C_H$ |
| ADAAnSTR2 flag clear register | ADAAnSTC2 | <ADAAn_base> + $38_H$ |
| **S/W trigger registers** | | |
| A/D converter S/W trigger register i | ADAAnTRGi | <ADAAn_base> + $A4_H$ + i × $4_H$ |
| A/D converter S/W trigger register 3 | ADAAnTRG3 | <ADAAn_base> + $B0_H$ |
| A/D converter S/W trigger register 4+i | ADAAnTRG4 + i | <ADAAn_base> + $B4_H$ + i × $4_H$ |
| A/D converter S/W trigger register 7 | ADAAnTRG7 | <ADAAn_base> + $C0_H$ |
| **A/D conversion result registers** | | |
| A/D converter latest conversion result register | ADAAnLCR | <ADAAn_base> + $A0_H$ |
| A/D converter conversion result register m | ADAAnCmCR | <ADAAn_base> + $3C_H$ + m × $4_H$ |
| A/D converter CGi buffer register i | ADAAnDBiCR | <ADAAn_base> + $C4_H$ + i × $4_H$ |
| A/D converter CGi buffer result register i | ADAAnDBiCRL | <ADAAn_base> + $D0_H$ + i × $4_H$ |
| Diagnostic conversion result register | ADAAnDGCR | <ADAAn_base> + $9C_H$ |
| **A/D conversion result upper/lower limit comparison registers** | | |
| A/D converter result check register | ADAAnCTL2 | <ADAAn_base> + $18_H$ |
| A/D converter result check (upper limit) | ADAAnUL | <ADAAn_base> + $1C_H$ |
| A/D converter result check (lower limit) | ADAAnLL | <ADAAn_base> + $20_H$ |
| A/D converter result check error flag | ADAAnSTR0 | <ADAAn_base> + $24_H$ |
| ADAAnSTR0 flag clear register | ADAAnSTC0 | <ADAAn_base> + $30_H$ |

**Table 22-12 ADAA registers overview (2/2)**

| Register name | Shortcut | Address |
|---|---|---|
| **Diagnose functions control registers** | | |
| Self-diagnosis function control register 0 | ADAAnDGCTL0 | <ADAAn_base> + $DC_H$ |
| **Emulation register** | | |
| Emulation register | ADAAnEMU | <ADAAn_base> + $128_H$ |

**<ADAAn_base>** The base addresses <ADAAn_base> of the ADAAn is defined in the first section of this chapter under the key word “Register addresses”.

## 22.6.2 Control registers details

### (1) ADAAnCTL0 – A/D converter mode control register 0

This register enables/disables the A/D converter. Additionally, it specifies the number of repetitions for one-shot conversion mode, and also whether to generate an error interrupt request if the A/D conversion result is overwritten before it is read.

**Access** This register can be read/written in16-bit units.

**Address** <ADAAn_base> + $100_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | ADAAn OEM4 | ADAAn OEM3 | ADAAn OEM2 | ADAAn OEM1 | ADAAn OEM0 | ADAAn CE | 0 | ADAAn SCT2[1:0] | | ADAAn SCT1[1:0] | | ADAAn SCT0[1:0] | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-13 ADAAnCTL0 register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 12 | ADAAn OEM4 | Specifies whether to generate the INTADAAnERR error interrupt when an A/D conversion result is overwritten in ADAAnLCR before it was read.<br>0: generate INTADAAnERR if the A/D conversion result is overwritten<br>1: do not generate INTADAAnERR<br>Refer to *22.5.12 "Result check functions" on page 1169* for details. |
| 11 | ADAAn OEM3 | Specifies whether to generate the INTADAAnERR error interrupt when an A/D conversion result is overwritten in ADAAnDB2CR before it was read.<br>0: generate INTADAAnERR if the A/D conversion result is overwritten<br>1: do not generate INTADAAnERR<br>Refer to *22.5.12 "Result check functions" on page 1169* for details. |
| 10 | ADAAn OEM2 | Specifies whether to generate the INTADAAnERR error interrupt when an A/D conversion result is overwritten in ADAAnDB1CR before it was read.<br>0: generate INTADAAnERR if the A/D conversion result is overwritten<br>1: do not generate INTADAAnERR<br>Refer to *22.5.12 "Result check functions" on page 1169* for details. |
| 9 | ADAAn OEM1 | Specifies whether to generate the INTADAAnERR error interrupt when an A/D conversion result is overwritten in ADAAnDB0CR before it was read.<br>0: generate INTADAAnERR if the A/D conversion result is overwritten<br>1: do not generate INTADAAnERR<br>Refer to *22.5.12 "Result check functions" on page 1169* for details. |
| 8 | ADAAn OEM0 | Specifies whether to generate the INTADAAnERR error interrupt when an A/D conversion result is overwritten in ADAAnCmCR before it was read.<br>0: generate INTADAAnERR if the A/D conversion result is overwritten<br>1: do not generate INTADAAnERR<br>Refer to *22.5.12 "Result check functions" on page 1169* for details. |
| 7 | ADAAn CE | This bit enables/disables the A/D converter.<br>0: disable A/D converter<br>1: enable A/D converter<br>Note that A/D conversion only starts on hardware trigger or software trigger, when ADAAnCTL0.ADAAnCE = 1.<br>Note also that the A/D converter needs time to stabilize after it has been enabled. A start trigger is accepted even immediately after power-on. After the values of the stabilization counter ADAAnCNT changes to $00_H$, the A/D conversion starts. |

**Table 22-13 ADAAnCTL0 register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>5 to 4</td><td>ADAAn SCT2[1:0]</td><td>Number of conversion repetitions of each channel of CG2 in one-shot conversion mode.
<table>
<tr><th>ADAAn SCT21</th><th>ADAAn SCT20</th><th>Number of conversion repetitions of each channel of CG2</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>2</td></tr>
<tr><td>1</td><td>0</td><td>3</td></tr>
<tr><td>1</td><td>1</td><td>4</td></tr>
</table></td></tr>
<tr><td>3 to 2</td><td>ADAAn SCT1[1:0]</td><td>Number of conversion repetitions of each channel of CG1 in one-shot conversion mode.
<table>
<tr><th>ADAAn SCT11</th><th>ADAAn SCT10</th><th>Number of conversion repetitions of each channel of CG1</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>2</td></tr>
<tr><td>1</td><td>0</td><td>3</td></tr>
<tr><td>1</td><td>1</td><td>4</td></tr>
</table></td></tr>
<tr><td>1 to 0</td><td>ADAAn SCT0[1:0]</td><td>Number of conversion repetitions of each channel of CG0 in one-shot conversion mode.
<table>
<tr><th>ADAAn SCT01</th><th>ADAAn SCT00</th><th>Number of conversion repetitions of each channel of CG0</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>2</td></tr>
<tr><td>1</td><td>0</td><td>3</td></tr>
<tr><td>1</td><td>1</td><td>4</td></tr>
</table></td></tr>
</table>

### (2) ADAAnCTL1 – A/D converter mode control register 1

This register specifies the conversion mode and controls the conversion operations.

**Access** This register can be read/written in 32-bit units.

**Address** <ADAAn_base> + $104_H$

**Initial Value** $0100\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ADAAn T2ETS0 | 0 | ADAAn T1ETS0 | 0 | ADAAn T0ETS0 | 0 | ADAAn CRAC | 0 | 0 | ADAAn MD1 | ADAAn MD0 | 0 | 0 | ADAAn DISC | ADAAn RCL |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | ADAAn FR[3:0] | | | | 0 | ADAAn TRM2 | ADAAn TRM1 | ADAAn TRM0 | 0 | 0 | 0 | ADAAn GPS |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-14 ADAAnCTL1 register contents (1/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 31 | ADAAn T2ETS0 | Specifies the valid edge of the H/W trigger ADAAnTTRGi.<br>0: no edge selected (no trigger accepted)<br>1: rising edge selected<br>For further information refer to the section *"H/W trigger expansion"* in this chapter. |
| 29 | ADAAn T1ETS0 | |
| 27 | ADAAn T0ETS0 | |
| 24 | ADAAn CRAC | Specifies the alignment of the A/D and diagnosis conversion result:<br>0: right-aligned<br>1: left-aligned |
| 21 | ADAAn MD1 | Specifies the A/D conversion start trigger for all CGs.<br>0: software trigger<br>1: hardware trigger and software trigger<br>This configuration is valid for all CGs.<br>The A/D converter only detects triggers when the A/D converter is enabled.<br>Refer to *22.5.5 "Starting A/D conversion (start trigger modes)" on page 1160* for details. |
| 20 | ADAAn MD0 | Specifies the A/D conversion mode for CG0:<br>0: one-shot conversion mode<br>Number of repetitions of scan list conversion is configured in ADAAnCTL0.ADAAnSCTi[1:0] for each CG individually.<br>1: continuous conversion mode<br>This configuration applies to the A/D conversion of CG0 only.<br>CG1 and CG2 are always operated in one-shot conversion mode.<br>Refer to *22.5.4 "A/D conversion modes" on page 1157* for details. |
| 17 | ADAAn DISC | Enables/disables the discharge function:<br>0: disable<br>1: enable<br>Refer to *22.5.14 "Discharge function" on page 1173* for details. |
| 16 | ADAAn RCL | Specifies whether the A/D conversion results ADAAnCmCR and ADAAnDBiCR are retained after they are read.<br>0: A/D conversion results are retained until they are overwritten by the next A/D conversion results<br>1: A/D conversion results are cleared after they are read |

**Table 22-14 ADAAnCTL1 register contents (2/3)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>11 to 8</td><td>ADAAn FR[3:0]</td><td>Specifies the ADAA total conversion time.<br>Refer to the section 22.5.9 “Resolution, sampling and conversion times” for details.
<table>
<tr><th>ADAAnFR[3:0]</th><th>Total conversion time [PCLK clocks] (discharge disabled)</th></tr>
<tr><td>$0000_B$</td><td>38</td></tr>
<tr><td>$0001_B$</td><td>45</td></tr>
<tr><td>$0010_B$</td><td>51</td></tr>
<tr><td>$0011_B$</td><td>58</td></tr>
<tr><td>$0100_B$</td><td>64</td></tr>
<tr><td>$0110_B$</td><td>77</td></tr>
<tr><td>$1000_B$</td><td>90</td></tr>
<tr><td>$1010_B$</td><td>103</td></tr>
<tr><td>$1100_B$</td><td>116</td></tr>
<tr><td>$1110_B$</td><td>129</td></tr>
<tr><td>all others</td><td>setting prohibited</td></tr>
</table>
Note:<br>Enabling the discharge function adds 7 PCLK clock cycles to the total conversion time.</td></tr>
<tr><td>6</td><td>ADAAn TRM2</td><td>Specifies the behaviour of CG2 conversion when transition to ADCHALT mode is requested.[a]<br>0: Halt the current A/D conversion of CG2 immediately, and enter ADCHALT mode.<br>1: Finish conversion of the currently converted channel of CG2, halt A/D conversion of CG2, and enter ADCHALT mode.<br>A/D conversion of CG2 is continued as soon as ADCHALT mode has been terminated.</td></tr>
</table>

**Table 22-14 ADAAnCTL1 register contents (3/3)**

| Bit position | Bit name | Function |
|---|---|---|
| 5 | ADAAn TRM1 | Specifies the behaviour of CG1 conversion when a start trigger for a CG2 A/D conversion occurs or when transition to ADCHALT mode is requested.[a]<br>0: Halt the current A/D conversion of CG1 immediately, and start the A/D conversion of CG2 or enter ADCHALT mode respectively.<br>1: Finish conversion of the currently converted channel of CG1, halt A/D conversion of CG1, and start the A/D conversion of CG2 or enter ADCHALT mode respectively.<br>A/D conversion of CG1 is continued as soon as all pending A/D conversions of CG2 have been completed or the ADCHALT mode has been terminated. |
| 4 | ADAAn TRM0 | Specifies the behaviour of CG0 conversion when a start trigger for a CG1 or CG2 A/D conversion occurs or when transition to ADCHALT mode is requested.[a]<br>0: Halt the current A/D conversion of CG0 immediately, and start the A/D conversion of CG2 or CG1 or enter ADCHALT mode respectively.<br>1: Finish conversion of the currently converted channel of CG0, halt A/D conversion of CG0, and start the A/D conversion of CG2 or CG1 or enter ADCHALT mode respectively.<br>A/D conversion of CG0 is continued as soon as all pending A/D conversions of CG2 or CG1 have been completed or the ADCHALT mode has been terminated. |
| 0 | ADAAn GPS | Switches the power of the ADAA off/on:<br>0: power off<br>1: power on<br>The A/D converter needs a stabilization time after A/D power on (see *22.5.15 “Stabilization control” on page 1173* ). |

a) The behaviour follows the priority scheme: ADCHALT > CG2 > CG1 > CG0. Refer to 1 “Order of A/D conversion” on page 7 for details.

### (3) ADAAnCGi – A/D converter channel group register i

This register creates the scan list for each CG. The channels set to the scan list of CGi are converted in ascending order, i.e. starting from the lowest channel number.
For details, refer to *22.5.3 “Channels and channel groups” on page 1155* .

**Diagnosis conversion**

Additionally the diagnosis A/D conversion channel can be added to the CG0 scan list via the ADAAnCG0 register.
For details, refer to *22.5.13 “Self-diagnosis function” on page 1171* .

**Access**

This register can be read/written in 32-bit units.
It can be written at any time even when the A/D converter is enabled (ADAAnCTL0.ADAAnCE = 1). The new value takes effect after the current A/D conversion of CGi has been completed.

- When CGi is not currently undergoing A/D conversion, the new value becomes effective immediately.
- When CGi is currently undergoing A/D conversion, the new value becomes effective upon completion of the scan list conversion of CGi currently being executed.
- When the stop trigger bit (ADAAnTRG4+i.ADAAnSPi) of CGi is set, the new value becomes effective when A/D conversion is stopped.

**Address**

<ADAAn_base> + i x $4_H$

**Initial Value**

$0000\ 0000_H$

ADAAnCG0:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAn DIAG | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCG0S[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

ADAAnCG1:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCG1S[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

ADAAnCG2:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCG2S[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-15 ADAAnCGi register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 | ADAAn DIAG | Enables/disables the diagnostic A/D conversion of the reference voltage ADDIAGOUT at the end of the A/D conversion of CG0:<br>0: disable A/D conversion of ADDIAGOUT<br>1: convert ADDIAGOUT<br>This bit is only available in ADAAnCG0.<br>Refer to *22.5.13 "Self-diagnosis function" on page 1171* for details. |
| 15 to 00 | ADAAn CGiS[15:00] | Specifies the analog inputs to be converted for CGi:<br>0: do not convert analog input ADAAnIm<br>1: convert analog input ADAAnIm |

**Note** It is possible to assign a channel to multiple channel groups.

### (4) ADAAnIOCi – A/D converter interrupt control register i

The A/D conversion end interrupt INTADAAnTi can be generated when the A/D conversion of a certain channel has been completed.

This register specifies the channel in each CGi scan list, for which INTADAAnTi is generated on A/D conversion completion.

If ADAAnIOCi = 0000 $0000_H$, INTADAAnTi is automatically generated at the completion of A/D conversion of CGi scan list.

**Access** This register can be read/written in 32-bit units.
The new value takes effect after the current A/D conversion of CGi has been completed.

**Address** <ADAAn_base> + $0C_H$ + i × $4_H$

**Initial Value** 0000 $0000_H$

ADAAnIOC0:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCG0IDG | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCG0I[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

ADAAnIOC1:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCG1I[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

ADAAnIOC2:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCG2I[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-16 ADAAnIOCi register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 31 | ADAAn CG0IDG | Specifies whether INTADAAnT0 is generated on completion of the A/D conversion of the reference voltage when the diagnostic mode is enabled for CG0 (ADAAnCG0.ADAAnDIAG = 1):<br>0: do not generate INTADAAnT0<br>1: generate INTADAAnT0<br>This bit is only available in ADAAnIOC0.<br>Refer to *22.5.13 "Self-diagnosis function" on page 1171* for details. |
| 15 to 00 | ADAAn CGiI[15:00] | Specifies whether the interrupt INTADAAnTi is generated on A/D conversion completion of channel m:<br>0: Do not generate INTADAAnTi<br>1: Generate INTADAAnTi<br>**Note:**<br>ADAAnCGiI bits, which are assigned to not available channels, must be set to 0. |

**Note** As the ADAAnIOCi register is associated with the ADAAnCGi register, their buffer registers must be updated simultaneously. As the update time depends on writing ADAAnCGi, always write ADAAnIOCi before ADAAnCGi if you want to change the interrupt generation for a CG.

### (5) ADAAnCNT – A/D converter stabilization counter

This register specifies the stabilization time.

**Access** This register can be read/written in 8-bit units.

**Address** <ADAAn_base> + $114_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ADAAnCNT[7:0] | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-17 ADAAnCNT register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 to 0 | ADAAn CNT[7:0] | Specifies the stabilization counter:<br>Stabilization time = ADAAnCNT[7:0] * PCLK clock cycles |

### (6) ADAAnTSELi – A/D converter trigger select control register i

This register specifies the input signals to be used in combination with hardware start trigger signals ADAAnTTRGi.

**Access** This register can be read/written in 16-bit units.
It can only be written when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0).

**Address** <ADAAn_base> + $108_H$ + i × $4_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnTiSEL[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-18 ADAAnTSELi register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | ADAAn TiSEL[15:00] | Specifies whether the corresponding input signal is to be used as hardware start trigger<br>0: do not use as hardware start trigger<br>1: use as hardware start trigger |

**Note** For details about the hardware start trigger function expansion refer to “H/W trigger expansion” in the first section of this chapter.

### 22.6.3 Conversion status registers

#### (1) ADAAnSTR1 – A/D converter overwrite error flag register

This register indicates for every channel whether the latest A/D conversion result has been overwritten before it was read.

**Access** This register can be read in 32-bit units.

**Address** <ADAAn_base> + $28_H$

**Initial Value** $0000\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnOWE[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 22-19 ADAAnSTR1 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | ADAAn OWE[15:00] | Indicates whether the A/D conversion result of channel m has been overwritten before it was read:<br>0: not overwritten<br>1: overwritten<br>This error flag is cleared by setting ADAAnSTR1.ADAAnOWECm to 1.<br>All flags are also cleared by enabling and disabling the ADAA (ADAAn.ADAAnCE= 0 and then ADAAn.ADAAnCE= 1). |

**Note** ADAAnSTR1.ADAAnOWEm is mirrored by the following overwrite error flags:

- Error flag in register with the latest A/D conversion result of CHm (ADAAnCmCR.ADAAnCmER1)

### (2) ADAAnSTC1 – ADAAnSTR1 flag clear register

This register is the clear control register of ADAAnSTR1.

**Access** This register can be written in 32-bit units.

**Address** <ADAAn_base> + $34_H$

**Initial Value** Reading this register returns always 0000 $0000_{H.}$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnOWEC[15:00] | | | | | | | | | | | | | | | |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 22-20 ADAAnSTC1 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | ADAAn OWEC[15:00] | Clear overwrite error flags in ADAAnSTR1:<br>0: no function<br>1: clears the corresponding ADAAnSTR1.ADAAnOWEm |

### (3) ADAAnSTR2 – A/D converter status flag register 2

This register indicates the current conversion status.

**Access** This register can be read in 16-bit units.

**Address** <ADAAn_base> + $2C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | ADAAn RQT3 | ADAAn RQ2 | ADAAn RQ1 | ADAAn RQ0 | 0 | 0 | 0 | 0 | ADAAn ST3 | ADAAn ST2 | ADAAn ST1 | ADAAn ST0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 22-21 ADAAnSTR2 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 11 | ADAAn RQ3 | Indicates whether a ADCHALT request is pending:<br>0: ADCHALT request is not pending<br>1: ADCHALT request is pending |
| 10 | ADAAn RQ2 | Indicates whether A/D conversion request for CG2 is pending:<br>0: A/D conversion request for CG2 is not pending<br>1: A/D conversion request for CG2 is pending |
| 9 | ADAAn RQ1 | Indicates whether A/D conversion request for CG1 is pending:<br>0: A/D conversion request for CG1 is not pending<br>1: A/D conversion request for CG1 is pending |
| 8 | ADAAn RQ0 | Indicates whether A/D conversion request for CG0 is pending:<br>0: A/D conversion request for CG0 is not pending<br>1: A/D conversion request for CG0 is pending |
| 3 | ADAAn ST3 | Indicates whether the A/D conversion is currently in the ADCHALT state due to a software trigger (ADAAnTRG3.ADAAnSTT3).<br>0: not in ADCHALT state<br>1: in ADCHALT state<br>This bit is cleared when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0). |
| 2 | ADAAn ST2 | Indicates whether A/D conversion of CG2 is currently performed:<br>0: A/D conversion is not currently performed<br>1: A/D conversion is currently performed<br>This bit is cleared when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0). |
| 1 | ADAAn ST1 | Indicates whether A/D conversion of CG1 is currently performed:<br>0: A/D conversion is not currently performed (including halt caused by A/D conversion of CG with higher priority)<br>1: A/D conversion is currently performed<br>This bit is cleared when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0). |
| 0 | ADAAn ST0 | Indicates whether A/D conversion of CG0 is currently performed:<br>0: A/D conversion is not currently performed (including halt caused by A/D conversion of CG with higher priority)<br>1: A/D conversion is currently performed<br>This bit is cleared when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0). |

### (4) ADAAnSTC2 – A/D converter status flag clear register 2

This register is used to clear the overwrite and result check status flags of ADAAnLCR and ADAAnDBiCR.

**Access** This register can be written in 8-bit units.

**Address** <ADAAn_base> + $38_H$

**Initial Value** Reading this register returns always $00_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ADAAn LERC1 | ADAAn LERC0 | ADAAn DB2ERC1 | ADAAn DB2ERC0 | ADAAn DB1ERC1 | ADAAn DB1ERC0 | ADAAn DB0ERC1 | ADAAn DB0ERC0 |
| W | W | W | W | W | W | W | W |

**Table 22-22 ADAAnSTC2 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 7 | ADAAn LERC1 | Clears the overwrite flag ADAAnLCR.ADAAnLER1:<br>0: no function<br>1: ADAAnLCR.ADAAnLER1 is cleared |
| 6 | ADAAn LERC0 | Clears the result check error flag ADAAnLCR.ADAAnLER0:<br>0: no function<br>1: ADAAnLCR.ADAAnLER0 is cleared |
| 5, 3, 1 | ADAAn DBiERC1 | Clears the overwrite flag ADAAnDBiCR.ADAAnDBiER1:<br>0: no function<br>1: ADAAnDBiCR.ADAAnDBiER1 is cleared |
| 4, 2, 0 | ADAAn DBiERC0 | Clears the result check error flag ADAAnDBiCR.ADAAnDBiER0:<br>0: no function<br>1: ADAAnDBiCR.ADAAnDBiER0 is cleared |

### 22.6.4 S/W trigger registers details

#### (1) ADAAnTRGi – A/D converter software trigger register i

This register is the trigger register to *start* the A/D conversion of CGi.

**Access** This register can be written in 8-bit units.

**Address** <ADAAn_base> + $A4_H$ + i × $4_H$

**Initial Value** Reading this register returns always $00_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ADAAn STTi |
| W | W | W | W | W | W | W | W |

**Table 22-23 ADAAnTRGi register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | ADAAn STTi | Starts the A/D conversion of CGi:<br>0: no function<br>1: starts A/D conversion of CGi |

Refer to *22.5.5 “Starting A/D conversion (start trigger modes)” on page 1160* for details.

#### (2) ADAAnTRG3 – A/D converter software trigger register 3

This register is the trigger register for transition to ADCHALT mode.

**Access** This register can be written in 8-bit units.

**Address** <ADAAn_base> + $B0_H$

**Initial Value** Reading this register returns always $00_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ADAAn STT3 |
| W | W | W | W | W | W | W | W |

**Table 22-24 ADAAnTRG3 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | ADAAn STT3 | 0: no function<br>1: transition to ADCHALT mode |

Refer to *22.5.8 “Pausing and resuming A/D conversion (ADCHALT mode)” on page 1164* for details.

### (3) ADAAnTRG4+i – A/D converter S/W trigger register 4+i

This register is the software trigger register to *stop* the A/D conversion of CGi.

**Access** This register can be written in 8-bit units.

**Address** <ADAAn_base> + $B4_H$ + i × $4_H$

**Initial Value** Reading this register returns always $00_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ADAAn SPi |
| W | W | W | W | W | W | W | W |

**Table 22-25 ADAAnTRG4 + i register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | ADAAn SPi | 0: no function<br>1: stops the A/D conversion of CGi |

Refer to *22.5.6 “Stopping A/D conversion” on page 1162* for details.

**Note** If a H/W start trigger for CGi occurs simultaneously with a CGI A/D conversion stop by ADAAnTRG4+i, the H/W start trigger has higher priority.

### (4) ADAAnTRG7 – A/D converter S/W trigger register 7

This register is the S/W trigger register to resume the A/D conversion after a ADCHALT.

**Access** This register can be written in 8-bit units.

**Address** <ADAAn_base> + $C0_H$

**Initial Value** Reading this register returns always $00_H$.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ADAAn SP3 |
| W | W | W | W | W | W | W | W |

**Table 22-26 ADAAnTRG7 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | ADAAn SP3 | 0: no function<br>1: resume A/D conversion |

Refer to *22.5.8 “Pausing and resuming A/D conversion (ADCHALT mode)” on page 1164* for details.

### 22.6.5 ADAA conversion result registers details

#### (1) ADAAnLCR – A/D converter latest conversion result register

This register stores the result and the status of the latest A/D conversion.

**Access** This register can be read in 32-bit units.

- The upper 16 bits store the A/D conversion result status.
- The lower 16 bits store the A/D conversion result.

**Address** <ADAAn_base> + $A0_H$

**Initial Value** $0000\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | ADAAn LCG[1:0] | | ADAAn LER1 | ADAAn LER0 | ADAAn LUR | ADAAn LCN[4:0] | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnLCR[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 22-27 ADAAnLCR register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 25 to 24 | ADAAn LCG[1:0] | Indicates the CG to which the result in ADAAnLCR[15:00] belongs.<br>ADAAn LCG1 / ADAAn LCG0 / Channel group<br>0 / 0 / CG0<br>0 / 1 / CG1<br>1 / 0 / CG2<br>1 / 1 / None |
| 23 | ADAAn LER1 | Indicates the overwrite error status.<br>0: not overwritten<br>1: overwritten<br>This error flag is cleared when ADAAnSTC2.ADAAnLERC1 is set to 1. |
| 22 | ADAAn LER0 | Indicates the A/D conversion result upper/lower limit comparison status.<br>0: The conversion result is within the setting range.<br>1: The conversion result is out of the setting range.<br>This error flag is cleared when ADAAnSTC2.ADAAnLERC0 is set to 1. |
| 21 | ADAAn LUR | Indicates the update status of the A/D conversion result.<br>0: The A/D conversion result has been read from the ADAAnLCR register.<br>1: 1: The A/D conversion result is new and has not been read from ADAAnLCR yet.<br>This bit is cleared after the read operation. |
| 20 to 16 | ADAAn LCN[4:0] | Indicates the channel number corresponding to the conversion result stored in the ADAAnLCR[15:00].<br>00000 = CH0<br>00001 = CH1<br>... = ...<br>01111 = CH15 |

**Table 22-27 ADAAnLCR register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>15 to 0</td><td>ADAAn LCR[15:00]</td><td>Result of the A/D conversion.<br>The alignment is determined by ADAAnCTL1.ADAAnCRAC, as follows.
<table>
<tr><th>ADAAn CRAC</th><th>Alignment</th><th>Bit positions of A/D conversion result</th></tr>
<tr><td>0</td><td>right-aligned</td><td>ADAAnLCR[09:00]</td></tr>
<tr><td>1</td><td>left-aligned</td><td>ADAAnLCR[15:06]</td></tr>
</table>
</td></tr>
</table>

**Note** If A/D conversion is performed using the internal reference voltage, the A/D conversion result is stored in ADAAnDGCR, not in the ADAAnLCR, ADAAnCmCR, and ADAAnDBiCR.

### (2) ADAAnCmCR – A/D converter conversion result register for channel m

This register stores the result and the status of the latest A/D conversion of channel m.

**Access** This register can be read in 32-bit units.

- The upper 16 bits store the A/D conversion result status.
- The lower 16 bits store the A/D conversion result.

**Address** <ADAAn_base> + $3C_H$ + m × $4_H$

**Initial Value** $0300\ 0000_H$ + m × $0001\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | ADAAn CmCG[1:0] | | ADAAn CmER1 | ADAAn CmER0 | ADAAn CmUR | ADAAn CmCN[4:0] | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnCmCR[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Notes**

1. The functions of the individual bits are identical to those of the ADAAnLCR register, except that each of the m registers hold the latest result for the specific channel m rather than the latest A/D conversion result for any channel.
2. If ADAAnCTL1.ADAAnRCL = 0, the A/D conversion result in ADAAnCmCR[15:00] is kept until it is overwritten by the next A/D conversion result.

   If ADAAnCTL1.ADAAnRCL = 1, ADAAnCmCR[15:00] is cleared by reading it.

**Table 22-28 ADAAnCmCR register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 25 to 24 | ADAAn CmCG[1:0] | Indicates the CG to which the result in ADAAnCmCR[15:00] belongs. |
| 23 | ADAAn CmER1 | Indicates the overwrite error status.<br>0: not overwritten<br>1: overwritten<br>This error flag reflects the value of ADAAnSTR1.ADAAnOWEm. It is cleared when ADAAnSTC1.ADAAnQWECm is set to 1. |
| 22 | ADAAn CmER0 | Indicates the A/D conversion result upper/lower limit comparison status.<br>0: The conversion result is within the setting range.<br>1: The conversion result is out of the setting range.<br>This error flag reflects the value of ADAAnSTR0.ADAAnRCEm. It is cleared when ADAAnSTC0.ADAAnRCECm is set to 1. |

| ADAAn CmCG1 | ADAAn CmCG0 | Channel group |
|---|---|---|
| 0 | 0 | CG0 |
| 0 | 1 | CG1 |
| 1 | 0 | CG2 |
| 1 | 1 | None |

**Table 22-28 ADAAnCmCR register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>21</td><td>ADAAn<br>CmUR</td><td>Indicates the update status of the A/D conversion result.<br>0: The A/D conversion result has been read from the ADAAnCmCR register.<br>1: 1: The A/D conversion result is new and has not been read from ADAAnCmCR yet.<br>This bit is cleared after the read operation.</td></tr>
<tr><td>20 to 16</td><td>ADAAn<br>CmCN[4:0]</td><td>Indicates the channel number corresponding to the conversion result stored in the ADAAnCmCR[15:00].<br>00000 = CH0<br>00001 = CH1<br>... = ...<br>01111 = CH15</td></tr>
<tr><td>15 to 0</td><td>ADAAn<br>CmCR[15:00]</td><td>Result of the A/D conversion.<br>The alignment is determined by ADAAnCTL1.ADAAnCRAC, as follows.
<table>
<tr><th>ADAAn CRAC</th><th>Alignment</th><th>Bit positions of A/D conversion result</th></tr>
<tr><td>0</td><td>right-aligned</td><td>ADAAnCmCR[09:00]</td></tr>
<tr><td>1</td><td>left-aligned</td><td>ADAAnCmCR[15:06]</td></tr>
</table></td></tr>
</table>

**Note** If A/D conversion is performed using the internal reference voltage, the A/D conversion result is stored in ADAAnDGCR, not in the ADAAnLCR, ADAAnCmCR, and ADAAnDBiCR.

### (3) ADAAnDBiCR – A/D converter CGI buffer register

This register stores the result and the status of the latest A/D conversion of CGi. It allows the A/D conversion results for all channels of CGi to be read continuously.

**Access** This register can be read in 32-bit units.

- The upper 16 bits store the A/D conversion result status.
- The lower 16 bits store the A/D conversion result.
  The conversion result can also be read via the ADAAnDBiCRL register.

**Address** <ADAAn_base> + $C4_H$ + i × $4_H$

**Initial Value** 0000 $0000_H$ + i × 0100 $0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | ADAAn DBiCG[1:0] | | ADAAn DBiER1 | ADAAn DBiER0 | ADAAn DBiUR | ADAAn DBiCN[4:0] | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnDBiCR[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Note** The functions of the individual bits are identical to those of the ADAAnLCR register, except that each of the i registers hold the latest result for the specific channel group i rather than the latest A/D conversion result for any channel.

**Table 22-29 ADAAnDBiCR register contents (1/2)**

| Bit position | Bit name | Function |
|---|---|---|
| 25 to 24 | ADAAn DBiCG[1:0] | Indicates the CG to which the result in ADAAnDBiCR[15:00] belongs.<br>ADAAn DBiCG1 / ADAAn DBiCG0 / Channel group<br>0 / 0 / CG0<br>0 / 1 / CG1<br>1 / 0 / CG2<br>1 / 1 / None |
| 23 | ADAAn DBiER1 | Indicates the overwrite error status.<br>0: not overwritten<br>1: overwritten<br>This error flag is cleared when ADAAnSTC2.ADAAnDBiERC1 is set to 1. |
| 22 | ADAAn DBiER0 | Indicates the A/D conversion result upper/lower limit comparison status.<br>0: The conversion result is within the setting range.<br>1: The conversion result is out of the setting range.<br>This error flag is cleared when ADAAnSTC2.ADAAnDBiERC0 is set to 1. |
| 21 | ADAAn DBiUR | Indicates the update status of the A/D conversion result.<br>0: The A/D conversion result has been read from the ADAAnDBiCR register.<br>1: 1: The A/D conversion result is new and has not been read from ADAAnDBiCR yet.<br>This bit is cleared after the read operation. |

**Table 22-29 ADAAnDBiCR register contents (2/2)**

<table>
<tr><th>Bit position</th><th>Bit name</th><th>Function</th></tr>
<tr><td>20 to 16</td><td>ADAAn DBiCN[4:0]</td><td>Indicates the channel number corresponding to the conversion result stored in the ADAAnDBiCR[15:00].<br>00000 = CH0<br>00001 = CH1<br>... = ...<br>01111 = CH15</td></tr>
<tr><td>15 to 0</td><td>ADAAn DBiCR[15:00]</td><td>Result of the A/D conversion.<br>The alignment is determined by ADAAnCTL1.ADAAnCRAC, as follows.
<table>
<tr><th>ADAAn CRAC</th><th>Alignment</th><th>Bit positions of A/D conversion result</th></tr>
<tr><td>0</td><td>right-aligned</td><td>ADAAnDBiCR[09:00]</td></tr>
<tr><td>1</td><td>left-aligned</td><td>ADAAnDBiCR[15:06]</td></tr>
</table></td></tr>
</table>

**Note** If A/D conversion is performed using the internal reference voltage, the A/D conversion result is stored in ADAAnDGCR, not in the ADAAnLCR, ADAAnCmCR, and ADAAnDBiCR.

### (4) ADAAnDBiCRL – A/D converter CGI buffer result register

This register stores the result of the latest A/D conversion of CGi.

The content of this register is identical to the lower 16 bit of the ADAAnDBiCR register (ADAAnDBiCR[15:00]).

**Access** This register can be read in 16-bit units.

**Address** <ADAAn_base> + $D0_H$ + i × $4_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnDBiCR[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

### (5) ADAAnDGCR – Diagnostic conversion result register

This register stores the A/D conversion result of the reference voltage ADDIAGOUT signal (when ADAAnCG0.ADAAnDIAG = 1).

The diagnosis conversion starts after the A/D conversion of the last channel of CG0 has been completed.

**Access** This register can be read in 16-bit units.

**Address** <ADAAn_base> + $9C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnDGCR[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 22-30 ADAAnDGCR register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | ADAAn DGCR[15:00] | Result of the diagnostic A/D conversion.<br>The alignment is determined by ADAAnCTL1.ADAAnCRAC, as follows. |

| ADAAn CRAC | Alignment | Bit positions of A/D conversion result |
|---|---|---|
| 0 | right-aligned | ADAAnDBiCR[09:00] |
| 1 | left-aligned | ADAAnDBiCR[15:06] |

## 22.6.6 A/D conversion result upper/lower limit comparison registers details

### (1) ADAAnCTL2 – A/D converter result check register

This register can enable/disable the conversion result upper/lower limit comparison function for each channel.

Refer to *22.5.12 “Result check functions” on page 1169* for details.

**Access** This register can be read/written in 32-bit units.
It can only be written when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0).

**Address** <ADAAn_base> + $18_H$

**Initial Value** $0000\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnRCK[15:00] | | | | | | | | | | | | | | | |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-31 ADAAnCTL2 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 00 | ADAAn RCK[15:00] | Enables/disables result upper/lower limit comparison for CHm.<br>0: do not perform upper/lower limit comparison for A/D conversion of CHm<br>1: perform upper/lower limit comparison for A/D conversion of CHm |

**Note** The settings are valid for A/D conversions of every CG.

### (2) ADAAnUL – A/D converter result check upper limit register

This register specifies the upper limit of the A/D conversion result.

Refer to *22.5.12 "Result check functions" on page 1169* for details.

**Access** This register can be read/written in 16-bit units.
It can only be written when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0).

**Address** <ADAAn_base> + $1C_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnUL[09:00] | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-32 ADAAnUL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to | ADAAn UL[09:00] | Specifies the upper limit of the A/D conversion result. |

### (3) ADAAnLL – A/D converter result lower limit register

This register specifies the lower limit of the A/D conversion result.

Refer to *22.5.12 "Result check functions" on page 1169* for details.

**Access** This register can be read/written in 16-bit units.
It can only be written when the A/D converter is disabled (ADAAnCTL0.ADAAnCE = 0).

**Address** <ADAAn_base> + $20_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnLL[09:00] | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-33 ADAAnLL register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 6 | ADAAn LL[09:00] | Specifies the lower limit of the A/D conversion result. |

### (4) ADAAnSTR0 – A/D converter result check error flag register

This register indicates the error status of the latest A/D conversion result upper/lower limit comparison for the channel set in the ADAAnCTL2 register. By evaluating this register it is possible to deduce which A/D conversion results are outside the specified range.

Refer to *22.5.12 “Result check functions” on page 1169* for details.

**Access** This register can be read in 32-bit units.

**Address** <ADAAn_base> + $24_H$

**Initial Value** $0000\ 0000_H$

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnRCE[15:00] | | | | | | | | | | | | | | | |
| R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

**Table 22-34 ADAAnSTR0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | ADAAn RCE[15:00] | Indicates whether the A/D conversion result is within the specified value range:<br>0: conversion results within the specified range<br>1: at least one Conversion result out of the specified range<br>This error flag is cleared by setting ADAAnSTC0.ADAAnRCECm to 1.<br>All flags are also cleared by enabling and disabling the ADAA (ADAAn.ADAAnCE= 0 and then ADAAn.ADAAnCE= 1). |

**Note** ADAAnSTR0.ADAAnRCEm is mirrored by the following A/D conversion result error flag:

- Error flag in A/D converter conversion result register for channel m (ADAAnCmCR.ADAAnCmER0)

### (5) ADAAnSTC0 – ADAAnSTR0 flag clear register

This register is the clear control register of ADAAnSTR0.

**Access** This register can be written in 32-bit units.

**Address** <ADAAn_base> + $30_H$

**Initial Value** Reading this register always returns 0000 $0000_H$.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADAAnRCEC[15:00] | | | | | | | | | | | | | | | |
| W | W | W | W | W | W | W | W | W | W | W | W | W | W | W | W |

**Table 22-35 ADAAnSTC0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 15 to 0 | ADAAn RCEC[15:00] | 0: no function<br>1: clears the corresponding ADAAnSTR0.ADAAnRCEm |

## 22.6.7 Diagnose functions registers

### (1) ADAAnDGCTL0 – Self-diagnosis function control register 0

This register specifies the reference voltage to be applied to diagnose the A/D conversion circuit operation.

**Access** This register can be read/written in 16-bit units.

**Address** <ADAAn_base> + $DC_H$

**Initial Value** $0000_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ADAAn PSEL | 0 | 0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-36 ADAAnDGCTL0 register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 2 | ADAAn PSEL | Specifies the reference voltage ADDIAGOUT:<br>0: ADDIAGOUT = AVSS<br>1: ADDIAGOUT = AVDD |

Refer to *22.5.13 “Self-diagnosis function” on page 1171* for details.

### 22.6.8 Emulation register

#### (1) ADAAnEMU - Emulation register

This register controls whether the A/D Converter can be stopped during emulation, for instance upon a breakpoint hit.

**Access** This register can be read/written in 8-bit units.

**Address** <ADAAn_base> + $128_H$

**Initial Value** $00_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | ADAAn SVSDIS |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Table 22-37 ADAAnEMU register contents**

| Bit position | Bit name | Function |
|---|---|---|
| 0 | ADAAn SVSDIS | Emulation control<br>0: ADAAn can be stopped during emulation<br>1: ADAAn continuous operating during emulation |

# Chapter 23 On-chip Debug Unit (OCD)

This microcontroller has a debug function on-chip. By using the on-chip debug emulator, programs can be debugged with the microcontroller mounted in the target system.

The debug functions incorporated in this microcontroller conform to IEEE-ISTO 5001™-2003 Class 1, a Nexus debug interface standard.

**Caution** The debug functions described in this chapter are supported by the microcontroller but whether they are usable depends on the debugger. For details of debugging, see the user's manual of the debugger.

## 23.1 V850E2/Fx4-G On-chip Debug Features

### 23.1.1 Modules behaviour during emulation break

This section specifies, which modules

- are always stopped (unconditional emulation break)
- can optionally be stopped (emulation break function)
- continue operation,

if the debugger stops the microcontroller's operation in case of an emulation break.

**Emulation break** An emulation break refers to a

- breakpoint hit
- manual break

during a debug session.

**(1) Modules with unconditional emulation break**

Following list shows all modules which are always stopped upon a emulation break:

**Table 23-1 Modules with unconditional emulation break**

| Module |
|---|
| Window Watchdog Timer A (WDTA) |

**(2) Modules with optional emulation break**

Modules with optional emulation break provide a control bit, that allows the application software to make the module stop or continue upon emulation break.

These modules support the emulation stop function.

Following list shows all modules which can optionally be stopped or continue upon an emulation break:

**Table 23-2 Modules with optional emulation break**

| Module | Control register bit |
|---|---|
| OS Timer (OSTM) | OSTMnEMU.OSTMnSVSDIS:<br>0: stop during break<br>1: continue during break |
| Timer Array Unit B (TAUB) | TAUBnEMU.TAUBnSVSDIS:<br>0: stop during break<br>1: continue during break |
| Timer Array Unit J (TAUJ) | TAUJnEMU.TAUJnSVSDIS:<br>0: stop during break<br>1: continue during break |
| Clocked Serial Interface G (CSIG) | CSIGnEMU.CSIGnSVSDIS:<br>0: stop during break<br>1: continue during break |
| Asynchronous Serial Interface E (URTE) | URTEnEMU.URTEnSVSDIS:<br>0: stop during break<br>1: continue during break |
| $I^2C$ Interface (IICB) | IICBnEMU.IICBnSVSDIS:<br>0: stop during break<br>1: continue during break |
| A/D Converter (ADAA) | ADAAnEMU.ADAAnSVSDIS:<br>0: stop during break<br>1: continue during break |

**(3) Modules continuing operation at emulation break**

Following list shows all modules continuing operation upon an emulation break:

**Table 23-3 Modules continuing operation at emulation break**

| Module |
|---|
| CAN Controller (FCN) |
| Key Return Function (KR) |

## 23.1.2 Signal masking

Following V850E2/Fx4-G external signals can be masked, so that they don't have any effect, while the microcontroller is controlled by the On-chip Debug Unit:

- $\overline{\text{RESET}}$
- NMI

## 23.2 Functional Overview

The On-Chip Debug functions are outlined below.

**(1) Debug interface**

This interface is used to communicate with the host by using the $\overline{\text{DCUTRST}}$, DCUTCK, DCUTMS, DCUTDI, DCUTDO, and $\overline{\text{DCUTRDY}}$ signals via the on-chip debug emulator.

**(2) Debug monitoring function**

The basic debug functions below can be used by running a monitoring program in a memory space for debugging while execution of the user-created program is paused.

- downloading the user-created program
- reading and writing the memory and registers
- running the user-created program starting at any address

**(3) Hardware break function**

Up to four breakpoints can be specified for instructions and data. If a breakpoint is specified for an instruction, execution can be interrupted at the specified address. If a breakpoint is specified for an address, execution can be interrupted when data at the specified address is accessed.

In addition, break conditions can be combined by using a sequence of up to two levels.

**(4) Software break function**

Execution of the user-created program stored in the RAM can be interrupted at the specified address.

**(5) Forced break function**

Execution of the user-created program can be interrupted forcibly.

**(6) Forced reset function**

The microcontroller can be reset forcibly.

**(7) Real-time RAM monitoring (RRM)**

The memory can be read during program execution. Because this read access uses debug-dedicated DMA, it has minimal effect on program execution.

**(8) Dynamic memory modification (DMM)**

The memory can be written during program execution. Because this write access uses debug-dedicated DMA, it has minimal effect on program execution.

**(9) Timer function**

Using a 32-bit counter, the time for running the user-created program can be measured based on the clock obtained by dividing the DCUTCK signal frequency by 2.

**(10) Mask function**

Some dedicated external signals can be masked, so that they don't have any effect, while the microcontroller is controlled by the On-Chip Debug Unit.

These signals are listed in the section *"Signal masking"* above in this chapter.

**(11) Modules run/stop selection during a break**

Upon a breakpoint hit the microcontroller's modules behave as follows:

- module always stops operation during break
- behaviour of the module during break is an user's option and can be specified by use of the module's emulation register.
  This emulation break function needs to be generally enabled, refer to *23.3 "Emulation Break Control" on page 1209* for details.
- module always stays in operation during break

The bahaviour of the modules of this microcontroller is described in the section *"Modules behaviour during emulation break"* above in this chapter.

**(12) Hot attach function**

The On-Chip Debug emulator can be connected and start debugging without resetting the CPU while it is running.

**(13) Security function**

To prevent the contents of the flash memory from being read by an unauthorized person, a 96-bit ID code can be written to the microcontroller. If the code the user inputs when starting a debugger does not match the ID code written to the microcontroller, the flash memory cannot be accessed. If the OCDID[95] bit of the On-Chip Debug ID register OCDIDH is set to 0, the flash memory cannot be accessed even if the ID code matches.

For details of how to set the ID code, see the user's manual of the used software tools and refer to section *"On-Chip Debug Interface Protection"* in the chapter *"Code Protection and Security"* in this manual.

**(14) On-Chip Debug and Stand-by modes**

Setting the device in STOP or DEEPSTOP mode is not possible, if the microcontroller is operated under control of an On-Chip Debugger.

## 23.3 Emulation Break Control

The emulation break function generates a stop request to the modules of the microcontroller, if the debugger is taking over the control of the microcontroller, for instance upon a breakpoint hit.

Those modules stop their operation during emulation break, if

- the module supports the emulation break function (for a list of these modules refer to *"Modules behaviour during emulation break"* in the first section of this chapter)
- emulation break is enabled in the module (by setting the module's emulation register).

**Debugger support**

In general the debugger provides an option to enable the emulation break function. In this case the application program does not need to take care for that.

## 23.4 Connection with On-Chip Debug Emulator

Following signals are used to connect the debugger with the microcontroller:

**Table 23-4 Signals used to connect On-Chip Debug emulator**

| Pin name | Description |
|---|---|
| VDD | Signal used to detect power supply to the target system, or the power supply for the buffers in the On-Chip Debug emulator |
| $\overline{\text{DCUTRST}}$ | Signal to asynchronously reset the debug functions of the microcontroller |
| DCUTCK | Clock signal used for debugging |
| DCUTMS | Signal to select the transfer mode for data communication |
| DCUTDI | Data signal input to the microcontroller |
| DCUTDO | Data signal output from the microcontroller |
| $\overline{\text{DCUTRDY}}$ | Synchronization signal for data communication |
| FLMD0 | Mode signal used to rewrite the flash memory in the microcontroller |

Refer to the section *“Operation Modes”* in chapter *“CPU System Functions”* for details.

## 23.5 Cautions on using On-Chip Debugging

### (1) Handling of device that was used for debugging

Do not mount a device that was used for debugging on a mass-produced product, because the flash memory was rewritten during debugging and thus the number of flash memory rewrites cannot be guaranteed.

# Chapter 24 Power Supply

The V850E2/Fx4-G devices provide separate power supply pins for the digital and analog circuits of the functional modules on the different power domains and for several groups of port I/O buffers.

## 24.1 Power supply pins naming

In this section some general information is given about the naming of the power supply pins.

In general the name of a power supply pin is composed of up to four fields with the following meaning:

| Supply type | Prefix | Kind of supply | Suffix |
|---|---|---|---|
| Symbol, representing the purpose of the supply | Consecutive number for separate supplies[a] | VDD or VSS | Consecutive number for different pins for same supply[a] |

a) Prefix and suffix number may be omitted, if only a single supply pin is used.

The different supply types are defined in the following table.

**Table 24-1 Supply types symbols**

| Symbol | Explanation |
|---|---|
| A | Analog circuits supply (e.g. analog parts of the A/D Converter) |
| B | Standard I/O buffer supply |
| E | Standard I/O buffer supply |
| OSC | Oscillator supply |
| REG | Internal voltage regulator input voltage |

Examples:

- E0VDD: supply for standard I/O buffers group “0” via single pin
- B1VDD2: supply for standard I/O buffers group “1” via several pins (this is the second pin)
- REG0VSS: input voltage for on-chip voltage regulator REG0

## 24.2 Power supply schemes

The following sections show the power supply schemes, i.e. the power supply pins and the modules they supply.

Information about the voltage ranges of the power supply pins and all conditions related to them are provided in the Data Sheet. The Data Sheet provides also details about the electrical properties of the port pins.

### 24.2.1 V850E2/FF4-G power supply scheme

The following table and diagram shows the power supply scheme of the V850E2/FF4-G devices.

**Table 24-2 V850E2/FF4-G power supply pins**

| Pin | Modules | Ports buffers |
|---|---|---|
| REG0VDD/REG0VSS/ REG0C | Internal voltage regulator REG0<br>$V_{REG0}$ supplies digital circuits of Always-On-Area modules | – |
| REG1VDD/REG1VSS/ REG1C | Internal voltage regulator REG1<br>$V_{REG1}$ supplies digital circuits of Isolated-Area-0 modules | – |
| E0VDD/E0VSS | – | JP0, P0 |
| E1VDD/E1VSS | – | P1, P3, P4 |
| OSCVDD/OSCVSS | MainOsc supply | – |
| A0VDD/A0VSS | Analog circuits of A/D Converter ADAA0 | P10 |
| I0VDD/VSS | Flash memory | – |

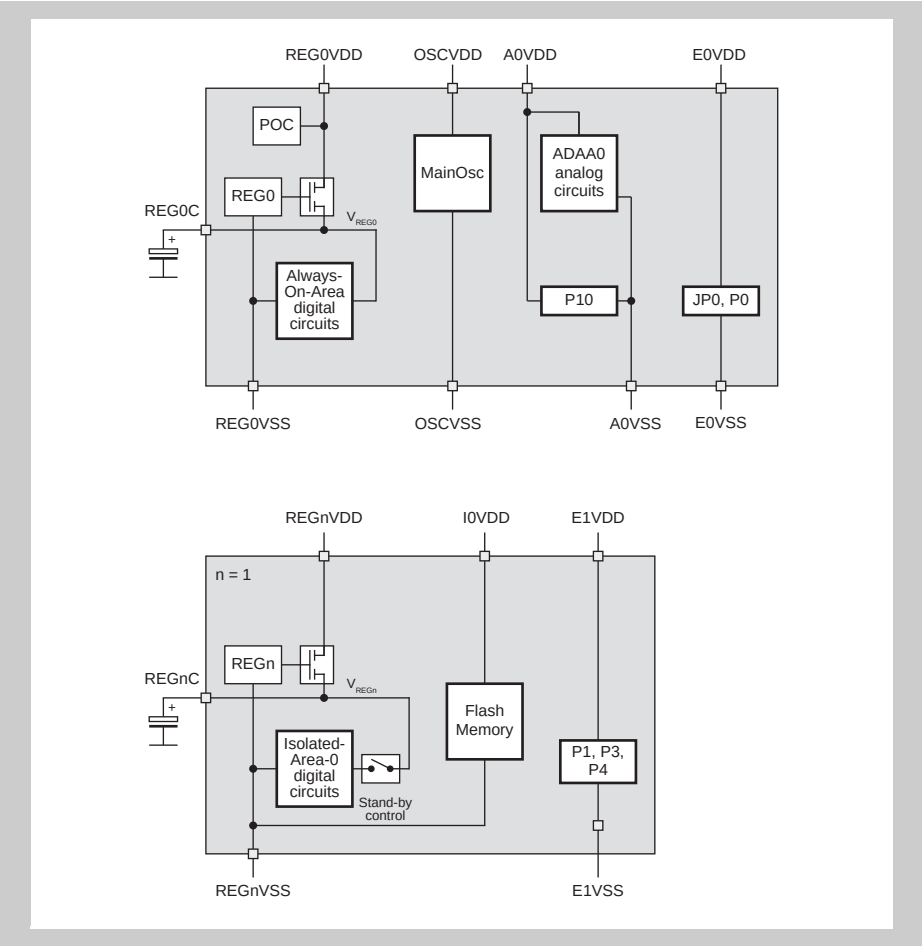



**Figure 24-1 V850E2/FF4-G power supply scheme**

**Note** Refer to the Data Sheet for information about the capacitance to be connected to REG0C and REG1C.

### 24.2.2 V850E2/FG4-G power supply scheme

The following table and diagram shows the power supply scheme of the V850E2/FG4-G devices.

**Table 24-3 V850E2/FG4-G power supply pins**

| Pin | Modules | Ports buffers |
|---|---|---|
| REG0VDD/REG0VSS/ REG0C | Internal voltage regulator REG0<br>$V_{REG0}$ supplies digital circuits of Always-On-Area modules | – |
| REG1VDD/REG1VSS/ REG1C | Internal voltage regulator REG1<br>$V_{REG1}$ supplies digital circuits of Isolated-Area-0 modules | – |
| E0VDD/E0VSS | – | JP0, P0 |
| E1VDD/E1VSS | – | P1, P3, P4, P27 |
| OSCVDD/OSCVSS | MainOsc supply | – |
| A0VDD/A0VSS | Analog circuits of A/D Converter ADAA0 | P10 |
| I0VDD/VSS | Flash memory | – |

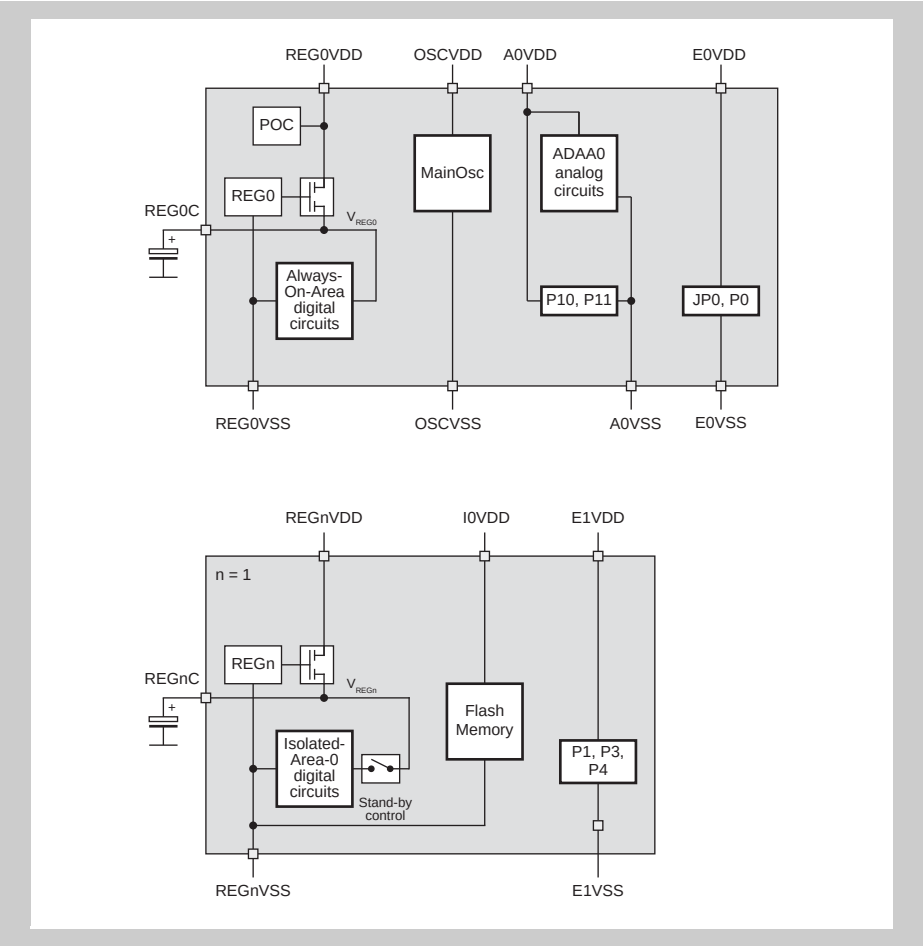



**Figure 24-2 V850E2/FG4-G power supply scheme**

**Note** Refer to the Data Sheet for information about the capacitance to be connected to REG0C and REG1C.

## 24.3 Power supply control

This section describes the power supply control.

**Caution** All figures are only for explanatory purposes without any relevance to the real hardware implementation.

For detailed information about voltage levels, time and frequency values refer to the Data Sheet.

The following figure outlines the power supply control.

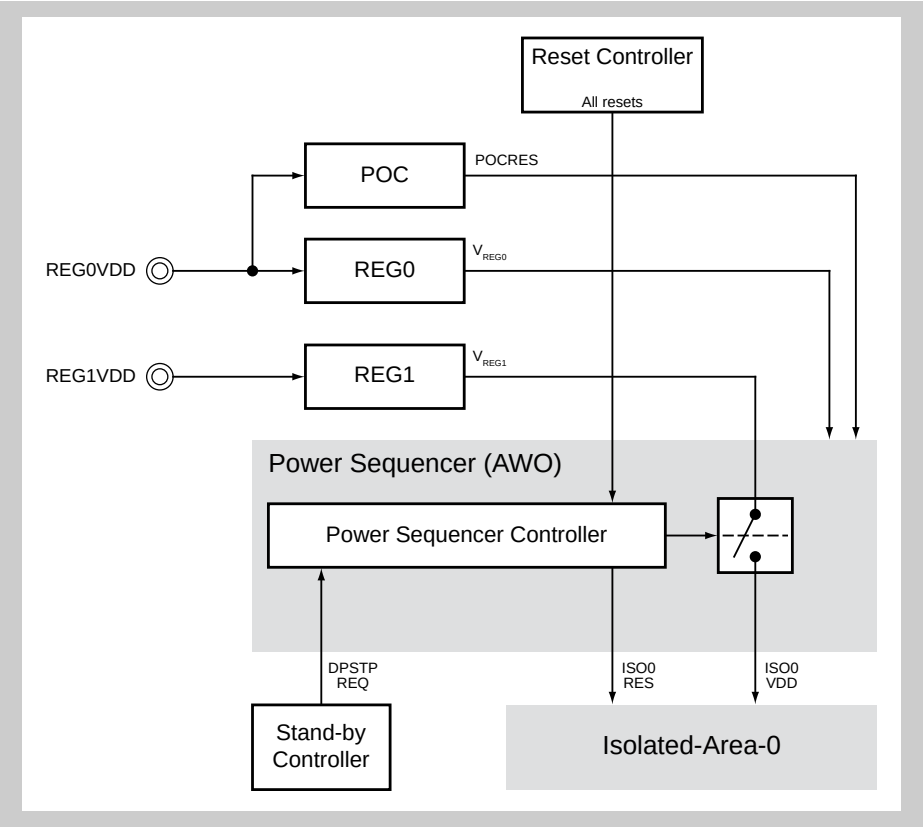



**Figure 24-3 Power supply control scheme**

**AWO supply** The supply voltage $V_{REG0}$ of the Always-On-Area is generated by the on-chip voltage regulator REG0 from the input voltage REG0VDD.
REG0VDD is observed by the Power-On-Clear circuit POC, that keeps the Always-On-Area in reset state (POCRES) as long as REG0VDD voltage is below the POC level $V_{POC}$.

**Isolated-Area-0 supply** The Isolated-Area-0 supply is generated by the on-chip voltage regulator REG1 from its input voltage REG1VDD.

Power-up and -down of the Isolated-Area-0 is necessary in two situations:

- initial power-up and final power-down
  The entire microcontroller is powered up respectively down, including the Always-On-Area.

- DEEPSTOP mode entry and wake-up
  In DEEPSTOP mode only the supply of the Isolated-Area-0 ISO0VDD is switched off, while the power supply of the Always-On-Area remains active. DEEPSTOP mode entry and wake-up is signalled via the DPSTPREQ signal, which are asserted by the Stand-by Controller.

**Isolated-Area-0 supply during reset**

Any reset switches off the power supply of the Isolated-Area-0.
Thus after a reset the Isolated-Area-0 has the same status as after initial power-up or after wake-up from DEEPSTOP mode, i.e. it has to be completely re-initialized.

**Note**

Be aware that also after any reset the internal CPU RAM content is undefined.

**Power Sequencer**

The Power Sequencer controls the correct power-up and -down of the Isolated-Area-0.

For that purpose it is located on the Always-On-Area and starts operation if the AWO power supply $V_{REG0}$ is stable and AWO reset AWORES is released.

For controlling the Isolated-Area-0 the Power Sequencer

- switches on the Isolated-Area-0 power supply ISO0VDD if the Isolated-Area-0 supplies $V_{REG1}$ are stable, respectively switches off ISO0VDD when $V_{REG1}$ are not stable
- controls the Isolated-Area-0 reset ISO0RES in order to avoid operation of the Isolated-Area-0, if its power supply is not stable

The Power Sequencer ensures that the Isolated-Area-0 voltage supply ISO0VDD is switched on and ISO0RES is released when $V_{REG1}$ is stable.

# Revision History

The table below gives an overview about the revision history of this document.

**Revison history overview**

| Rev. | Date | Summary |
|---|---|---|
| 1.00 | Feb 28, 2013 | 1st edition |

# Index

## Numerics



## A

## B



## C

## P

## T

V850E2/Fx4-G User Manual

Publication Date: Rev. 1.00 February 28, 2013

Published by: Renesas Electronics Corporation

**SALES OFFICES** Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.


**Renesas Electronics America Inc.**
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

**Renesas Electronics Canada Limited**
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

**Renesas Electronics Europe Limited**
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

**Renesas Electronics Europe GmbH**
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

**Renesas Electronics (China) Co., Ltd.**
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

**Renesas Electronics Hong Kong Limited**
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

**Renesas Electronics Taiwan Co., Ltd.**
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

**Renesas Electronics Singapore Pte. Ltd.**
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

**Renesas Electronics Malaysia Sdn.Bhd.**
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

**Renesas Electronics Korea Co., Ltd.**
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141



Colophon 1.3

# V850E2/Fx4-G

Renesas Electronics Corporation

R01UH0366EJ0100