To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

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.

RENESAS

Application Note

V850 Series™

Peripheral Macro Driver

V850ES/Fx2 Microcontrollers,
V850ES/Sx2 Microcontrollers

Document No. U17330EE1V0AN00
Date Published October 2005

NOTES FOR CMOS DEVICES

① VOLTAGE APPLICATION WAVEFORM AT INPUT PIN

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (MAX) and V_{IH} (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (MAX) and V_{IH} (MIN).

② HANDLING OF UNUSED INPUT PINS

Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

③ PRECAUTION AGAINST ESD

A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

④ STATUS BEFORE INITIALIZATION

Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

⑤ POWER ON/OFF SEQUENCE

In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current.

The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

⑥ INPUT OF SIGNAL DURING POWER OFF STATE

Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements.

Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

- **The information in this document is current as of October, 2005. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "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.
  "Special": 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, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)
(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics America Inc.
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 1101
Fax: 0211-65 03 1327

Sucursal en España
Madrid, Spain
Tel: 091- 504 27 87
Fax: 091- 504 28 60

Succursale Française
Vélizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

Filiale Italiana
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

Branch The Netherlands
Eindhoven, The Netherlands
Tel: 040-244 58 45
Fax: 040-244 45 80

Branch Sweden
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

United Kingdom Branch
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Singapore
Tel: 65-6253-8311
Fax: 65-6250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

Table of Contents

List of Tables

Introduction

Readers

This application note is intended for users who want make use of the an Application Programming Interface (API) as known as the Peripheral Macro Driver.

Purpose

This application note presents the software application note of the Peripheral Macro Driver.

Organization

This system specification describes the following sections:

- Module structure
- Application Programming Interface
- Example code

Legend

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
Note	:	Explanation of (Note) in the text
Caution	:	Information requiring particular attention
Remark	:	Supplementary explanation to the text
Numeric notation	:	Binary... xxxx or xxxB Decimal... xxxx Hexadecimal... xxxxH or 0x xxxx

Prefixes representing powers of 2 (address space, memory capacity)

K (kilo): $2^{10} = 1024$

M (mega): $2^{20} = 1024^2 = 1,048,576$

G (giga): $2^{30} = 1024^3 = 1,073,741,824$

Chapter 1 Overview

NEC devices provide many different peripherals. The line concept of V850ES/Fx2 and V850ES/Sx2 Microcontrollers is to use the same peripheral macros for each specific device within or across these lines. The configuration of each peripheral is done by manipulating bits and bytes in special function registers (SFR's). The names of the SFR's are short abbreviations, less good to remember. A definition file (header file) including all register and bit definitions is available for each specific device. A fast configuration presuppose a good knowledge of the peripheral and all related registers.

The Peripheral Macro Driver represent an Application Programming Interface (API) between the hardware level of a device and the user program. The term Peripheral Macro Driver and Application Programming Interface is used with the same meaning inside this document.

This software provides simple parametrized function calls to initialize and utilize the peripherals. The names of the functions and parameters include less abbreviations and are more-or-less self-explaining. The clarity and transparency of the program source code is improved. There is no more need to set bits and bytes in SFR's with cryptic abbreviations.

The description of the correct calling of the APIs with parameters is subject of this manual.
The Peripheral Macro Driver is intended for Users who have a basic understanding of the peripheral.
This document is an extension to the User's Manual. It do not replace the description in the User's Manual.

1.1 Peripheral Macro Driver Software

The Driver uses function-like C-language macro defines. The advantage is that no code is used if they are unused. The disadvantage is that they need more code size compared to a function call in the case huge macros are often called. If this disadvantage should be avoided, the defines should either be included in tiny substitute functions, or the define should be replaced by a function definition.
Many macros are conditional of input parameters. Unused code definition inside the macro is removed by the preprocessor and the optimizing compiler. E.g. an initialization function for a timer include only the code which is necessary for this specific timer channel. Code for initializing the other timer channels is removed.

The Peripheral Macro Driver is written in ANSI-C language and can be used with any compiler (e.g. Green Hills or IAR). Mainly all APIs are grouped in header files for each peripheral. A device specific file ("device.h") summarize the including of all header files. It includes the Special Function Register (SFR) header, the device specific and generic Peripheral Macro Driver header. The device specific Peripheral Macro Driver header file is necessary for the generic file and define e.g. initialization macros for port pins etc. The generic part include the only functions which should by called by the User directly. The macros inside the device specific Peripheral Macro Driver header file should not be used.
The software package include a demo project for Green Hills and IAR software development environment.

1.2 Supported Devices

The Peripheral Macro Driver is written for the Devices of the V850ES/Fx2 and V850ES/Sx2 Microcontrollers. The generic macro headers can be reused for other devices if they have included the same peripheral macros. The device specific macro header file must be adapted to a different device. The easiest way to do it, is to copy the device specific macro header and replace or modify the relevant sections.

The macros support the biggest device of each family. V850ES/FJ2 of the F-Line and V850ES/SJ2 of the S-Line are fully supported. All smaller devices represent a subset of these devices. A smaller device uses also a subset of all APIs.

Supported devices

V850ES/Fx2 Microcontrollers

µPD70F3230(A)	µPD70F3234(A)	µPD70F3238(A)
µPD70F3231(A)	µPD70F3235(A)	µPD70F3239(A)
µPD70F3232(A)	µPD70F3236(A)	
µPD70F3233(A)	µPD70F3237(A)	

V850ES/Sx2 Microcontrollers

µPD70F3260(A)	µPD70F3271(A)	µPD70F32F81(A)
µPD70F3260Y(A)	µPD70F3271Y(A)	µPD70F32F81Y(A)
µPD70F3261(A)	µPD70F32F71(A)	µPD70F3282(A)
µPD70F3261Y(A)	µPD70F32F71Y(A)	µPD70F3282Y(A)
µPD70F32F61(A)	µPD70F3272(A)	µPD70F3283(A)
µPD70F32F61Y(A)	µPD70F3272Y(A)	µPD70F3283Y(A)
µPD70F3262(A)	µPD70F3273(A)	µPD70F32F83(A)
µPD70F3262Y(A)	µPD70F3273Y(A)	µPD70F32F83Y(A)
µPD70F3263(A)	µPD70F32F73(A)	µPD70F3284(A)
µPD70F3263Y(A)	µPD70F32F73Y(A)	µPD70F3284Y(A)
µPD70F32F63(A)	µPD70F3274(A)	µPD70F32F84(A)
µPD70F32F63Y(A)	µPD70F3274Y(A)	µPD70F32F84Y(A)
µPD70F3264(A)	µPD70F32F74(A)	µPD70F3285(A)
µPD70F3264Y(A)	µPD70F32F74Y(A)	µPD70F3285Y(A)
µPD70F32F64(A)	µPD70F3275(A)	µPD70F3286(A)
µPD70F32F64Y(A)	µPD70F3275Y(A)	µPD70F3286Y(A)
µPD70F3265(A)	µPD70F3276(A)	µPD70F32F86(A)
µPD70F3265Y(A)	µPD70F3276Y(A)	µPD70F32F86Y(A)
µPD70F3266(A)	µPD70F32F76(A)	µPD70F3287(A)
µPD70F3266Y(A)	µPD70F32F76Y(A)	µPD70F3287Y(A)
µPD70F32F66(A)	µPD70F3280(A)	µPD70F3288(A)
µPD70F32F66Y(A)	µPD70F3280Y(A)	µPD70F3288Y(A)
µPD70F3270(A)	µPD70F3281(A)	µPD70F32F88(A)
µPD70F3270Y(A)	µPD70F3281Y(A)	µPD70F32F88Y(A)

Notes: 1. The macros do not check the availability of the peripherals in the specific device. E.g. I²C is not supported in non “Y”-type devices.

2. Some peripherals are less present in smaller devices. E.g. the timer channels of a small device is also reduced compared to the biggest devices of each family. The macros do not check the maximum channel count.

3. Please check the User’s Manual for the availability and maximum count of each peripheral.

Chapter 2 Peripheral Macro Driver File Structure

Only one file should be included to use the peripheral macro driver within a project. This file is called “Device.h” and include all necessary header files for the specific device. The selection of the device is made in the same file by one define: “MCU_TYPE” should be set to one device string select able from a given list. Is the specific microcontroller not defined in the list, it could be created by copying and modifying from a given similar device.

Table 2-1: Main include files

File name	Description
Device.h	Include file for User Application
Dxxxxx.h	Device SFRs Definitions
DxxxxxextIO.h	Device special SFRs Definitions (e.g. FCAN etc.)
Macro_Dxxxxx.h	Device specific macro defines
Macro.h	Generic macro defines

Include files for all devices which define all standard and special SFRs can be downloaded from NECs web page. These files (Dxxxxx.h, DxxxxxextIO.h) are included by the header file “Device.h”.

The generic macro defines (Macro.h) are valid for all devices including the same peripheral hardware macros. The specific macro defines like port settings etc., are included in a device specific macro header file (Macro_Dxxxxx.h). Only both files together create the peripheral macro driver software.

Table 2-2: Macro include files (included in Macro.h)

File name	User's Manual Chapter	Group
Macro_Ports.h	Port Functions	Ports
Macro_Bus.h	Bus Control Function	Ports
Macro_RTO.h	Real-Time Output Function	Ports
Macro_Interrupt.h	Interrupt/Exception Processing Function	Interrupts
Macro_KeyInt.h	Key Interrupt Function	Interrupts
Macro_Timer_P.h	16-Bit Timer/Event Counter P	Timers
Macro_Timer_Q.h	16-Bit Timer/Event Counter Q	Timers
Macro_Timer_M.h	16-Bit Interval Timer M	Timers
Macro_Watchtimer.h	Watch Timer Functions	Timers
Macro_Watchdog.h	Functions Of Watchdog Timer 2	Timers
Macro_UART_A.h	Asynchronous Serial Interface A (UARTA)	Serial
Macro_CSI_B.h	3-Wire Variable-Length Serial I/O (CCSIB)	Serial
Macro_IIC.h	I²C Bus	Serial
Macro_FCAN.h	FCAN not supported	Serial
Macro_IEBus.h	IE-Bus not supported	Serial
Macro_Clk_Pwr.h	Clock Generation Function	Clock & Power
	Stand-by Function	Clock & Power
	Clock Monitor	Clock & Power
	Reset Function	Clock & Power
	Low-Voltage Detector	Clock & Power
	RAM Retention Voltage Detection Operation	Clock & Power
Macro_AD.h	A/D Converter	Analog
Macro_DA.h	D/A Converter	Analog
Macro_DMA.h	DMA Controller	Miscellaneous
Macro_OCD.h	On Chip Debug Interface	Miscellaneous
Macro_CRC.h	CRC Function	Miscellaneous
Macro_ROMcorr.h	ROM Correction Function	Miscellaneous

Chapter 3 API Description

3.1 Overview Application Programming Interface

This document is not a replacement for the description in the User's Manual of all the peripherals. It should help the User to initialize and handle the peripherals. The User's Manual is the basic description for each peripheral. This document describe only the usage of the peripheral macro driver.

3.1.1 Ports - Port Functions

API	Short description	Return
Port_Init (port, value, direction)	Set value and direction	-
Port_Pull-up (port, pull-up)	Connect/Disconnect internal pull-up resistor	-
Port_Direction (port, direction)	Set port direction	-
Port_Read (port)	Read port	port value
Port_Write (port, value)	Write port	-
Port_Pin_Read (port, pin)	Read port pin	pin level
Port_Pin_Write (port, pin, value)	Write port pin	-

3.1.2 Ports - Bus Control Function

API	Short description	Return
BUS_Init_bus_mode_global (DA_bus_mode)	Set multiplexed mode (AD0-AD15, A16-An) or separate mode (D0-D15, A0-An)	-
BUS_Init_mode_specific (bus_width, chip_select)	Initialize bus width, DA bus and chip select port Initialize chip select port pin Set bus with for specific chip select.	-
BUS_Init_Waits(data_waits,addr_setup_wait,addr_hold_wait,idle_states,chip_select)	Initialise data wait states, address setup waits, address hold waits and idle states	-
BUS_Init_Port_ASTB()	Only applicable if multiplexed bus mode is used (BUS_Multiplexed_Mode)	-
BUS_Init_Address_Bus_A0_A15 (pattern)	Only applicable if separate bus mode is used (BUS_Separate_Mode)	-
BUS_Init_Address_Bus_A16_An (pattern)	Initialize address bus	-
BUS_Init_Bus_AD0_AD7()	Initialize Data bus 8 bit (BUS_Width_8_Bit)	-
BUS_Init_Bus_AD0_AD15()	Initialize Data bus 16 bit (BUS_Width_16_Bit)	-
BUS_Init_Port_WR0()	Initialize Write low strobe signal	-
BUS_Init_Port_WR1()	Initialize Write high strobe signal	-
BUS_Init_Port_RD()	Initialize Read strobe signal	-
BUS_Init_Port_CLKOUT()	Initialize Internal system clock output	-
BUS_Init_Port_WAIT()	Initialize External wait control	-
BUS_Init_Port_HLDRQ()	Initialize Bus hold control	-
BUS_Init_Port_HLDAK()	Initialize Bus hold control	-

3.1.3 Ports - Real-Time Output Function

API	Short description	Return
RTO_Init (channels, pattern, trigger_source, trigger_edge)	Initialize RTO; Trigger Timer is not initialized	-
RTO_Enable()	Enable/Disable RTO	-
RTO_Disable()		
RTO_Write (channels, value)	Write new value to RTO buffer	-

3.1.4 Interrupts - Interrupt/Exception Processing Function

API	Short description	Return
Get_Interrupt_Request_Flag (Interrupt name)	Returns value of interrupt request flag	flag
Clear_Interrupt_Request_Flag (Interrupt name)	Clears interrupt request flag	-
Set_Interrupt_Trigger (Interrupt name, trigger type)	Sets trigger for external interrupts: rising/falling edge or both edges are possible	-
Set_Interrupt_Priority (Interrupt name, priority)	Priority is set for any defined maskable interrupt. (Except NMI) **Caution: "priority" MUST be between 0 and 7!**	-
Enable_Interrupt (Interrupt name)	Enable/Disable any defined maskable interrupt.	-
Disable_Interrupt (Interrupt name)		
Enable_All_Interrupts()	Enable maskable interrupts global.	-
Disable_All_Interrupts()	Disable maskable interrupts global.	-

3.1.5 Interrupts - Key Interrupt Function

API	Short description	Return
KeyInt_Init (pattern)	Initialize key return unit. Pattern define used input pins	-
KeyInt_Disable()	Disable key return unit	-

3.1.6 Timers - 16-Bit Timer/Event Counter P

Table 3-1: Timers - 16-Bit Timer/Event Counter P (1/2)

API	Short description	Return
TM_P_Interval_Mode_Init (timer, clock, interval_value, interval_output)	Initialize the basic configuration of the timer in the interval timer mode	-
TM_P_Interval_Mode_Phase_Init (timer, phase_value, phase_output)	Init additionally phase value in the interval mode.	-
TM_P_Interval_Mode_Write_Interval_Value (timer, interval_value)	Update compare values in interval mode	-
TM_P_Interval_Mode_Write_Phase_Value (timer, phase_value)	Write phase value in the interval mode.	-
TM_P_Ext_Evt_Cnt_Mode_Init (timer, input, input_edge, event_count_value, event_count_output)	Initialize the basic configuration of the timer in the external event count mode	-
TM_P_Ext_Evt_Cnt_Mode_Pre_Evt_Cnt_Init (timer, pre_event_count_value, pre_event_count_output)	Init additionally pre event count value	-
TM_P_Ext_Evt_Cnt_Mode_Write_Evt_Cnt_Value (timer, event_count_value)	Update compare values in external event count mode	-
TM_P_Ext_Evt_Cnt_Mode_Write_Pre_Evt_Cnt_Value (timer, pre_event_count_value)	Write additionally pre event count value	-
TM_P_Trg_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the trigger pulse mode	-
TM_P_Trg_Pulse_Mode_Duty_Init (timer, duty_value, duty_output)	Init additionally duty cycle	-
TM_P_Trg_Pulse_Mode_Write_Values (timer, cycle_value, duty_value)	Update compare values in trigger pulse mode	-
TM_P_Trg_Pulse_Mode_Write_Duty_Value (timer, duty_value)	Write duty cycle value	-
TM_P_Trg_Pulse_Mode_Write_SW_Trigger (timer)	Write sw trigger pulse as external trigger input in trigger pulse mode	-
TM_P_One_Shot_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the one shot pulse mode	-
TM_P_One_Shot_Pulse_Mode_Delay_Init (timer, delay_value, delay_output)	Init additionally delay time	-
TM_P_One_Shot_Pulse_Mode_Write_Cycle_Value (timer, cycle_value)	Update compare values in one shot pulse mode	-
TM_P_One_Shot_Pulse_Mode_Write_Delay_Value (timer, delay_value)	Write delay time	-
TM_P_One_Shot_Pulse_Mode_Write_SW_Trigger (timer)	Write sw trigger pulse as external trigger input in one shot pulse mode	-
TM_P_PWM_Mode_Clk_Internal_Init (timer, clock, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the pwm mode - Internal clock	-
TM_P_PWM_Mode_Clk_External_Init (timer, input, input_edge, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the pwm mode - External clock	-
TM_P_PWM_Mode_PWM_Init (timer, pwm_value, pwm_output)	Set PWM value	-
TM_P_PWM_Mode_Write_Values (timer, cycle_value, pwm_value)	Update compare values in PWM mode	-
TM_P_PWM_Mode_Write_PWM_Value (timer, pwm_value)	Set PWM value	-

Table 3-1: Timers - 16-Bit Timer/Event Counter P (2/2)

API	Short description	Return
TM_P_Free_Running_Mode_Clk_Internal_Init (timer, clock)	Initialize the basic configuration of the timer in the free running mode - Internal clock	-
TM_P_Free_Running_Mode_Clk_External_Init (timer, input, input_edge)	Initialize the basic configuration of the timer in the free running mode - External clock	-
TM_P_Free_Running_Mode_Compare_Init (timer, cap_cmp_selection, cmp_value, output)	Init compare register	-
TM_P_Free_Running_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)	Init capture register	-
TM_P_Free_Running_Mode_Write_Compare (timer, cap_cmp_selection, cmp_value)	Update compare values in free running mode	-
TM_P_Pulse_Width_Measure_Mode_Init (timer, clock)	Initialize the basic configuration of the timer in the pulse width measurement mode	-
TM_P_Pulse_Width_Measure_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)	Init additionally capture mode	-
TM_P_Capture_Read (timer, cap_cmp_selection)	Read capture value	16-bit
TM_P_Overflow_Read (timer)	Read overrun flag	flag
TM_P_Counter_Read (timer)	Read timer counter value	16-bit
TM_P_Enable (timer)	Enable/Disable timer	-
TM_P_Disable (timer)		
TM_P_Input_Special (special_input, special_input_set)	Special inputs; Device specific	-
TM_P_Sync_Mode (timer, sync_mode)	Sync mode; Device specific	-

3.1.7 Timers - 16-Bit Timer/Event Counter Q

Table 3-2: Timers - 16-Bit Timer/Event Counter Q (1/2)

API	Short description	Return
TM_Q_Interval_Mode_Init (timer, clock, interval_value, interval_output)	Initialize the basic configuration of the timer in the interval timer mode	-
TM_Q_Interval_Mode_Phase_Init (timer, phase_selection, phase_value, phase_output)	Init additionally phase value in the interval mode.	-
TM_Q_Interval_Mode_Write_Interval_Value (timer, interval_value)	Update compare values in interval mode	-
TM_Q_Interval_Mode_Write_Phase_Value (timer, phase_selection, phase_value)	Write phase value in the interval mode.	-
TM_Q_Ext_Evt_Cnt_Mode_Init (timer, input, input_edge, event_count_value, event_count_output)	Initialize the basic configuration of the timer in the external event count mode	-
TM_Q_Ext_Evt_Cnt_Mode_Pre_Evt_Cnt_Init (timer, pre_event_selection, pre_event_count_value, pre_event_count_output)	Init additionally pre event count value	-
TM_Q_Ext_Evt_Cnt_Mode_Write_Evt_Cnt_Value (timer, event_count_value)	update compare values in external event count mode	-
TM_Q_Ext_Evt_Cnt_Mode_Write_Pre_Evt_Cnt_Value (timer, pre_event_selection, pre_event_count_value)	Write pre event count value	-
TM_Q_Trg_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the trigger pulse mode	-
TM_Q_Trg_Pulse_Mode_Duty_Init (timer, duty_selection, duty_value, duty_output, duty1_value)	Init additionally duty cycle	-
TM_Q_Trg_Pulse_Mode_Write_Values (timer, cycle_value, duty1_value, duty2_value, duty3_value)	Update compare values in trigger pulse mode	-
TM_Q_Trg_Pulse_Mode_Write_Duty_Value (timer, duty_selection, duty_value, duty1_value)	Write duty cycle	-
TM_Q_Trg_Pulse_Mode_Write_SW_Trigger (timer)	Write sw trigger pulse as external trigger input in trigger pulse mode	-
TM_Q_One_Shot_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the one shot pulse mode	-
TM_Q_One_Shot_Pulse_Mode_Delay_Init (timer, delay_selection, delay_value, delay_output)	Init additionally delay time	-
TM_Q_One_Shot_Pulse_Mode_Write_Cycle_Value (timer, cycle_value)	Update compare values in one shot pulse mode	-
TM_Q_One_Shot_Pulse_Mode_Write_Delay_Value (timer, delay_selection, delay_value)	Write delay time	-
TM_Q_One_Shot_Pulse_Mode_Write_SW_Trigger (timer)	Write sw trigger pulse as external trigger input in one shot pulse mode	-
TM_Q_PWM_Mode_Clk_Internal_Init (timer, clock, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the pwm mode - Internal clock	-
TM_Q_PWM_Mode_Clk_External_Init (timer, input, input_edge, cycle_value, cycle_output)	Initialize the basic configuration of the timer in the pwm mode - External clock	-
TM_Q_PWM_Mode_PWM_Init (timer, pwm_selection, pwm_value, pwm_output, pwm1_value)	Set PWM value	-
TM_Q_PWM_Mode_Write_Values (timer, cycle_value, pwm1_value, pwm2_value, pwm3_value)	Update cycle value and all PWM values	-

Table 3-2: Timers - 16-Bit Timer/Event Counter Q (2/2)

API	Short description	Return
TM_Q_PWM_Mode_Write_PWM_Value (timer, pwm_selection, pwm_value, pwm1_value)	Update compare values in PWM mode	-
TM_Q_Free_Running_Mode_Clk_Internal_Init (timer, clock)	Initialize the basic configuration of the timer in the free running mode - Internal clock	-
TM_Q_Free_Running_Mode_Clk_External_Init (timer, input, input_edge)	Initialize the basic configuration of the timer in the free running mode - External clock	-
TM_Q_Free_Running_Mode_Compare_Init (timer, cap_cmp_selection, cmp_value, output)	Init compare register	-
TM_Q_Free_Running_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)	Init capture register	-
TM_Q_Free_Running_Mode_Write_Compare (timer, cap_cmp_selection, cmp_value)	Update compare values in free running mode	-
TM_Q_Pulse_Width_Measure_Mode_Init (timer, clock)	Initialize the basic configuration of the timer in the pulse width measurement mode	-
TM_Q_Pulse_Width_Measure_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)	Init additionally capture mode	-
TM_Q_Capture_Read (timer, cap_cmp_selection)	Read capture value	16-bit
TM_Q_Overflow_Read (timer)	Read overrun flag	flag
TM_Q_Counter_Read (timer)	read timer counter value	16-bit
TM_Q_Enable (timer)	Enable / Disable timer	-
TM_Q_Disable (timer)		
TM_Q_Input_Special (special_input, special_input_set)	Special inputs; Device specific	-
TM_Q_Sync_Mode (timer, sync_mode)	Sync mode; Device specific	-

3.1.8 Timers - 16-Bit Interval Timer M

API	Short description	Return
TM_M_Interval_Mode_Init (clock, interval_value)	Initialization (Interval mode)	-
TM_M_Enable()	Enable/start timer	-
TM_M_Disable()	Disable/stop timer	-
TM_M_Set_Interval (interval_value)	Modify interval time	-

3.1.9 Timers - Watch Timer Functions

API	Short description	Return
Watch timer_Init (clock_source, clock_watch, clock_interval)	Clock initialization of watch timer and watch interval timer.No start of timers	-
Watch timer_Enable()	Enable/Disable Watch timer register; Does not enable clock supply to watch timer.	-
Watch timer_Disable()		
Watch timer_Enable_Clk_Supply()	Enable/Disable Watch timer and Watch interval timer clock supply. Watch timer need additional enable/disable. Watch interval timer directly start/stop.	-
Watch timer_Disable_Clk_Supply()		
Watch timer_Adjust()	Reset watch timer register; Same as Disable + Enable	-
Watch timer_Set_Prescaler (prescaler, divisor)	Optional prescaler if sub clock is not used as clock source; Device specific	-

3.1.10 Timers - Functions of Watchdog Timer 2

API	Short description	Return
WatchDog_Set (mode, clock)	Init watchdog clock supply without starting	-
WatchDog_Start ()	Start Watchdog	-
WatchDog_Reset()	Reset Watchdog	-
WatchDog_Generate_Overflow()	Generate Watchdog overflow -> Stop/NMI/Reset	-

3.1.11 Serial - Asynchronous Serial Interface A (UARTA)

API	Short description	Return
UART_A_Init (channel, mode, direction, oscillator, baudrate, bit length, parity, stop bits)	Initialize UART channel	-
UART_A_Disable (channel)	Disable/stop channel	-
UART_A_TxEnable (channel)	Enable / Disable transmission on channel	-
UART_A_TxDisable (channel)		
UART_A_RxEnable (channel)	Enable / Disable reception on channel	-
UART_A_RxDisable (channel)		
UART_A_Tx_Status_Read (channel)	Read transfer status flag (transfer in progress)	flag
UART_A_Rx_Overrun_Error_Read_Clear (channel)	Read and clear overrun error status	flag
UART_A_Rx_Parity_Error_Read_Clear (channel)	Read and clear parity error status	flag
UART_A_Rx_Framing_Error_Read_Clear (channel)	Read and clear framing error status	flag
UART_A_Tx_Write_Byte (channel, value)	Write byte to transmit data register	-
UART_A_Rx_Read_Byte (channel)	Read byte from receive data register	value 8-bit

3.1.12 Serial - 3-Wire Variable-Length Serial I/O (CCSIB)

API	Short description	Return
CSI_B_Init (channel, clock_source, relax_level, active_edge, mode, direction, bit length)	Initialize CSI	-
CSI_B_Set_BRG (channel, prescaler, divisor)	Set baud rate generator	-
CSI_B_Disable (channel)	Reset to default values	-
CSI_B_Stop (channel)	Stop SIO (Re-initialization have to be made using CSI_B_Init	-
CSI_B_Transfer_Clock_Output_Enable (channel)	Enable / Disable transfer clock output	-
CSI_B_Transfer_Clock_Output_Disable (channel)		
CSI_B_Write_16bit (channel, value_16bit)	Write 16 bit value to transmission register	-
CSI_B_Write_8bi t (channel, value_8bit)	Write 8 bit value to transmission register	-
CSI_B_Read_16bit (channel)	Read 16 bit value from reception register	16-bit value
CSI_B_Read_8bit (channel)	Read 8 bit value from reception register	8-bit value
CSI_B_Operation (channel)	Read operating status	flag
CSI_B_Overrun (channel)	Read overrun error status	flag

3.1.13 Serial - I2C Bus

API	Short description	Return
IIC_Init (channel, clock, filter)	Basic initialization of I²C peripheral	-
IIC_Enable (channel)	Enable/Disable IIC module and reset status registers	-
IIC_Disable (channel)		
IIC_Generate_Start (channel)	Generate start condition	-
IIC_Generate_Stop (channel)	Generate stop condition	-
IIC_Acknowledge_Enable (channel)	Enable/Disable acknowledgment of received data	-
IIC_Acknowledge_Disable (channel)		
IIC_Wait_and_Int_8th_clock (channel)	Control of wait and Interrupt request generation	-
IIC_Wait_and_Int_9th_clock (channel)		
IIC_Stop_Interrupt_Enable (channel)	Enable/disable generation of interrupt request when stop condition is detected	-
IIC_Stop_Interrupt_Disable (channel)		
IIC_Cancel_Wait (channel)	Wait cancellation control	-
IIC_Exit_Communication (channel)	Exit from communications	-
IIC_Get_Status_Master (channel)	Master device status	flag
IIC_Get_Status_Arbitration (channel)	Arbitration loss detection	flag
IIC_Get_Status_Extension (channel)	Detection of extension code reception	flag
IIC_Get_Status_Addr_Match (channel)	Matching slave addresses detection	flag
IIC_Get_Status_Direction (channel)	Transmit/receive status detection	flag
IIC_Get_Status_Acknowledge (channel)	Acknowledge detection	flag
IIC_Get_Status_Start (channel)	Start condition detection	flag
IIC_Get_Status_Stop (channel)	Stop condition detection	flag
IIC_Get_Status_Start_Issued (channel)	Start issued/rejected	flag
IIC_Get_Status_Bus (channel)	IIC bus status (released/communication)	flag
IIC_Start_If_No_Stop (channel)	Initial start enable trigger	-
IIC_No_Start_If_No_Stop (channel)		
IIC_Com_Reservation_Enable (channel)	Enable / Disable Communication Reservation Function	-
IIC_Com_Reservation_Disable (channel)		
IIC_Digital_Filter_Enable (channel)	Enable / Disable Digital Filter	-
IIC_Digital_Filter_Disable (channel)		
IIC_Get_Status_SDA_Level (channel)	Get current level of SDA/SCL line	level 1-bit
IIC_Get_Status_SCL_Level (channel)		
IIC_Write_Data (channel, value)	Write data	-
IIC_Read_Data (channel)	Read data	8 / 16 bit
IIC_Set_Slave_Address (channel, address)	Set slave address	-

3.1.14 Clock & Power - Clock Generation Function

API	Short description	Return
Clock_Set_CPU_Clock (clock, main_feedback, sub_feedback, main_operation)	Set clock for CPU	-
Clock_Get_CPU_Clock_Status ()	Read Status of CPU clock (f_{CPU})	Enum
Clock_Set_PLL (PLL_operation, PLL_multiplier, PLL_Lockup_Time)	Configure PLL; (Clock base for CPU and peripherals)	-
Clock_Get_PLL_Status()	Get Status of PLL	flag
Clock_Output_Enable (clock)	Enable / Disable clock output	-
Clock_Output_Disable()		
Clock_Ring_Oscl_Enable()	Enable / Disable Ring-Oscillator	-
Clock_Ring_Oscl_Disable()		

3.1.15 Clock & Power - Standby Function

API	Short description	Return
Standby_Enter (standby_mode, release_condition)	Entering a stand-by mode (IDLE1, IDLE2, SW_STOP)	-
Oscl_Stabil_Time_Set (ost)	Set Stabilization Oscillation Time	-

3.1.16 Clock & Power - Clock Monitor

API	Short description	Return
Clock_Monitor_Enable()	Enable Clock Monitor. Disabling not possible -> Reset	-

3.1.17 Clock & Power - Reset Function

API	Short description	Return
Reset_Get_Source()	Get information about reset source	value
Reset_Clear_Source()	Clear information about reset source	-

3.1.18 Clock & Power - Low-Voltage Detector

API	Short description	Return
LVI_Init (operation, generation, level)	Initialize Low Voltage Detector	-
LVI_Disable()	Set default values	-
LVI_Get_Status()	Status of voltage detection	flag

3.1.19 Clock & Power - RAM Retention Voltage Detection Operation

API	Short description	Return
LVI_RAM_Get_Status()	The flag indicates whether the internal RAM is valid or not.	flag
LVI_RAM_Clear_Status()	Clear the flag	-
LVI_RAM_Set_Invalid()	Emulate: RAM status flag set	-
LVI_RAM_Set_Valid()	Emulate: RAM status flag clear	-

3.1.20 Analog - A/D Converter

API	Short description	Return
AD_Init (mode, trigger_type, trigger_edge, time)	Initialization of A/D converter; (Timer not initialized as src. if trigger_type=AD_Timer_Trigger)	-
AD_Enable()	Enable / Disable A/D converter operation	-
AD_Disable()		
AD_Select_Channel (channel)	Select channel for conversion result(s)	-
AD_Get_Status()	Get status (Idle, Operating)	flag
AD_Get_Result_10bit(channel)	Get 10 bit result	16-bit
AD_Get_Result_8bit(channel)	Get 8 bit result	8-bit
AD_PowerFail_Init (selection, value)	Initialize power-fail compare mode (Enabling must be done separately)	-
AD_PowerFail_Enable()	Enable / Disable power-fail comparison	-
AD_PowerFail_Disable()		
AD_PowerFail_Set_Value (value)	Set value for power-fail comparison	-

3.1.21 Analog - D/A Converter

API	Short description	Return
DA_Init (channel, mode, value)	Initialize DA converter	-
DA_Enable (channel)	Enable/Disable DA converter channel. Do not (re)set port pin settings.	-
DA_Disable (channel)		
DA_Write_Value (channel, value)	Write value to DA converter.	-

3.1.22 Miscellaneous - DMA Controller

API	Short description	Return
DMA_Init (channel, src_address, src_location, src_direction, dest_address, dest_location, dest_direction, count, data_size, trigger_src)	Initialize DMA controller channel	-
DMA_Enable (channel)	Enable/Disable DMA transfer	-
DMA_Disable (channel)		
DMA_Trigger_By_Software (channel)	Enable and start DMA transfer by software	-
DMA_Read_Status (channel)	Read DMA status (finished / in progress)	flag
DMA_Terminal_Count (channel)	Read DMA status (finished / in progress) Same as above	flag
DMA_Clear_Request (channel)	Clear DMA transfer request	-

3.1.23 Miscellaneous - On Chip Debug Interface

API	Short description	Return
OCD_Enable()	Enable / Disable On_chip debugging	-
OCD_Disable()		

3.1.24 Miscellaneous - CRC Function

API	Short description	Return
CRC_Init()	Initialize CRC unit	-
CRC_Write_Next (data)	Write next 16-bit data for CRC calculation	-
CRC_Read_Result()	Read result of CRC calculation	16-bit

3.1.25 Miscellaneous - ROM Correction Function

API	Short description	Return
ROM_Correction_Init (channel, address)	Initialize Rom Correction Function	-

3.2 Detailed Application Programming Interface

3.2.1 Ports - Port Functions

(1) Parameters

(a) port

Valid port name. E.g. P0, P7L, PDL

(b) direction

A set bit level configures the port pin to input, while a zero set the pin for output.
The maximum valid bit number is defined by the port.
1 = Input (default)
0 = Output

(c) pull-up

A set bit level connect the port pin to the pull-up resistor, while a zero set the pin to default
0 = No pull-up resistor connected (default)
1 = Pull-up resistor connected

(d) pin

Any valid pin/bit of the port.

(e) value

Any 8-/16-bit value
The maximum valid bit number is defined by the port.

(2) Function description

(a) Port_Init (port, value, direction)

Set value and direction of the port for each pin.

(b) Port_Pull-up (port, pull-up)

Connect/Disconnect internal pull-up resistor for each pin.

(c) Port_Direction (port, direction)

Set port direction for each pin.

(d) Port_Read (port)

Return: Port levels in case of input pins, or port register levels in case of output pins.
The width is dependent of the port width and port name.

(e) Port_Write (port, value)

Write port

(f) Port_Pin_Read (port, pin)

Return: Port level in case of input pins, or port register level in case of output pins.
The width is dependent of the port width and port name.

(g) Port_Pin_Write (port, pin, value)

Write port pin

3.2.2 Ports - Bus Control Function

(1) Parameters

(a) DA_bus_mode

Choose multiplexed or separate bus mode. (Value for EXIMC register)
Following settings are possible:

- BUS_Multiplexed_Mode
- BUS_Separate_Mode

(b) bus_width

Choose bus width. (Value for BSC register)
Following settings are possible:

- BUS_Width_8_Bit
- BUS_Width_16_Bit

(c) addr_setup_wait

Choose address setup wait setting. (Value for ASC register)
Following settings are possible:

- BUS_Addr_No_Setup_Wait
- BUS_Addr_Use_Setup_Wait

(d) addr_hold_wait)

Choose address hold wait setting. (Value for ASC register)
Following settings are possible:

- BUS_Addr_No_Hold_Wait
- BUS_Addr_Use_Hold_Wait

(e) idle_state

Choose idle wait setting. (Value for BCC register)
Following settings are possible:

- BUS_No_Idle_State
- BUS_Use_Idle_State

(f) data_waits

Choose data wait setting. (Value for DWC0/1 register)
Following settings are possible:
BUS_Data_Waits_0, BUS_Data_Waits_1, BUS_Data_Waits_2, BUS_Data_Waits_3, BUS_Data_Waits_4, BUS_Data_Waits_5, BUS_Data_Waits_6, BUS_Data_Waits_7

(2) Function description

(a) BUS_Init_bus_mode_global(DA_bus_mode)

Set multiplexed mode (AD0-AD15, A16-An) or separate mode (D0-D15, A0-An) for the ext. bus.
This setting is valid for the whole external area.
The initialization of the bus and control bins must be done separately.

(b) BUS_Init_mode_specific(bus_width,chip_select)

Initialize bus width, DA bus and chip select port.
Initialize chip select port pin.
Set bus with for specific chip select.

(c) BUS_Init_Waits(data_waits,addr_setup_wait,addr_hold_wait,idle_states,chip_select)

Initialise data wait states, address setup waits, address hold waits and idle states.
This setting is valid for only the chosen chip select signal.

(d) BUS_Init_Port_ASTB()

Initialize address strobe pin.
Only applicable if multiplexed bus mode is used (BUS_Multiplexed_Mode).
This configures the pin for this alternate function.

(e) BUS_Init_Address_Bus_A0_A15(pattern)

A set bit level configure the corresponding address pin as alternate function (address output).
Only applicable if separate bus mode is used (BUS_Separate_Mode)

(f) BUS_Init_Address_Bus_A16_An(pattern)

Initialize address bus
A set bit level configure the corresponding address pin as alternate function (address output).

(g) BUS_Init_Bus_AD0_AD7()

Initialize Data bus 8 bit (BUS_Width_8_Bit)
This configures the lower eight bit of the 16-bit bus for this alternate function. The upper eight bit remain their previous setting (I/O port).

(h) BUS_Init_Bus_AD0_AD15()

Initialize Data bus 16 bit (BUS_Width_16_Bit).This configures the whole 16-bit bus for this alternate function.

(i) BUS_Init_Port_WR0()

Initialize Write low strobe signal for output.

(j) BUS_Init_Port_WR1()

Initialize Write high strobe signal for output.

(k) BUS_Init_Port_RD()

Initialize Read strobe signal for output.

(l) BUS_Init_Port_CLKOUT()

Initialize Internal system clock output. The clock signal can be used to supply external devices.

(m) BUS_Init_Port_WAIT()

Initialize External wait control. An external device could shortly hold the bus.

(n) BUS_Init_Port_HLDRQ()

Initialize Bus hold control. An external device can use the bus as master. This signal is a request for mastering.

(o) BUS_Init_Port_HLDAK()

Initialize Bus hold control. An external device can use the bus as master. This signal is the acknowledge to the request for mastering. The external device can now use the bus as master.

3.2.3 Ports - Real-Time Output Function

(1) Parameters

(a) trigger_source

Choose timer to trigger RTO.
Following settings are possible:

- RTO_2Bit_P5_4Bit_P4
- RTO_2Bit_P4_4Bit_P0
- RTO_6Bit_P0
- RTO_6Bit_P4

(b) trigger_edge

Choose trigger edge for RTO.
Following settings are possible:

- RTO_Edge_Falling
- RTO_Edge_Rising

(c) channels

Choose channel for data output.
Following settings are possible:

- RTO_2Bit_Channel, RTO_4Bit_Channel, RTO_6Bit_Channel

(d) pattern

A set bit level configure the corresponding pin as alternate function (RTO).

(2) Function description

(a) RTO_Init (channels, pattern, trigger_source, trigger_edge)

Initialize RTO; Trigger timer is not initialized. The timer interrupt request will trigger the copy from the buffer register to the output port.
The “pattern” choose the port pins which should be used for this function. All other pins (low level in “patter”) will remain their function.

(b) RTO_Enable() / RTO_Disable()

Enable/Disable RTO

(c) RTO_Write (channels, value)

Write new value to RTO buffer. The value is not directly transferred to the port. Only the trigger timer will initiate a copy.

3.2.4 Interrupts - Interrupt/Exception Processing Function

(1) Parameters

(a) Interrupt name

The valid interrupt source names are device specific.

- e.g. V850ES/FJ2: INTLVI, INTP0, INTP1, INTP2, INTP3, INTP4, INTP5, INTP6, INTP7, INTTQ0OV, INTTQ0CC0, INTTQ0CC1, INTTQ0CC2,..., INTC3REC, INTC3TRX

(b) trigger type

Following settings are possible:

- INT_NO_EDGE, INT_RISING_EDGE, INT_FALLING_EDGE, INT_BOTH_EDGES

(c) priority

The highest priority has the value 0. The lowest priority has the value 7.
In human neighbourhood the lowest value should be zero while the higher values are counting up. Because of this, the following constants have a vice versa count direction.
Following settings are possible:

- INT_LEVEL_0=7, INT_LEVEL_1, INT_LEVEL_2, INT_LEVEL_3, INT_LEVEL_4, INT_LEVEL_5, INT_LEVEL_6, INT_LEVEL_7=0
- INT_LOWEST (= INT_LEVEL0)
- INT_HIGHEST (= INT_LEVEL7)

(2) Function description

(a) Get_Interrupt_Request_Flag (Interrupt name)

Return: Level of interrupt request flag
This macro can be used to poll an interrupt flag.

(b) Clear_Interrupt_Request_Flag (Interrupt name)

Clears interrupt request flag.
The interrupt source or the software can set the flag again to request an interrupt.

(c) Set_Interrupt_Trigger (Interrupt name, trigger type)
Sets trigger for external interrupts: rising/falling edge or both edges are possible

(d) Set_Interrupt_Priority (Interrupt name, priority)
Priority is set for any defined maskable interrupt. (Except NMI)

Caution: The value of “priority” must be between 0 and 7!

(e) Enable_Interrupt (Interrupt name) / Disable_Interrupt (Interrupt name)

Enable/Disable any defined maskable interrupt.

(f) Enable_All_Interrupts() / Disable_All_Interrupts()

Enable / Disable maskable interrupts global. The interrupt controller will be enabled or disabled. The setting by Enable_Interrupt(...)/Disable_Interrupt(...) is unaffected by these macros. All other macros for the interrupt controller can be made independently.It does not modify the interrupt control register of any maskable interrupt source.
The macro is equal to the “ei” or “di” assembly instruction.

3.2.5 Interrupts - Key Interrupt Function

(1) Parameters

(a) pattern

A set bit level configure the corresponding pin as alternate function (Key Interrupt pin).

(2) Function description

(a) KeyInt_Init (pattern)

Initialize key return unit. The corresponding pin to a high bit in the parameter “pattern” will be configured to accept a falling edge. A falling edge will generate the INTKR interrupt request.
The interrupt must be enabled separately. See 3.1.4 on page 17 for details.

(b) KeyInt_Disable()

Disable key return unit.
Does not reset the port pin settings.

3.2.6 Timers - 16-Bit Timer/Event Counter P

(1) Parameters

(a) timer

Choose timer macro.

Caution: Device specific timer count.

- e.g. V850ES/FJ2: TM_P_0=0, TM_P_1=1, TM_P_2=2, TM_P_3=3

(b) clock_source

Choose clock source for timer.

Caution: Device and timer specific clock sources. Not all combinations are possible.

- e.g. V850ES/FJ2: TM_P_Clk_FXX, TM_P_Clk_FXX_2, TM_P_Clk_FXX_4, TM_P_Clk_FXX_8, TM_P_Clk_FXX_16, TM_P_Clk_FXX_32, TM_P_Clk_FXX_64, TM_P_Clk_FXX_128, TM_P_Clk_FXX_FXT, TM_P_Clk_Extern

(c) input_edge, ext_trigger_edge, trigger_edge

Choose valid edge of input.
Following settings are possible:

- TM_P_Input_No_Edge
- TM_P_Input_Falling_Edge
- TM_P_Input_Rising_Edge
- TM_P_Input_Both_Edges

(d) sync_mode

Synchronisation with another timer.

Caution: Device specific. Not all timer can be synchronized.

Following settings are possible:

- TM_P_No_Sync
- TM_P_Sync

(e) interval_output, phase_output, event_count_output, pre_event_count_output, cycle_output, duty_output, delay_output, pwm_output, output

Choose if port pin should be configured as timer output.
Following settings are possible:

- TM_P_Output_No_Output
- TM_P_Output_Normal
- TM_P_Output_Inverted

(f) cap_cmp_selection, phase_selection, pre_event_selection, duty_selection, delay_selection, pwm_selection

Timer P include two capture or compare registers:

- TM_P_Cap_Cmp_0=0, TM_P_Cap_Cmp_1=1

(g) input

Choose if timer input pin should be used for counting, or capturing.

- TM_P_Input_No_Input, TM_P_Input_External_Input

(h) special_input_set

Instead of using an external device pin for capturing or counting, an internal signal can be used as external timer input.

Caution: Device and timer specific.

- TM_P_Special_Input_Reset, TM_P_Special_Input_Set

(i) special_input

Instead of using an external device pin for capturing or counting, an internal signal can be used as external timer input. The timer should be configured for external input.
Choose internal source.

Caution: Device and timer specific. This is an example for V850ES/FJ2.

- TM_P_INPUT_TMP00_CAN0_TSOUT
- TM_P_INPUT_TMP01_CAN1_TSOUT
- TM_P_INPUT_TMP01_TMM_INTTM0EQ
- TM_P_INPUT_TMP10_RXDA0
- TM_P_INPUT_TMP10_RXDA1
- TM_P_INPUT_TMP20_CAN2_TSOUT
- TM_P_INPUT_TMP21_CAN3_TSOUT
- TM_P_INPUT_TMP30_RXDA2
- TM_P_INPUT_TMP31_RXDA3

(2) Function description

The interrupts for each channel must be enabled separately. See 3.1.4 on page 17 for details.

(a) TM_P_Interval_Mode_Init (timer, clock, interval_value, interval_output)

Initialize the basic configuration of the timer in the interval timer mode.

The timer count up starting at zero to the value of the interval value. Then the timer register is reset, an interrupt request (INTTPnCC0) is set and if the output is enabled, the output (TOPn0) toggles. The output toggle each time the interval value matches the timer.

(b) TM_P_Interval_Mode_Phase_Init (timer, phase_value, phase_output)

Init additionally phase value in the interval mode.
This is optional.

The interrupt request INTTPnCC1 is generated when the timer register matches the phase value.
The phase value must be smaller or equal than the interval value, otherwise it would never be reached, because the interval value is the greatest value of the timer.
The output (TOPn1) toggle each time the phase value matches the timer.

(c) TM_P_Interval_Mode_Write_Interval_Value (timer, interval_value)

Update new compare values in interval mode.
The value is directly written to the compare registers (TPnCCR0) of the timer.

(d) TM_P_Interval_Mode_Write_Phase_Value (timer, phase_value)

Write new phase value in the interval mode.
The value is directly written to the compare registers (TPnCCR1) of the timer.

(e) TM_P_Ext_Evt_Cnt_Mode_Init (timer, input, input_edge, event_count_value, event_count_output)

Initialize the basic configuration of the timer in the external event count mode.

Usually, this mode is used to count edges from an external device pin (input = TM_P_Input_External_Input). The edge to count can be specified by “input_edge”. Positive, negative or both edges can be specified. (None is also possible, but less useful.)
The counter start at zero and count up to the “event_count_value”. Then the counter is reset, an interrupt request (INTTPnCC0) is set and if the output is enabled, the output (TOPn0) toggles.
Often, the input pin TIPn0 is shared with the output pin TOPn0 in the devices. In this case, no output can be generated while counting external signals.

(f) TM_P_Ext_Evt_Cnt_Mode_Pre_Evt_Cnt_Init (timer, pre_event_count_value, pre_event_count_output)

Init additionally pre-event count value.
This is optional.

The interrupt request INTTP0CC1 is generated when the counter register matches the pre-event count value.
The pre-event count value must be smaller or equal than the event count value, otherwise it would never be reached, because the event count value is the greatest value of the counter.
The output (TOPn1) toggle each time the pre-event value matches the counter.

(g) TM_P_Ext_Evt_Cnt_Mode_Write_Evt_Cnt_Value (timer, event_count_value)

Update compare values in external event count mode
The value is directly written to the compare registers (TPnCCR0) of the timer.

(h) TM_P_Ext_Evt_Cnt_Mode_Write_Pre_Evt_Cnt_Value (timer, pre_event_count_value)

Write additionally pre-event count value
The value is directly written to the compare register (TPnCCR1) of the timer.

(i) TM_P_Trg_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the trigger pulse mode.

The timer count up by the “clock” source to the “cycle_value”. Then the timer is reset, an interrupt request (INTTPnCC0) is set and if the output is enabled, the output (TOPn0) toggle. The timer run free till an external valid edge is recognized. A valid edge reset the timer again to zero and toggle the timer output (TOPn0). The timer output toggle either upon a match of the cycle value or a valid input edge. (The cycle output TOPn0 is not the triggered pulse. See next macro.)
The width of the generated pulse (TOPn1) is defined by the duty value. The duty value is set by the following macro.
If the timer should be triggered by software, set input = TM_P_Input_No_Input.
Often, the input pin TIPn0 is shared with the output pin TOPn0 in the devices. In this case, no output can be generated while triggering by an external signal.

(j) TM_P_Trg_Pulse_Mode_Duty_Init (timer, duty_value, duty_output)

Init additionally duty cycle.

The width of the generated pulse (duty output - TOPn1) is defined by the duty value. The output TOPn1 is set by a reset of the timer. This is either the case by reaching the cycle value, or recognizing a valid input edge. The output is reset when the timer reaches the duty value.

(k) TM_P_Trg_Pulse_Mode_Write_Values (timer, cycle_value, duty_value)

Update compare values in trigger pulse mode.
The values are reloaded automatically when the previous cycle value resets the timer.

(l) TM_P_Trg_Pulse_Mode_Write_Duty_Value (timer, duty_value)

Write duty cycle value
The value is reloaded automatically when the cycle value resets the timer.

(m) TM_P_Trg_Pulse_Mode_Write_SW_Trigger (timer)

Write sw trigger pulse as external trigger input in trigger pulse mode.

The timer register is reset once and continue operating in the Trigger Pulse Mode. A pulse will be generated if output (TOPn1) is enabled.

(n) TM_P_One_Shot_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the one shot pulse mode.

The timer count up by the “clock” source to the “cycle_value”. Then the timer is reset, an interrupt request (INTTPnCC0) is set and if the output is enabled, the output (TOPn0) toggle. The timer wait till an external valid edge is recognized. A valid edge start the timer and toggle the timer output (TOPn0). The timer output toggle either upon a match of the cycle value or a valid input edge. (The cycle output TOPn0 is not the triggered pulse. See next macro.)
The start of the generated delayed pulse (TOPn1) is defined by the delay value. The delay value is set by the following macro.
If the timer should be triggered by software, set input = TM_P_Input_No_Input.
Often, the input pin TIPn0 is shared with the output pin TOPn0 in the devices. In this case, no output can be generated while triggering by an external signal.

(o) TM_P_One_Shot_Pulse_Mode_Delay_Init (timer, delay_value, delay_output)

Init additionally delay time

The start of the generated delayed pulse (delay output, TOPn1) is defined by the delay value. The output TOPn1 is reset by a reset of the timer. This is the case by reaching the cycle value. The output is set when the timer reaches the delay value. After reaching the delay value, the timer continue counting till it reached the cycle value. Then it reset and wait for a next trigger edge.
While the timer count before he reached the cycle value, every valid input edge is ignored. The external trigger is only processed while the timer wait after it reaches the cycle value.

(p) TM_P_One_Shot_Pulse_Mode_Write_Cycle_Value (timer, cycle_value)

Update compare value in one shot pulse mode
The value is directly written to the compare registers of the timer.

(q) TM_P_One_Shot_Pulse_Mode_Write_Delay_Value (timer, delay_value)

Write delay time
The value is directly written to the compare registers of the timer.

(r) TM_P_One_Shot_Pulse_Mode_Write_SW_Trigger (timer)

Write sw trigger pulse as external trigger input in one shot pulse mode
The timer is started in the Trigger Pulse Mode. A delayed pulse will be generated if delay output TOPn1 is enabled.

(s) TM_P_PWM_Mode_Clk_Internal_Init (timer, clock, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the PWM mode - Internal clock

The PWM is output on the TOPn1 pin. The cycle output TOPn0 toggle upon a match of the timer and the cycle value.
This macro is used when an internal clock is used to supply the timer.

(t) TM_P_PWM_Mode_Clk_External_Init (timer, input, input_edge, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the PWM mode - External clock

The PWM is output on the TOPn1 pin. The cycle output TOPn0 toggle upon a match of the timer and the cycle value.
This macro is used when an external clock is used to supply the timer.
Often, the input pin TIPn0 is shared with the output pin TOPn0 in the devices. In this case, no output can be generated while an external clock supply is used.

(u) TM_P_PWM_Mode_PWM_Init (timer, pwm_value, pwm_output)

Set PWM value and PWM output TOPn1.

(v) TM_P_PWM_Mode_Write_Values (timer, cycle_value, pwm_value)

Update cycle and width values in PWM mode.
The values are reloaded automatically when the previous cycle value resets the timer.

(w) TM_P_PWM_Mode_Write_PWM_Value (timer, pwm_value)

Set PWM value
The value is reloaded automatically when the previous cycle value resets the timer.

(x) TM_P_Free_Running_Mode_Clk_Internal_Init (timer, clock)

Initialize the basic configuration of the timer in the free running mode - Internal clock

The timer run free with the supplied clock. It is reset at 0xFFFF and generate an overflow interrupt request INTTPnOV.

(y) TM_P_Free_Running_Mode_Clk_External_Init (timer, input, input_edge)

Initialize the basic configuration of the timer in the free running mode - External clock

The timer run free with the supplied clock. It is reset at 0xFFFF and generate an overflow interrupt request INTTPnOV.

(z) TM_P_Free_Running_Mode_Compare_Init (timer, cap_cmp_selection, cmp_value, output)

Init compare register.

One of the available capture/compare registers is configured as compare register. When the timer matches the compare value (’cmp_value’), it generates an interrupt request INTTP*nm* for the chosen register *m*. If output is enabled, it toggles the output pin TOP*nm*.

(aa)TM_P_Free_Running_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)

Init capture register

One of the available capture/compare registers is configured as capture register. When a valid edge is recognized at the timer input TIP*nm*, it generates an interrupt request INTTP*nm* for the chosen register *m*. The current timer value is transferred to the capture register *m.* (The timer continue counting up.) The captured values can be read with the TM_P_Capture_Read (timer) macro. If output is enabled, it toggles the output pin TOP*nm*.
Often, the input pin TIPn0 is shared with the output pin TOPn0 in the devices. In this case, no output can be generated while an external trigger signal is used for capturing.

(ab)TM_P_Free_Running_Mode_Write_Compare (timer, cap_cmp_selection, cmp_value)

Update compare value in free running mode.
The compare value is directly reloaded to the compare register of the timer.

(ac)TM_P_Pulse_Width_Measure_Mode_Init (timer, clock)

Initialize the basic configuration of the timer in the pulse width measurement mode.

(ad)TM_P_Pulse_Width_Measure_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)

Init additionally capture mode.

One of the available capture/compare registers is configured as capture register. When a valid edge is recognized at the timer input TIP*nm*, it generates an interrupt request INTTP*nm* for the chosen register *m*. The current timer value is transferred to the capture register *m* and the timer restarts at zero. The captured value can be read with the TM_P_Capture_Read (timer) macro.
If a special input should be used as input source, “input” should be “TM_P_Input_External_Input” and the special input should be selected with the TM_P_Input_Special(...) macro.

Useful for pulse with measurement is configuring one register for capturing at rising edge, the other for falling edge. The captured value in the register of the rising edge represents the time of the low phase of the signal. The captured value in the register of the falling edge represents the time of the high phase of the signal. The cycle of the signal can be calculated by adding both captured values.

(ae)TM_P_Capture_Read (timer, cap_cmp_selection)

Return: 16-bit value of captured timer value

(af) TM_P_Overflow_Read (timer)

Return: Overrun flag

(ag)TM_P_Counter_Read (timer)

Return: Current timer counter value

(ah)TM_P_Enable (timer) / TM_P_Disable (timer)

Enable/Disable timer.
Start/Stop timer.

(ai) TM_P_Input_Special (special_input, special_input_set)

Special inputs; Device specific

Some timers in a device allow a special input for capturing or clock supply. The Timer is configured to use an external signal. This external signal could be at a port pin or an internal redirected signal.
This mode is useful for baud rate measurements or similar purposes.
Please tale a look to the User's Manual for informations about allowed combinations.

(aj) TM_P_Sync_Mode (timer, sync_mode)

Sync mode; Device specific

Some timers in a device allow a synchronization of two timers. One timer is the Master timer and its clock supply and cycle time is copied to the second slave timer. The slave timer can use all of its capture/compare registers while the master timer configures the clock supply and cycle time.
This is for example useful for synchronous multi-channel PWM outputs.
Please take a look to the User's Manual for informations about allowed combinations.

3.2.7 Timers - 16-Bit Timer/Event Counter Q

(1) Parameters

(a) timer

Choose timer macro.

Caution: Device specific timer count.

- e.g. V850ES/FJ2: TM_Q_0=0, TM_Q_1=1, TM_Q_2=2

(b) clock_source

Choose clock source for timer.

Caution: Device and timer specific clock sources. Not all combinations are possible.

- e.g. V850ES/FJ2: TM_Q_Clk_FXX, TM_Q_Clk_FXX_2, TM_Q_Clk_FXX_4, TM_Q_Clk_FXX_8, TM_Q_Clk_FXX_16, TM_Q_Clk_FXX_32, TM_Q_Clk_FXX_64, TM_Q_Clk_FXX_128, TM_Q_Clk_Extern

(c) input_edge, ext_trigger_edge, trigger_edge

Choose valid edge of input.
Following settings are possible:

- TM_Q_Input_No_Edge
- TM_Q_Input_Falling_Edge
- TM_Q_Input_Rising_Edge
- TM_Q_Input_Both_Edges

(d) interval_output, phase_output, event_count_output, pre_event_count_output, cycle_output, duty_output, delay_output, pwm_output, output

Choose if port pin should be configured as timer output.
Following settings are possible:

- TM_Q_Output_No_Output
- TM_Q_Output_Normal
- TM_Q_Output_Inverted

(e) cap_cmp_selection, phase_selection, pre_event_selection, duty_selection, delay_selection, pwm_selection

Timer Q include four capture or compare registers.
Following constants can be used to choose one register:

- TM_Q_Cap_Cmp_0=0, TM_Q_Cap_Cmp_1=1, TM_Q_Cap_Cmp_2=2, TM_Q_Cap_Cmp_3=3

TM_Q_Cap_Cmp_0 often has a special purpose. It defines the cycle time for example in many timer modes. Please take a look to the function description to confirm if register zero can be chosen. TM_Q_Cap_Cmp_1 to TM_Q_Cap_Cmp_3 can usually be used freely.

(f) input

Choose if timer input pin should be used for counting, or capturing.

- TM_Q_Input_No_Input, TM_Q_Input_External_Input

(g) sync_mode
Synchronisation with another timer.

Caution: Device specific. Not all timer can be synchronized.

Following settings are possible:

- TM_Q_No_Sync
- TM_Q_Sync

(2) Function description

(a) TM_Q_Interval_Mode_Init (timer, clock, interval_value, interval_output)

Initialize the basic configuration of the timer in the interval timer mode.

The timer count up starting at zero to the value of the interval value. Then the timer register is reset, an interrupt request (INTTQnCC0) is set and if the output is enabled, the output (TOQn0) toggles. The output toggle each time the interval value matches the timer.

(b) TM_Q_Interval_Mode_Phase_Init (timer, phase_selection, phase_value, phase_output)

Init additionally phase value in the interval mode.
This is optional.

While the first compare register (0) is used for the interval time, three compare registers are left for the phase. “phase_selection” must be between TM_Q_Cap_Cmp_1 and TM_Q_Cap_Cmp_3 (*m*=1-3).
The interrupt request INTTQnCC*m* is generated when the timer register matches the phase value. The phase value must be smaller or equal than the interval value, otherwise it would never be reached, because the interval value is the greatest value of the timer.
The output (TOQ*nm*) toggle each time the phase value matches the timer.

(c) TM_Q_Interval_Mode_Write_Interval_Value (timer, interval_value)

Update compare values in interval mode
The value is directly written to the compare registers (TQnCCR0) of the timer.

(d) TM_Q_Interval_Mode_Write_Phase_Value (timer, phase_selection, phase_value)

Write phase value in the interval mode.
The value is directly written to the compare registers (TQnCCRm) of the timer.

(e) TM_Q_Ext_Evt_Cnt_Mode_Init (timer, input, input_edge, event_count_value, event_count_output)

Initialize the basic configuration of the timer in the external event count mode.

Usually, this mode is used to count edges from an external device pin
(input = TM_P_Input_External_Input). The edge to count can be specified by “input_edge”. Positive, negative or both edges can be specified. (None is also possible, but less useful.)

The counter start at zero and count up to the “event_count_value”. Then the counter is reset, an interrupt request (INTTQnCC0) is set and if the output is enabled, the output (TOQn0) toggles.
Often, the input pin TIQn0 is shared with the output pin TOQn0 in the devices. In this case, no output can be generated while counting external signals.

(f) TM_Q_Ext_Evt_Cnt_Mode_Pre_Evt_Cnt_Init (timer, pre_event_selection, pre_event_count_value, pre_event_count_output)

Init additionally pre-event count value.
This is optional.

While the first compare register (0) is used for the event time, three compare registers are left for the pre-event. The value of “pre_event_selection” should be between TM_Q_Cap_Cmp_1 and TM_Q_Cap_Cmp_3 (*m*=1-3).
The interrupt request INTTQ0CCm is generated when the counter register matches the pre-event count value.
The pre-event count values must be smaller or equal than the event count value, otherwise they would never be reached, because the event count value is the greatest value of the counter.
The output (TOQnm) toggle each time the pre-event value matches the counter.

(g) TM_Q_Ext_Evt_Cnt_Mode_Write_Evt_Cnt_Value (timer, event_count_value)

Update compare values in external event count mode
The value is directly written to the compare registers of the timer.

(h) TM_Q_Ext_Evt_Cnt_Mode_Write_Pre_Evt_Cnt_Value (timer, pre_event_selection, pre_event_count_value)

Write additionally pre-event count value
The value is directly written to the compare registers of the timer.

TM_Q_Cap_Cmp_0 (m=0) should not be used for “pre_event_selection”, because this register define the event count value.

(i) TM_Q_Trg_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the trigger pulse mode.

The timer count up by the “clock” source to the “cycle_value”. Then the timer is reset, an interrupt request (INTTQnCC0) is set and if the output is enabled, the output (TOQn0) toggle. The timer run free till an external valid edge is recognized. A valid edge reset the timer again to zero and toggle the timer output (TOQn0). The timer output toggle either upon a match of the cycle value or a valid input edge. (The cycle output TOQn0 is not the triggered pulse. See next macro.)
The width of the generated pulses (TOQnm) are defined by the duty values. The duty values are set by the following macro.
If the timer should be triggered by software, set input = TM_Q_Input_No_Input.
Often, the input pin TIQn0 is shared with the output pin TOQn0 in the devices. In this case, no output can be generated while triggering by an external signal.

(j) TM_Q_Trg_Pulse_Mode_Duty_Init (timer, duty_selection, duty_value, duty_output, duty1_value)

Init additionally duty cycle.

The width of the generated pulse (duty output - TOQnm) is defined by the duty value. The output TOQnm is set by a reset of the timer. This is either the case by reaching the cycle value, or recognizing a valid input edge. The output is reset when the timer reaches the duty value.

TM_Q_Cap_Cmp_0 (m=0) should not be used for “duty_selection”, because this register define the cycle value.

(k) TM_Q_Trg_Pulse_Mode_Write_Values (timer, cycle_value, duty1_value, duty2_value, duty3_value)

Update compare values in trigger pulse mode.
The values are reloaded automatically when the previous cycle value resets the timer.

(l) TM_Q_Trg_Pulse_Mode_Write_Duty_Value (timer, duty_selection, duty_value, duty1_value)

Write duty cycle value
The value is reloaded automatically when the cycle value resets the timer.

TM_Q_Cap_Cmp_0 should not be used for “duty_selection”, because this register define the cycle value.

(m) TM_Q_Trg_Pulse_Mode_Write_SW_Trigger (timer)

Write sw trigger pulse as external trigger input in trigger pulse mode.

The timer register is reset once and continue operating in the Trigger Pulse Mode. Pulses will be generated if outputs (TOPnm) are enabled.

(n) TM_Q_One_Shot_Pulse_Mode_Init (timer, clock, input, trigger_edge, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the one shot pulse mode.

The timer count up by the “clock” source to the “cycle_value”. Then the timer is reset, an interrupt request (INTTQnCC0) is set and if the output is enabled, the output (TOQn0) toggle. The timer wait till an external valid edge is recognized. A valid edge start the timer and toggle the timer output (TOQn0). The timer output toggle either upon a match of the cycle value or a valid input edge. (The cycle output TOQn0 is not the triggered pulse. See next macro.)
The start of the generated delayed pulses (TOQnm) are defined by the delay values. The delay values are set by the following macro.
If the timer should be triggered by software, set input = TM_P_Input_No_Input.
Often, the input pin TIQn0 is shared with the output pin TOQn0 in the devices. In this case, no output can be generated while triggering by an external signal.

(o) TM_Q_One_Shot_Pulse_Mode_Delay_Init (timer, delay_selection, delay_value, delay_output)

Init additionally delay time.

The start of the generated delayed pulse (delay output, TOPnm) is defined by the delay value. The output TOPnm is reset by a reset of the timer. This is the case by reaching the cycle value. The output is set when the timer reaches the delay value. After reaching the delay value, the timer continue counting till it reached the cycle value. Then it reset and wait for a next trigger edge.
While the timer count before he reached the cycle value, every valid input edge is ignored. The external trigger is only processed while the timer wait after it reaches the cycle value.

TM_Q_Cap_Cmp_0 (m=0) should not be used for “delay_selection”, because this register define the cycle value.

(p) TM_Q_One_Shot_Pulse_Mode_Write_Cycle_Value (timer, cycle_value)

Update compare values in one shot pulse mode.
The value is directly written to the compare registers of the timer.

(q) TM_Q_One_Shot_Pulse_Mode_Write_Delay_Value (timer, delay_selection, delay_value)

Write delay time
The value is directly written to the compare registers of the timer.

(r) TM_Q_One_Shot_Pulse_Mode_Write_SW_Trigger (timer)

Write sw trigger pulse as external trigger input in one shot pulse mode
The timer is started in the Trigger Pulse Mode. A delayed pulses will be generated if delay output TOQnm is enabled.

(s) TM_Q_PWM_Mode_Clk_Internal_Init (timer, clock, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the PWM mode - Internal clock

The PWMs are output on the TOQnm pins. The cycle output TOPn0 toggle upon a match of the timer and the cycle value.
This macro is used when an internal clock is used to supply the timer.

(t) TM_Q_PWM_Mode_Clk_External_Init (timer, input, input_edge, cycle_value, cycle_output)

Initialize the basic configuration of the timer in the PWM mode - External clock

The PWMs are output on the TOQnm pins. The cycle output TOQn0 toggle upon a match of the timer and the cycle value.
This macro is used when an external clock is used to supply the timer.
Often, the input pin TIQn0 is shared with the output pin TOQn0 in the devices. In this case, no output can be generated while an external clock supply is used.

(u) TM_Q_PWM_Mode_PWM_Init (timer, pwm_selection, pwm_value, pwm_output, pwm1_value)

Set PWM value and PWM output TOPnm.

TM_Q_Cap_Cmp_0 (m=0) should not be used for “pwm_selection”, because this register define the cycle value.

(v) TM_Q_PWM_Mode_Write_Values (timer, cycle_value, pwm1_value, pwm2_value, pwm3_value)

Update cycle value and all PWM values
The values are reloaded automatically when the previous cycle value resets the timer.

(w) TM_Q_PWM_Mode_Write_PWM_Value (timer, pwm_selection, pwm_value, pwm1_value)

The values are reloaded automatically when the previous cycle value resets the timer.
Update cycle and width values in PWM mode.

TM_Q_Cap_Cmp_0 should not be used for “pwm_selection”, because this register define the cycle value.

(x) TM_Q_Free_Running_Mode_Clk_Internal_Init (timer, clock)

Initialize the basic configuration of the timer in the free running mode - Internal clock

The timer run free with the supplied clock. It is reset at 0xFFFF and generate an overflow interrupt request INTTQnOV.

(y) TM_Q_Free_Running_Mode_Clk_External_Init (timer, input, input_edge)

Initialize the basic configuration of the timer in the free running mode - External clock

The timer run free with the supplied clock. It is reset at 0xFFFF and generate an overflow interrupt request INTTQnOV.

(z) TM_Q_Free_Running_Mode_Compare_Init (timer, cap_cmp_selection, cmp_value, output)

Init compare register.

One of the available capture/compare registers is configured as compare register. When the timer matches the compare value (’cmp_value’), it generates an interrupt request INTTQ*nm* for the chosen register *m*. If output is enabled, it toggles the output pin TOQ*nm*.

(aa)TM_Q_Free_Running_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)

Init capture register

One of the available capture/compare registers is configured as capture register. When a valid edge is recognized at the timer input TIQ*nm*, it generates an interrupt request INTTQ*nm* for the chosen register *m*. The current timer value is transferred to the capture register *m.* (The timer continue counting up.) The captured values can be read with the TM_Q_Capture_Read (timer) macro. If output is enabled, it toggles the output pin TOQ*nm*.
Often, the input pin TIQn0 is shared with the output pin TOQn0 in the devices. In this case, no output can be generated while an external trigger signal is used for capturing.

(ab)TM_Q_Free_Running_Mode_Write_Compare (timer, cap_cmp_selection, cmp_value)

Update compare values in free running mode.
The compare value is directly reloaded to the compare register of the timer.

(ac)TM_Q_Pulse_Width_Measure_Mode_Init (timer, clock)

Initialize the basic configuration of the timer in the pulse width measurement mode.

(ad)TM_Q_Pulse_Width_Measure_Mode_Capture_Init (timer, cap_cmp_selection, input, input_edge)

Init additionally capture mode.

One of the available capture/compare registers is configured as capture register. When a valid edge is recognized at the timer input TIQ*nm*, it generates an interrupt request INTTP*nm* for the chosen register *m*. The current timer value is transferred to the capture register *m* and the timer restarts at zero. The captured value can be read with the TM_Q_Capture_Read (timer) macro.
If a special input should be used as input source, “input” should be “TM_Q_Input_External_Input” and the special input should be selected with the TM_Q_Input_Special(...) macro.

Useful for pulse with measurement is configuring one register for capturing at rising edge, another for falling edge. The captured value in the register of the rising edge represents the time of the low phase of the signal. The captured value in the register of the falling edge represents the time of the high phase of the signal. The cycle of the signal can be calculated by adding both captured values.

(ae)TM_Q_Capture_Read (timer, cap_cmp_selection)

Return: 16-bit captured timer value

(af) TM_Q_Overflow_Read (timer)

Return: Overrun flag

(ag)TM_Q_Counter_Read (timer)

Return: Current 16-bit timer counter value

(ah)TM_Q_Enable (timer) / TM_Q_Disable (timer)

Enable / Disable timer
Start/Stop timer.

(ai) TM_Q_Input_Special (special_input, special_input_set)

Special inputs; Device specific

Some timers in a device allow a special input for capturing or clock supply. The Timer is configured to use an external signal. This external signal could be at a port pin or an internal redirected signal.
This mode is useful for baud rate measurements or similar purposes.
Please tale a look to the User’s Manual for informations about allowed combinations.

(aj) TM_Q_Sync_Mode (timer, sync_mode)

Sync mode; Device specific

Some timers in a device allow a synchronization of two timers. One timer is the Master timer and its clock supply and cycle time is copied to the second slave timer. The slave timer can use all of its capture/compare registers while the master timer configures the clock supply and cycle time.
This is for example useful for synchronous multi-channel PWM outputs.
Please take a look to the User’s Manual for informations about allowed combinations.

3.2.8 Timers - 16-Bit Interval Timer M

(1) Parameters

(a) clock

Choose clock source for timer.

Caution: Device and timer specific clock sources. Not all combinations are possible.

- TM_M_Clk_FXX, TM_M_Clk_FXX_2, TM_M_Clk_FXX_4, TM_M_Clk_FXX_64, TM_M_Clk_FXX_512, TM_M_Clk_WT, TM_M_Clk_FR_8, TM_M_Clk_FXT

(b) interval_value

Any 16-bit value as compare value for the counter.

(2) Function description

(a) TM_M_Interval_Mode_Init (clock, interval_value)

Initialization in interval mode.
The timer count up with the “clock” source till its value matches the interval value. Then the timer restarts at zero and generates an interrupt request (INTTM0EQ0).

(b) TM_M_Enable()

Enable/start timer

(c) TM_M_Disable()

Disable/stop timer

(d) TM_M_Set_Interval (interval_value)

Modify interval time.
To set M clocks as the interval period, set M-1 as interval value.
The compare value is directly written to the compare register of the timer.

3.2.9 Timers - Watch Timer Functions

(1) Parameters

(a) clock_source

Clock source for Watch timer and Watch interval timer (f_W)
Either a dedicated 32.768 kHz Crystal or the main clock divided by a prescaler can be used as clock supply. The 32.768 kHz crystal has the advantage, that a accurate 2Hz overflow frequency can be generated without a special requirement for the main clock.
(The setting of the prescaler is device specific.)
Following settings are possible:

- Watch timer_Clk_Source_Subclock
- Watch timer_Clk_Source_BRG

(b) clock_watch

Watch timer frequency
The clock source f_W will be divided by a fixed value to generate the watch timer overflow.
Following settings are possible:

- Watch timer_Clk_fW_16384 = f_W / 1638
- Watch timer_Clk_fW_8192 = f_W / 8192
- Watch timer_Clk_fW_32 = f_W / 32
- Watch timer_Clk_fW_16 = f_W / 16

(c) clock_interval

Watch timer interval frequency
The clock source f_W will be divided by a fixed value to generate the watch interval timer overflow.
Following settings are possible:

- Watch timer_Interval_Clk_fW_16 = f_W / 16
- Watch timer_Interval_Clk_fW_32 = f_W / 32
- Watch timer_Interval_Clk_fW_64 = f_W / 64
- Watch timer_Interval_Clk_fW_128 = f_W / 128
- Watch timer_Interval_Clk_fW_256 = f_W / 256
- Watch timer_Interval_Clk_fW_512 = f_W / 512
- Watch timer_Interval_Clk_fW_1024 = f_W / 1024
- Watch timer_Interval_Clk_fW_2048 = f_W / 2048

(d) prescaler setting

The setting of the prescaler is device specific.
The main clock source f_X will be divided by a fixed prescaler value and a divisor to generate the clock source for the watch timer f_W.
Following settings are possible:

- Watch timer_Clk_FXX = main clock
- Watch timer_Clk_FXX_2 = main clock / 2
- Watch timer_Clk_FXX_4 = main clock / 4
- Watch timer_Clk_FXX_8 = main clock / 8

(e) divisor

Any value between 1 and 256 (Value 0 = divider of 256).

(2) Function description

(a) Watch timer_Init (clock_source, clock_watch, clock_interval)

Clock initialization of watch timer and watch interval timer.
No start of timers.

(b) Watch timer_Enable() / Watch timer_Disable()

Enable/Disable Watch timer register;
Does not enable clock supply to watch timer.

(c) Watch timer_Enable_Clk_Supply() / Watch timer_Disable_Clk_Supply()

Enable/Disable Watch timer and Watch interval timer clock supply.
Watch timer need additional enable/disable.
Watch interval timer directly start/stop.

(d) Watch timer_Adjust()

Reset watch timer register; Same as Disable + Enable

(e) Watch timer_Set_Prescaler (prescaler, divisor)

Optional prescaler if subclock is not used as clock source.
Device specific
The main clock will be divided by a fixed prescaler value and a divisor between 1 and 256 (Value 0 = divider of 256).
The watch timer clock source f_W is calculated as follows:

$f_W = f_X / (\text{prescaler} * \text{divisor} * 2)$ f_X = main clock

3.2.10 Timers - Functions Of Watchdog Timer 2

(1) Parameters

(a) mode

Upon an overflow of the watchdog timer the system can perform following three operations: The system can stop, generate an NMI (INTWDT2) or generate a system reset.

- WatchDog_Mode_Stop
- WatchDog_Mode_NMI
- WatchDog_Mode_Reset

(b) clock

Watchdog 2 clock sources are devices specific
This is an example for V850ES/Fx2:

- WatchDog_Clk_Stop No clock source

Using ring oscillator		100 kHz (min.)	200 kHz (typ.)	400 kHz (max.)
• WatchDog_Clk_2_12_f_R	$2^{12}/f_R$	41.0 ms	20.5 ms	10.2 ms
• WatchDog_Clk_2_13_fr	$2^{13}/f_R$	81.9 ms	41.0 ms	20.5 ms
• WatchDog_Clk_2_14_fr	$2^{14}/f_R$	163.8 ms	81.9 ms	41.0 ms
• WatchDog_Clk_2_15_fr	$2^{15}/f_R$	327.7 ms	163.8 ms	81.9 ms
• WatchDog_Clk_2_16_fr	$2^{16}/f_R$	655.4 ms	327.7 ms	163.8 ms
• WatchDog_Clk_2_17_fr	$2^{17}/f_R$	1,310.7 ms	655.4 ms	327.7 ms
• WatchDog_Clk_2_18_fr	$2^{18}/f_R$	2,621.4 ms	1,310.7 ms	655.4 ms
• WatchDog_Clk_2_19_fr	$2^{19}/f_R$	5,242.9 ms	2,621.5 ms	1,310.7 ms
Using main clock		XX=20 MHz	f_{XX}=16 MHz	f_{XX}=10 MHz
• WatchDog_Clk_2_16_f_{XX}	$2^{18}/f_{XX}$	13.1 ms	16.4 ms	26.2 ms
• WatchDog_Clk_2_17_f_{XX}	$2^{19}/f_{XX}$	26.2 ms	32.8 ms	52.4 ms
• WatchDog_Clk_2_28_f_{XX}	$2^{20}/f_{XX}$	52.4 ms	65.5 ms	104.9 ms
• WatchDog_Clk_2_29_f_{XX}	$2^{21}/f_{XX}$	104.9 ms	31.1 ms	209.7 ms
• WatchDog_Clk_2_20_f_{XX}	$2^{22}/f_{XX}$	209.7 ms	262.1 ms	419.4 ms
• WatchDog_Clk_2_21_f_{XX}	$2^{23}/f_{XX}$	419.4 ms	524.3 ms	838.9 ms
• WatchDog_Clk_2_22_f_{XX}	$2^{24}/f_{XX}$	838.9 ms	1,048.6 ms	1,677.7 ms
• WatchDog_Clk_2_23_f_{XX}	$2^{25}/f_{XX}$	1,677.7 ms	2,097.2 ms	3,355.4 ms

(2) Function description

(a) WatchDog_Set (mode, clock)

Init watchdog clock supply without starting.
The “mode” define the action when the timer overflow.
The “clock” define the base clock which is used for counting.

(b) WatchDog_Start ()

Start Watchdog. It cannot be stopped. Only a reset will also reset and stop the watchdog timer.

(c) WatchDog_Reset()

Reset Watchdog timer register. This must be done periodically to avoid an overflow of the timer.

(d) WatchDog_Generate_Overflow()

Generate Watchdog overflow -> Stop / NMI / Reset
This macro generate a unnaturally overflow of the watchdog timer. The action which is performed was defined by the “mode” parameter during initialization.

3.2.11 Serial - Asynchronous Serial Interface A (UARTA)

(1) Parameters

(a) mode

This setting define if you only want to send or receive bytes or both.

- UART_A_Transmission
- UART_A_Reception
- UART_A_Trans_Recept

(b) direction

This setting define the shift direction for the transmit register. A standard UART interface uses the “UART_A_LSB_First “direction.

- UART_A_MSB_First
- UART_A_LSB_First

(c) parity

This setting define if a parity bit should be transmitted additionally to every byte and received bytes checked by the parity bit.

- UART_A_No_Parity
- UART_A_Zero_Parity
- UART_A_Odd_Parity
- UART_A_Even_Parity

(d) bit length

- UART_A_Length_7
- UART_A_Length_8

(e) stop bits

- UART_A_Stop_1
- UART_A_Stop_2

(f) transmit_state

This returned status flag indicate the current internal state.

- UART_A_Tx_Inactive
- UART_A_Tx_Active

(g) error_state

This returned status flag indicate if the received byte was accepted correctly, or if a overrun/framing/parity error was detected.

- UART_A_Rx_No_Error
- UART_A_Rx_Error

(h) System frequency

This setting is needed to set the correct divisors for the baud rate generator.
If a different system frequency is used, the macro must be modified.
Following standard system frequencies could be set:

- UART_A_SYS_10MHZ, UART_A_SYS_16MHZ, UART_A_SYS_20MHZ

(i) baudrates

The setting of the baudrates depend on the system frequency.
If a different baudrate is used, the macro must be modified.
Following standard baud rates could be set:

- UART_A_BR_EXT, UART_A_BR_300, UART_A_BR_600, UART_A_BR_1200, UART_A_BR_2400, UART_A_BR_4800, UART_A_BR_9600, UART_A_BR_19200, UART_A_BR_31250, UART_A_BR_38400, UART_A_BR_57600, UART_A_BR_115200

(2) Function description

(a) UART_A_Init (channel, mode, direction, oscillator, baudrate, bit length, parity, stop bits)

Initialize UART channel
The oscillator and baudrate parameters are used to set the correct values for the baud rate generator. If a different system frequency or non-standard baud rate is used, the macro must be modified.

(b) UART_A_Disable (channel)

Disable/stop UART channel. The operation of the whole UART macro channel stops. Current transmissions are stopped.

(c) UART_A_TxEnable (channel) / UART_A_TxDisable (channel)

Enable / Disable transmission on channel.
The reception of bytes is independent of this setting.

(d) UART_A_RxEnable (channel) / UART_A_RxDisable (channel)

Enable / Disable reception on channel
The transmission of bytes is independent of this setting.

(e) UART_A_Tx_Status_Read (channel)

Read transfer status flag (transfer in progress)

(f) UART_A_Rx_Overrun_Error_Read_Clear (channel)

Return: flag
Read and clear overrun error status

(g) UART_A_Rx_Parity_Error_Read_Clear (channel)

Return: flag
Read and clear parity error status

(h) UART_A_Rx_Framing_Error_Read_Clear (channel)

Return: flag
Read and clear framing error status

(i) UART_A_Tx_Write_Byte (channel, value)

Write byte to transmit data register

(j) UART_A_Rx_Read_Byte (channel)

Return: 8-bit read byte from receive data register

3.2.12 Serial - 3-Wire Variable-Length Serial I/O (CCSIB)

(1) Parameters

(a) clock_source

Device specific

- CSI_B_Clk_FXX_2, CSI_B_Clk_FXX_4, CSI_B_Clk_FXX_8, CSI_B_Clk_FXX_16, CSI_B_Clk_FXX_32, CSI_B_Clk_FXX_64, CSI_B_Clk_BRG, CSI_B_Clk_TMP0, CSI_B_Clk_FXX_128, CSI_B_Clk_External

(b) relax_level

This define the clock level if no transmission is made. The clock level could remain high or low.

- CSI_B_Clk_Level_High
- CSI_B_Clk_Level_Low

(c) active_edge

This define the clock edge which an external device or the V850 use for capturing the incoming data.

- CSI_B_Clk_Edge_RISING
- CSI_B_Clk_Edge_FALLING

(d) mode

This define if transmission, reception or both should be used. Additionally, the single or continuous transfer mode can be chosen.

- CSI_B_Mode_Single_Transmission
- CSI_B_Mode_Single_Reception
- CSI_B_Mode_Single_Trans_Recept
- CSI_B_Mode_Cont_Transmission
- CSI_B_Mode_Cont_Reception
- CSI_B_Mode_Cont_Trans_Recept

(e) direction

The direction of the bits can be configured.

- CSI_B_MSB_FIRST
- CSI_B_LSB_FIRST

(f) bit length

The bit length for transmission and reception can be configured between eight and sixteen bits.

- CSI_B_Bit_Length_8, CSI_B_Bit_Length_9, CSI_B_Bit_Length_10, CSI_B_Bit_Length_11, CSI_B_Bit_Length_12, CSI_B_Bit_Length_13, CSI_B_Bit_Length_14, CSI_B_Bit_Length_15, CSI_B_Bit_Length_16

(g) transfer_status

Status flag for CSI operation

- CSI_B_Transfer_Inactive
- CSI_B_Transfer_Active

(h) overrun_error

Status flag for CSI operation

- CSI_B_Rx_Overrun_Inactive
- CSI_B_Rx_Overrun_Active

(i) Prescaler for baud rate generator (to be extended)
CSI_B_BRG_FXX, CSI_B_BRG_FXX_2, CSI_B_BRG_FXX_4, CSI_B_BRG_FXX_8

(2) Function description

(a) CSI_B_Init (channel, clock_source, relax_level, active_edge, mode, direction, bit length)

Initialize Clocked Synchronous Interface
A shift register of “bit length” width is shifted out by a clock signal. The clock signal could be generated internally or externally. The transmitted data is shifted out on one device pin while the received is shifted in the same register from another pin. Transmission and reception is done in parallel.

(b) CSI_B_Set_BRG (channel, prescaler, divisor)

Set baud rate generator
Calculating the baud rate

f_{BRG} = prescaler / divisor / 2

Set value of divisor from 1 to 255
divisor = 256 if the value the divisor is 0.

(c) CSI_B_Disable (channel)

Reset CSI to default values. The whole macro stops operating. A current transmission/reception will be terminated.

(d) CSI_B_Stop (channel)

Stop SIO (Re-initialization have to be made using CSI_B_Init

(e) CSI_B_Transfer_Clock_Output_Enable (channel) /
CSI_B_Transfer_Clock_Output_Disable (channel)

Enable / Disable transfer clock output.
This macro controls starting of the transfer operation in the master mode.
If only reception is enabled in the single transfer mode, the reception operation is started when the receive register is read. To read the last receive data, disable the output clock, read the last receive data, and then disable starting the next reception operation.
Similarly, if only reception is enabled in the continuous transfer mode, starting the reception operation can be disabled after completion of reception of the last receive data, by disabling the output clock one clock before reception of the last receive data is completed. After the last data is read, reception is enabled by CSI_B_Transfer_Clock_Output_Enable(...) again and reading the reception register. In the slave reception mode, this macro also enables the internal operating clock, and therefore, execute CSI_B_Transfer_Clock_Output_Enable(...).

(f) CSI_B_Write_16bit (channel, value_16bit)

Write 16 bit value to transmission register

(g) CSI_B_Write_8bi t (channel, value_8bit)

Write 8 bit value to transmission register

(h) CSI_B_Read_16bit (channel)

Return: Read 16 bit value from reception register

(i) CSI_B_Read_8bit (channel)

Return: Read 8 bit value from reception register

(j) CSI_B_Operation (channel)

Return: flag transfer_status
Read operating status

(k) CSI_B_Overrun (channel)

Return: flag overrun_error
Read overrun error status

3.2.13 Serial - I²C Bus

(1) Parameters

(a) clock

The clock is generated by a prescaler and a choose-able divider. Some total divisions could be achieved by different register settings. This is the reason that some clock constants appear more than one time. Please read the User's Manual for further details.
Device specific;

- IIC_Clk_LowSpeed_FXX_44, IIC_Clk_LowSpeed_FXX_88, IIC_Clk_LowSpeed_FXX_132, IIC_Clk_LowSpeed_FXX_176, IIC_Clk_LowSpeed_FXX_220, IIC_Clk_LowSpeed_FXX_86, IIC_Clk_LowSpeed_FXX_172, IIC_Clk_LowSpeed_FXX_258, IIC_Clk_LowSpeed_FXX_344, IIC_Clk_LowSpeed_FXX_86_2, IIC_Clk_LowSpeed_FXX_66, IIC_Clk_LowSpeed_FXX_132_2, IIC_Clk_LowSpeed_FXX_198, IIC_Clk_HighSpeed_FXX_24, IIC_Clk_HighSpeed_FXX_48, IIC_Clk_HighSpeed_FXX_72, IIC_Clk_HighSpeed_FXX_96, IIC_Clk_HighSpeed_FXX_24_2, IIC_Clk_HighSpeed_FXX_18, IIC_Clk_HighSpeed_FXX_36, IIC_Clk_HighSpeed_FXX_54, IIC_Clk_HighSpeed_FXX_24_3, IIC_Clk_HighSpeed_FXX_24_4, IIC_Clk_HighSpeed_FXX_36_2, IIC_Clk_HighSpeed_FXX_48_2, IIC_Clk_HighSpeed_FXX_60, IIC_Clk_HighSpeed_FXX_12

(b) filter

The digital filter can only be used in high speed mode.

- IIC_Filter_On
- IIC_Filter_Off

(c) Master device status

Returned status flag of master device status. If this I²C macro generated successfully a start condition, this bit will be set. It is reset when this device is a slave or in stand-by mode.

- IIC_Master_Status
- IIC_Slave_Status

(d) Arbitration loss detection

Returned status flag of arbitration loss

- IIC_Arbitration_Loss
- IIC_Arbitration_Ok

(e) Detection of extension code reception

Returned status flag indicate if an extension code was received.

- IIC_Extension_Code_Receiced
- IIC_No_Extension_Code

(f) Matching addresses detection

Returned status flag indicate, if the set slave address for the V850 was detected.

- IIC_Address_Match
- IIC_No_Address_Match

(g) Transmit/receive status detection

Returned status flag indicate the current transfer direction. After a generated start condition it is set. It changes the status after transmitting the slave address. If the slave address was even, it is in the transmission status. If the slave address was odd it is in the reception status.

- IIC_Transmission_Status
- IIC_Reception_Status

(h) Acknowledge detection

Returned status flag. It is set when the slave device acknowledged the received data.

- IIC_Ack_Detected
- IIC_Ack_Not_Detected

(i) Start condition detection

Returned status flag. If a start condition of regardless origin was detected before.

- IIC_Start_Detected
- IIC_Start_Not_Detected

(j) Stop condition detection

Returned status flag. If a stop condition of regardless origin was detected before.

- IIC_Stop_Detected
- IIC_Stop_Not_Detected

(k) Start issued/rejected

Returned status flag. If a start condition was tried to generate and cancelled.

- IIC_Start_Issued
- IIC_Start_Rejected

(l) IIC bus status

This returned flag indicate the current bus status.

- IIC_Bus_Released
- IIC_Bus_Communication

(2) Function description

(a) IIC_Init (channel, clock, filter)

Basic initialization of I²C peripheral.
The I²C macro is enabled and the clock source will be chosen.
A digital filter can be used additionally only in the high speed clock mode.
The corresponding port pins are set for the alternate function.

(b) IIC_Enable (channel) / IIC_Disable (channel)

Enable I²C module or disable I²C module and reset status registers.

(c) IIC_Generate_Start (channel)

Generate a start condition.
Either a stop condition must be detected/generated before, or the IIC_Start_If_No_Stop (...) macro must be used before.
A start condition could be rejected. Read status by IIC_Get_Status_Start_Issued(...) macro.

(d) IIC_Generate_Stop (channel)

Generate stop condition.

(e) IIC_Acknowledge_Enable (channel) / IIC_Acknowledge_Disable (channel)

Enable/Disable acknowledgment of received data.
This must be set/reset before the ninth clock will be generated.

(f) IIC_Wait_and_Int_8th_clock (channel) / IIC_Wait_and_Int_9th_clock (channel)

Control of wait and Interrupt request generation.
The I²C macro could wait after transmission of the data byte before or after the acknowledge bit.
If the transmission wait after the eighth bit, use the IIC_Cancel_Wait(...) macro to cancel the wait.
When the I²C macro enter the wait status, an interrupt could be generated.

(g) IIC_Stop_Interrupt_Enable (channel) / IIC_Stop_Interrupt_Disable (channel)

Enable/disable generation of interrupt request when stop condition is detected

(h) IIC_Cancel_Wait (channel)

Wait cancellation control.
Use it either to allow the next reception, or transmit the acknowledge bit after the eighth clock cycle.

(i) IIC_Exit_Communication (channel)

Exit from current communications.

(j) IIC_Get_Status_Master (channel)

Return: flag IIC_Master_Status / IIC_Slave_Status
Returned status flag of master device status. If this I²C macro generated successfully a start condition, this bit will be set. It is reset when this device is a slave or in stand-by mode.

(k) IIC_Get_Status_Arbitration (channel)

Return flag IIC_Arbitration_Loss / IIC_Arbitration_Ok
Returned status flag of arbitration loss

(l) IIC_Get_Status_Extension (channel)

Return flag IIC_Extension_Code_Receiced / IIC_No_Extension_Code
Returned status flag indicate if an extension code was received.

(m) IIC_Get_Status_Addr_Match (channel)

Return flag IIC_Address_Match / IIC_No_Address_Match
Flag indicate, if the set slave address for the V850 was detected.

(n) IIC_Get_Status_Direction (channel)

Return flag IIC_Transmission_Status / IIC_Reception_Status
Flag indicate the current transfer direction. After a generated start condition it is set. It changes the status after transmitting the slave address. If the slave address was even, it is in the transmission status. If the slave address was odd it is in the reception status.

(o) IIC_Get_Status_Acknowledge (channel)

Return flag IIC_Ack_Detected / IIC_Ack_Not_Detected
It is set when the slave device acknowledged the received data.

(p) IIC_Get_Status_Start (channel)

Return flag IIC_Start_Detected / IIC_Start_Not_Detected
If a start condition of regardless origin was detected before.

(q) IIC_Get_Status_Stop (channel)

Return flag IIC_Stop_Detected / IIC_Stop_Not_Detected
If a stop condition of regardless origin was detected before.

(r) IIC_Get_Status_Start_Issued (channel)

Return flag IIC_Start_Issued / IIC_Start_Rejected
If a start condition was tried to generate and cancelled.

(s) IIC_Get_Status_Bus (channel)

Return flag IIC_Bus_Released / IIC_Bus_Communication
This returned flag indicate the current bus status.

(t) IIC_Start_If_No_Stop (channel) / IIC_No_Start_If_No_Stop (channel)

A stop condition must be detected/generated before a start condition can be generated.
By using the IIC_Start_If_No_Stop(...) macro, a start condition can directly be generated.
A start condition could be rejected if no stop condition was detected before. (Read status by IIC_Get_Status_Start_Issued(...) macro.)

(u) IIC_Com_Reservation_Enable (channel) / IIC_Com_Reservation_Disable (channel)

Enable / Disable Communication Reservation Function.
Enable communication reservation to wait till an stop condition is detected on the bus. A previous set start condition takes directly place to reserve the bus.

(v) IIC_Digital_Filter_Enable (channel) / IIC_Digital_Filter_Disable (channel)

Enable / Disable Digital Filter.
Only in high speed mode.

(w) IIC_Get_Status_SDA_Level (channel) / IIC_Get_Status_SCL_Level (channel)

Get current level of SDA / SCL line

(x) IIC_Write_Data (channel, value)

Write data

(y) IIC_Read_Data (channel)

Read received data

(z) IIC_Set_Slave_Address (channel, address)

Set slave address. If this slave address is detected in non-master mode, an interrupt will be generated and the corresponding flag in the status register is set.

3.2.14 Clock & Power - Clock Generation Function

(1) Parameters

(a) clock

CPU clock dividers are device specific.

- CPU_Clk_FXX, CPU_Clk_FXX_2, CPU_Clk_FXX_4, CPU_Clk_FXX_8, CPU_Clk_FXX_16, CPU_Clk_FXX_32, CPU_Clk_FXT

(b) main_feedback, sub_feedback

A feedback resistor could be connected to allow an oscillation for the main or subclock. They are switched on by default (=1)

- 0 = off / disconnected / no oscillator used
- 1 = on / connected / oscillator used

(c) main_operation

The main oscillator can be switched on or off. This setting is only possible while operating on a different clock source.

- 0 = off
- 1 = on = default

(d) Clock status

Returned enum of current used CPU clock

- CLK_Main_Clock_Used
- CLK_Subclock_Used
- CLK_Ring_Oscl_Used

(e) PLL_Lockup_Time

Selection of PLL lockup time.
Device specific.

- PLL_Lockup_Time_2_12_fX = $2^{12} / f_X$
- PLL_Lockup_Time_2_13_fX = $2^{13} / f_X$

(f) PLL_Multiplier

CPU operation clock selection

- CLK_PLL_ClockThrough
- CLK_PLL_Used

(g) Status of PLL

Returned status of PLL.

- CLK_PLL_Locked
- CLK_PLL_Unlocked

(2) Function description

(a) Clock_Set_CPU_Clock (clock, main_feedback, sub_feedback, main_operation)

Set clock for CPU and configure oscillators.
The clock define the prescaler for the CPU.
THe feedback resistors of the oscillators can be disconnected to disable oscillation of an unused oscillator.
The main oscillator can be switched on or off. This setting is only possible while operating on a different clock source.

(b) Clock_Get_CPU_Clock_Status ()

Return enum CLK_Main_Clock_Used / CLK_Subclock_Used / CLK_Ring_Oscl_Used
Read Status of CPU clock (f_{CPU})

(c) Clock_Set_PLL (PLL_operation, PLL_multiplier, PLL_Lockup_Time)

Configure PLL. The PLL can be switched on and off by the PLL_operation parameter.
The PLL_multiplier parameter choose either the multiplied or direct oscillator clock.
Clock base for CPU and peripherals.

(d) Clock_Get_PLL_Status()

Return flag CLK_PLL_Locked / CLK_PLL_Unlocked
Get Status of PLL

(e) Clock_Output_Enable (clock) / Clock_Output_Disable()

Enable / Disable clock output on the device pin.

3.2.15 Clock & Power - Standby Function

(1) Parameters

(a) standby_mode

Following modes are possible:

- IDLE1
- SW_STOP1
- IDLE2
- SW_STOP2 (= SW_STOP1)

(b) release_condition

Sources which can cause a stand-by release

- Standby_Release_WDT2_NMI_Int
- Standby_Release_WDT2_NMI
- Standby_Release_WDT2_Int
- Standby_Release_WDT2
- Standby_Release_NMI_Int
- Standby_Release_NMI
- Standby_Release_Int
- Standby_Release_Reset_Only

(c) ost

The oscillation stabilization time is device specific.

		4 MHz	5 MHz
• OST_2_10_Fxx	$2^{10}/f_{XX}$	(0.256 ms	0.205 ms)
• OST_2_11_Fxx	$2^{11}/f_{XX}$	(0.512 ms	0.410 ms)
• OST_2_12_Fxx	$2^{12}/f_{XX}$	(1.024 ms	0.819 ms)
• OST_2_13_Fxx	$2^{13}/f_{XX}$	(2.048 ms	1.638 ms)
• OST_2_14_Fxx	$2^{14}/f_{XX}$	(4.096 ms	3.277 ms)
• OST_2_15_Fxx	$2^{15}/f_{XX}$	(8.192 ms	6.554 ms)
• OST_2_16_Fxx	$2^{16}/f_{XX}$	(16.38 ms	13.107 ms)

(2) Function description

(a) Standby_Enter (standby_mode, release_condition)

Entering a stand-by mode (IDLE1, IDLE2, SW_STOP). The condition which is needed to release the stand-by mode can be specified.

(b) Oscl_Stabil_Time_Set (ost)

Set Stabilization Oscillation Time.

3.2.16 Clock & Power - Clock Monitor

(1) Function description

(a) Clock_Monitor_Enable()

Enable Clock Monitor. Disabling is not possible except by reset.

3.2.17 Clock & Power - Reset Function

(1) Parameters

(a) Reset source

- RESET_SRC_LVI = 0x01 Generation of reset signal from low voltage detector
- RESET_SRC_CLKMON = 0x02 Generation of reset signal from clock monitor
- RESET_SRC_WDT2 = 0x10 Generation of reset signal from watchdog timer 2

(2) Function description

(a) Reset_Get_Source()

Return flag RESET_SRC_LVI / RESET_SRC_CLKMON / RESET_SRC_WDT2
Get information about reset source.

(b) Reset_Clear_Source()

Clear information about reset source

3.2.18 Clock & Power - Low-Voltage Detector

(1) Parameters

(a) operation

- LVI_ON
- LVI_OFF

(b) generation

- LVI_GENERATE_INTERRUPT
- LVI_GENERATE_RESET

(c) level

LVI threshold voltage

- LVI_LEVEL_4_4_V = 4.4V threshold level
- LVI_LEVEL_4_2_V = 4.4V threshold level

(d) LVI status

- LVI_DETECTED
- LVI_NOT_DETECTED^

(2) Function description

(a) LVI_Init (operation, generation, level)

Initialize Low Voltage Detector

(b) LVI_Disable()

Set default values and disable LVI.

(c) LVI_Get_Status()

Return LVI_NOT_DETECTED / LVI_DETECTED
Status of voltage detection

3.2.19 Clock & Power - RAM Retention Voltage Detection Operation

(1) Parameters

(a) RAM validation flag

- LVI_RAM_VALID
- LVI_RAM_INVALID

(2) Function description

(a) LVI_RAM_Get_Status()

Return LVI_RAM_VALID / LVI_RAM_INVALID
The flag indicates whether the internal RAM is valid or not. An invalid RAM content caused by too low supply voltage could be indicated by this flag.

(b) LVI_RAM_Clear_Status()

Clear the flag

(c) LVI_RAM_Set_Invalid()

Only for emulation: RAM status flag set

(d) LVI_RAM_Set_Valid()

Only for emulation: RAM status flag clear

3.2.20 Analog - A/D Converter

(1) Parameters

(a) time

Conversion time setting device specific

	A/D Conv. Time	f_{XX}=20MHz	f_{XX}=16MHz	f_{XX}=4MHz	A/D Stabilization Time
• AD_Clk_31_fXX	$31/f_{XX}$	---	---	7.75 µs	$6/f_{XX}$
• AD_Clk_62_fXX	$62/f_{XX}$	3.10 µs	3.88 µs	15.50 µs	$31/f_{XX}$
• AD_Clk_93_fXX	$93/f_{XX}$	4.65 µs	5.81 µs	---	$47/f_{XX}$
• AD_Clk_124_fXX	$124/f_{XX}$	6.20 µs	7.75 µs	---	$50/f_{XX}$
• AD_Clk_155_fXX	$155/f_{XX}$	7.75 µs	9.69 µs	---	$50/f_{XX}$
• AD_Clk_186_fXX	$186/f_{XX}$	9.30 µs	11.63 µs	---	$50/f_{XX}$
• AD_Clk_217_fXX	$217/f_{XX}$	10.85 µs	13.56 µs	---	$50/f_{XX}$
• AD_Clk_248_fXX	$248/f_{XX}$	12.40 µs	15.50 µs	---	$50/f_{XX}$
• AD_Clk_279_fXX	$279/f_{XX}$	13.95 µs	---	---	$50/f_{XX}$
• AD_Clk_310_fXX	$310/f_{XX}$	15.50 µs	---	---	$50/f_{XX}$
• AD_Clk_341_fXX	$341/f_{XX}$	---	---	---	$50/f_{XX}$
• AD_Clk_372_fXX	$372/f_{XX}$	---	---	---	$50/f_{XX}$
• AD_Clk_403_fXX	$403/f_{XX}$	---	---	---	$50/f_{XX}$
• AD_Clk_434_fXX	$434/f_{XX}$	---	---	---	$50/f_{XX}$
• AD_Clk_465_fXX	$465/f_{XX}$	---	---	---	$50/f_{XX}$
• AD_Clk_496_fXX	$496/f_{XX}$	---	---	---	$50/f_{XX}$

(b) trigger_type

Trigger source for A/D converter

- AD_Software_Trigger
- AD_External_Trigger
- AD_Timer_Trigger_0
- AD_Timer_Trigger_1

(c) mode

Conversion mode

- AD_Continuous_Select_Mode
- AD_Continuous_Scan_Mode
- AD_One_Shot_Select_Mode
- AD_One_Shot_Scan_Mode

(d) trigger_edge

Edge(s) to trigger next conversion.

- AD_Trigger_No_Edge
- AD_Trigger_Falling_Edge
- AD_Trigger_Rising_Edge
- AD_Trigger_Both_Edges

(e) status

Returned AD unit status

- AD_Converter_Idle
- AD_Converter_Operating

(f) selection

- AD_PowerFail_If_HigherOrEqual Interrupt (INTAD) when ADA0CRnH >= ADA0PFT
- AD_PowerFail_If_Lower Interrupt (INTAD) when ADA0CRnH < ADA0PFT

(g) channel

AD_CH0=0, AD_CH1, AD_CH2, AD_CH3, AD_CH4, AD_CH5, AD_CH6, AD_CH7, AD_CH8, AD_CH9, AD_CH10, AD_CH11, AD_CH12, AD_CH13, AD_CH14, AD_CH15

(2) Function description

(a) AD_Init (mode, trigger_type, trigger_edge, time)

Initialization of A/D converter;
If trigger type is set for timer trigger, the timer must be configured separately.
Set trigger edge to no edge if external trigger is unused.

(b) AD_Enable() / AD_Disable()

Enable / Disable A/D converter operation

(c) AD_Select_Channel (channel)

Select channel for conversion result(s)

(d) AD_Get_Status()

Return AD_Converter_Idle / AD_Converter_Operating
Get status (Idle, Operating)

(e) AD_Get_Result_10bit (channel)

Get 10-bit result

(f) AD_Get_Result_8bit (channel)

Get 8-bit result

(g) AD_PowerFail_Init (selection, value)

Initialize power-fail compare mode.
Enabling must be done separately.

(h) AD_PowerFail_Enable() / AD_PowerFail_Disable()

Enable / Disable power-fail comparison.

(i) AD_PowerFail_Set_Value (value)
Set value for power-fail comparison.

3.2.21 Analog - D/A Converter

(1) Parameters

(a) channel

Device specific.

- DA_CH0=0, DA_CH1

(b) mode

In the normal mode, the written value is directly converted to an analog value.
In the real time mode, the analog conversion is triggered by a timer.

- DA_NORMAL_MODE
- DA_REAL_TIME_MODE

(c) value
Any eight bit value

(2) Function description

(a) DA_Init (channel, mode, value)

Initialize DA converter and port pins. The mode specify if the value should be immediately converted, or if it should be triggered by a timer signal.

(b) DA_Enable (channel) / DA_Disable (channel)

Enable/Disable DA converter channel.
Do not (re)set port pin settings.

(c) DA_Write_Value (channel, value)

Write value to DA converter.

3.2.22 Miscellaneous - DMA Controller

(1) Parameters

(a) channel

Choose one of the available DMA channels.
The maximum channel number is device specific.

(b) src_address, dest_address

Source and destination address. The address could either be a SFR, in internal RAM or external memory.

(c) src_location, dest_location

Specify if the address is in internal RAM or is it a SFR or in external memory.

- DMA_EXT_OR_IO
- DMA_IRAM

(d) src_direction, dest_direction

Specify if the source or destination address should be incremented, decremented or fixed after each DMA transfer.

- DMA_INCREMENT
- DMA_DECREMENT
- DMA_FIXED

(e) count

The total count of transfers.
The value will be decreased by one by the macro to write the correct register value.

(f)

(g) data_size

Transfer data could either 8-bit or 16-bit.

- DMA_TRANSFER_16
- DMA_TRANSFER_8

(h) trigger_src

Device specific trigger source for DMA operation.
This is an example for sources of V850ES/FJ2:

- DMA_INT_DISABLE, DMA_INTLVI, DMA_INTP0, DMA_INTP1, DMA_INTP2, DMA_INTP3, DMA_INTP4, DMA_INTP5, DMA_INTP6, DMA_INTP7, DMA_INTTQ0OV, DMA_INTTQ0CC0, DMA_INTTQ0CC1, DMA_INTTQ0CC2, DMA_INTTQ0CC3, DMA_INTTP0OV, DMA_INTTP0CC0, DMA_INTTP0CC1, DMA_INTTP1OV, DMA_INTTP1CC0, DMA_INTTP1CC1, DMA_INTTP2OV, DMA_INTTP2CC0, DMA_INTTP2CC1, DMA_INTTP3OV, DMA_INTTP3CC0, DMA_INTTP3CC1, DMA_INTTM0EQ, DMA_INTCB0R, DMA_INTCB0T, DMA_INTCB1R, DMA_INTCB1T, DMA_INTUA0R, DMA_INTUA0T, DMA_INTUA1R, DMA_INTUA1T, DMA_INTAD, DMA_INTC0ERR, DMA_INTC0WUP, DMA_INTC0REC, DMA_INTC0TRX, DMA_INTKR, DMA_INTC2REC, DMA_INTX2TRX, DMA_INTC3REC, DMA_INTC3TRX

(i) DMA_Read_Status

Returned status of DMA operation

- DMA_FINISHED
- DMA_INPROGRESS

(2) Function description

(a) DMA_Init (channel, src_address, src_location, src_direction, dest_address, dest_location, dest_direction, count, data_size, trigger_src)

Initialize DMA controller channel

(b) DMA_Enable (channel) / DMA_Disable (channel)

Enable/Disable DMA transfer

(c) DMA_Trigger_By_Software (channel)

Enable and start DMA transfer by software

(d) DMA_Read_Status (channel)

Return DMA_FINISHED / DMA_INPROGRESS
Read DMA status (finished / in progress)

(e) DMA_Terminal_Count (channel)

Return DMA_FINISHED / DMA_INPROGRESS
Read DMA status (finished / in progress) Same as above

(f) DMA_Clear_Request channel)

Clear DMA transfer request

3.2.23 Miscellaneous - On Chip Debug Interface

(1) Function description

(a) OCD_Enable() / OCD_Disable()

Enable / Disable On-chip debugging.
If the port pins for the on-chip debug interface should be used as port pins, or debugging should be avoided, the OCD interface should be disabled.
To enable on-chip debugging, the OCD_Enable(...) macro should be used.

3.2.24 Miscellaneous - CRC Function

(1) Parameters

(a) data

Any 16-bit value

(2) Function description

(a) CRC_Init()

Initialize CRC unit. The result register will be reset to zero.

(b) CRC_Write_Next (data)

Write next 16-bit data for CRC calculation. The whole block to be marked with the CRC must be continuously copied 16-bit by 16-bit to the CRC peripheral. The calculated CRC sum can be read by the following macro.

(c) CRC_Read_Result()

Read result of CRC calculation

3.2.25 Miscellaneous - ROM Correction Function

(1) Parameters

(a) channel

The count of ROM correction points is device specific.
E.g. V850ES/SJ2:

- ROMCorr_CH0=0, ROMCorr_CH1, ROMCorr_CH2, ROMCorr_CH3, ROMCorr_CH4, ROMCorr_CH5, ROMCorr_CH6, ROMCorr_CH7

(b) address

If this address matches the program counter, a debug trap instruction (DBTRAP) will be executed. Any internal ROM address.

(2) Function description

(a) ROM_Correction_Init (channel, address)

A debug trap instruction will be executed, if the given address matches the program counter. The function which is called should have the possibility to react on this exception. It could launch corrected code from any location or enter a user debug mode.

[MEMO]

Chapter 4 Demos

4.1 Description

For all macros is a demo code available. The code is in the same way organized in different files like the header files of the Peripheral Macro Driver. The peripherals are grouped in the demo files. E.g. the generic macro header file for the serial interface UART is called “Macro_UART_A.h” and the corresponding test and demo code is included in the “Demo_Serial.c” file. The demo code for the CSI and IIC interface is also present in this file.

The demo program source can be used as demo code for all APIs. Each demo part inside a file represent one full functional mode of the peripheral.That means, that for most possible modes of a peripheral a complete demo including initialization, usage and termination is included.

The demo project uses the UART channel 0 for communication with the user. Using this interface and a connected PC with a terminal program, each individual demo can be launched and reports can be received back. A hardware transceiver driver for the RS232 interface should be connected to the UART0 interface. This driver should convert the CMOS level to RS232 standard and vice versa.

UART setting: UART0, 38400 bps, 8 Bit, No parity, 1 Stop bit

After programming the demo code into the device, it starts sending the main menu via the UART interface after reset. The menu is printed in the window of the terminal program on the connected PC. Now, the user can choose one peripheral group for testing by key hit. A chosen peripheral group offer a dedicated sub-menu where the user can choose the peripheral. Each peripheral offer at least one additional sub-menu where a certain demo code can be launched.

Table 4-1: Menu Structure

Main menu	1st sub-menu	2nd sub-menu	3rd sub-menu (optional)
Peripheral group	Peripheral	Test	Test options

Depending on the peripheral, the demos need additional external hardware or interaction of the user. Which additional things are needed for the different demos is described within the demo projects. An ICE is suitable for many tests. Instead of a real device, an ICE offer the possibility to step through the code and look into or modify SFRs of the peripheral.t

Table 4-2: External Test Hardware

Group	Peripheral test	Optional or required tool
Ports	Ports	Logic tester (Voltmeter) / Oscilloscope
	Bus	External 8/16 bit memory
	RTO	Oscilloscope / Logic Analyser
Interrupts	Interrupt	-
	Key Int	-
Timers	Timer_P	Oscilloscope, Wave generator
	Timer_Q	Oscilloscope, Wave generator
	Timer_M	-
	Watch timer	-
	Watchdog	-
Serial	UART_A	RS232 transceiver
	CSI_B	Oscilloscope, Wave generator
	IIC	External EEPROM 24x02 or similar (Adress pins low level)
Clock & Power	Clock & Power	-
Analog	Macro_AD	-
	Macro_DA	Oscilloscope
Miscellaneous	DMA	-
	OCD	-
	CRC	-
	ROMcorr	-

All dedicated demo programs use interrupts. The interrupt service routine (ISR) for each interrupt set only a flag or increase a variable and return from interrupt. The demo function in the main program poll for this flag or analyse the variable count. This is a very simple way to use interrupts for the peripheral. It is not intended to demonstrate the optimal use of interrupts, but it should demonstrate the way interrupts are generated.

The source files for the demo programs include comments for better understanding and simple diagrams. E.g. the demo program for the PWM timer output indicate the expected signal at the corresponding device pin.

NEC

Facsimile Message

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

From:

Name

Company

Tel. FAX

Address

Thank you for your kind support.

North America
NEC Electronics America Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Market Communication Dept.
Fax: +49(0)-211-6503-1344

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-6250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	☐	☐	☐	☐
Technical Accuracy	☐	☐	☐	☐
Organization	☐	☐	☐	☐

[MEMO]