NEC

Preliminary User's Manual

Applilet3

Device Driver Configurator

API Reference

Target Device

V850 Microcontroller

Document No. ZUD-CD-09-0243-01 (1st edition)
Date Published November 2009 CP(K)

Tamotsu Iwasaki, Team Manager
Development Tool Solution Group
Multipurpose Microcomputer Systems Division
Microcomputer Operations Unit
NEC Electronics Corporation

[MEMO]

SUMMARY OF CONTENTS

CHAPTER 1 GENERAL ... 12

CHAPTER 2 OUTPUT FILES ... 13

CHAPTER 3 API FUNCTIONS ... 19

APPENDIX A INDEX ... 223

- The information in this document is current as of November, 2009. 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

[MEMO]

[MEMO]

[MEMO]

TABLE OF CONTENTS

CHAPTER 1 GENERAL ... 12

1.1 Overview ... 12
1.2 Features ... 12

CHAPTER 2 OUTPUT FILES ... 13

2.1 Overview ... 13
2.2 Output File ... 14

CHAPTER 3 API FUNCTIONS ... 19

3.1 Overview ... 19
3.2 Output Function ... 19
3.3 Function Reference ... 26
3.3.1 System ... 28
3.3.2 External Bus ... 42
3.3.3 Port ... 45
3.3.4 INT ... 51
3.3.5 Serial ... 62
3.3.6 A/D ... 122
3.3.7 D/A ... 131
3.3.8 Timer ... 137
3.3.9 Watch Timer ... 164
3.3.10 RTC ... 169
3.3.11 Real-Time Output ... 200
3.3.12 DMA ... 209
3.3.13 LVI ... 217

APPENDIX A INDEX ... 223

LIST OF FIGURES

Figure No. Title, Page

3-1 Notation Format of API Functions ... 26

LIST OF TABLES

Table No. Title, Page

2-1 File List ... 14
3-1 API Function List ... 19
3-2 API Functions: [System] ... 28
3-3 API Functions: [External Bus] ... 42
3-4 API Functions: [Port] ... 45
3-5 API Functions: [INT] ... 51
3-6 API Functions: [Serial] ... 62
3-7 API Functions: [A/D] ... 122
3-8 API Functions: [D/A] ... 131
3-9 API Functions: [Timer] ... 137
3-10 API Functions: [Watch Timer] ... 164
3-11 API Functions: [RTC] ... 169
3-12 API Functions: [Real-Time Output] ... 200
3-13 API Functions: [DMA] ... 209
3-14 API Functions: [LVI] ... 217

CHAPTER 1 GENERAL

This chapter gives an overview of the Applilet3.

1.1 Overview

The design tool enables you to output the source code (device driver programs, C source files and header files) necessary to control the peripheral hardware functions provided by the microcontroller (clock generator, port functions, etc.) by configuring various information using the GUI.

1.2 Features

The Applilet3 has the following features.

- Reporting function
  You can output configured information using the Applilet3 as files in various formats for use as design documents.

- Renaming function
  The user can change default names assigned to the files output by the Applilet3 and the API functions contained in the source code.

- Code generating function
  The Applilet3 can output not only device driver programs in accordance with the information configured using the GUI, but also a build environment such as sample programs containing main functions and link directive files.

- Project/workspace file generating function
  The Applilet3 can output project and workspace files that can be used in application system integrated development environments (PM+, MULTI and IAR Embedded Workbench).

CHAPTER 2 OUTPUT FILES

This appendix describes the files output by Applilet3.

2.1 Overview

Below are the files output by the Applilet3.

Unit of Output	File Name	Description
Peripheral function	*PeripheralFunctionName*.c	Initial function, API function
	*PeripheralFunctionName*_user.c	Interrupt function, callback function
	PeripheralFunctionName 名 .h	Defines macros for assigning values to registers
Project	option.asm	Option bytes, secures ROM for MINICUBE2
	option.inc	Defines macros for setting values in option bytes
	systeminit.c	Call initial function of peripheral function Call CG_ReadResetSource
	main.c	main function
	macrodriver.h	Defines common macros used by all source files
	user_define.h	Empty file (for user definitions)
	lk.dir	Link directive
	ProjectName.prw	Work space for PM+
	ProjectName.prj	Project

2.2 Output File

Below is a list of files output by the Applilet3.

Table 2-1. File List

Peripheral Function	File Name	Names of API Functions Included
System	CG_system.c	CLOCK_Init CG_ChangeClockMode CG_ChangeFrequency CG_SelectPowerSaveMode CG_SelectStabTime CG_SelectPllMode CG_SelectSSCGMode WDT2_Restart CRC_Start CRC_SetData CRC_GetResult
	CG_system_user.c	MD_INTWDT2 CLOCK_UserInit CG_ReadResetSource
	CG_system.h	-
External Bus	CG_bus.c	BUS_Init
	CG_bus_user.c	BUS_UserInit
	CG_bus.h	-
Port	CG_port.c	PORT_Init PORT_ChangePmnInput PORT_ChangePmnOutput
	CG_port_user.c	PORT_UserInit
	CG_port.h	-
INT	CG_int.c	INTP_Init KEY_Init INT_MaskableInterruptEnable INTPn_Disable INTPn_Enable KEY_Disable KEY_Enable
	CG_int_user.c	MD_INTNMI MD_INTP*n* MD_INTKR INTP_UserInit KEY_UserInit
	CG_int.h	-
Serial	CG_serial.c	UARTAn_Init UARTAn_Start UARTAn_Stop UARTAn_SendData UARTAn_ReceiveData UARTBn_Init UARTBn_Start

Peripheral Function	File Name	Names of API Functions Included
Serial	CG_serial.c	UARTBn_Stop UARTBn_SendData UARTBn_ReceiveData CSIBn_Init CSIBn_Start CSIBn_Stop CSIBn_SendData CSIBn_ReceiveData CSIBn_SendReceiveData CSIEn_Init CSIEn_Start CSIEn_Stop CSIEn_SendData CSIEn_ReceiveData CSIEn_SendReceiveData IIC0n_Init IIC0n_Stop IIC0n_StopCondition IIC0n_MasterSendStart IIC0n_MasterReceiveStart IIC0n_SlaveSendStart IIC0n_SlaveReceiveStart
	CG_serial_user.c	MD_INTUA*n*T MD_INTUA*n*R MD_INTUB*n*TIT MD_INTUB*n*TIF MD_INTUB*n*TIR MD_INTUB*n*TIRE MD_INTUB*n*TITO MD_INTCB*n*T MD_INTCB*n*R MD_INTCE*n*T MD_INTCE*n*TIOF MD_INTIIC*n* UARTAn_UserInit UARTAn_SendEndCallback UARTAn_ReceiveEndCallback UARTAn_ErrorCallback UARTAn_SoftOverRunCallback UARTBn_UserInit UARTBn_SendEndCallback UARTBn_ReceiveEndCallback UARTBn_SingleErrorCallback UARTBn_FIFOErrorCallback UARTBn_TimeoutErrorCallback UARTBn_SoftOverRunCallback CSIBn_UserInit CSIBn_SendEndCallback CSIBn_ReceiveEndCallback CSIBn_ErrorCallback

Peripheral Function	File Name	Names of API Functions Included
Serial	CG_serial_user.c	CSIEn_UserInit CSIEn_SendEndCallback CSIEn_ReceiveEndCallback CSIEn_ErrorCallback IIC0n_UserInit IIC0n_MasterSendEndCallback IIC0n_MasterReceiveEndCallback IIC0n_MasterErrorCallback IIC0n_SlaveSendEndCallback IIC0n_SlaveReceiveEndCallback IIC0n_SlaveErrorCallback IIC0n_GetStopConditionCallback
	CG_serial.h	-
A/D	CG_ad.c	AD_Init AD_Start AD_Stop AD_SelectADChannel AD_SetPFTCondition AD_Read AD_ReadByte
	CG_ad_user.c	MD_INTAD AD_UserInit
	CG_ad.h	-
D/A	CG_da.c	DAn_Init DAn_Start DAn_Stop DAn_SetValue
	CG_da_user.c	DAn_UserInit
	CG_da.h	-
Timer	CG_timer.c	TMPn_Init TMPn_Start TMPn_Stop TMPn_ChangeTimerCondition TMPn_GetPulseWidth TMPn_GetFreeRunningValue TMPn_ChangeDuty TMPn_SoftwareTriggerOn TMQ0_Init TMQ0_Start TMQ0_Stop TMQ0_ChangeTimerCondition TMQ0_GetPulseWidth TMQ0_GetFreeRunningValue TMQ0_ChangeDuty TMQ0_SoftwareTriggerOn TMMn_Init TMMn_Start TMMn_Stop

Peripheral Function	File Name	Names of API Functions Included
Timer	CG_timer.c	TMMn_ChangeTimerCondition
	CG_timer_user.c	MD_INTTP*n*OV MD_INTTP*n*CC0 MD_INTTP*n*CC1 MD_INTTQ0OV MD_INTTQ0CC0 MD_INTTQ0CC1 MD_INTTQ0CC2 MD_INTTQ0CC3 MD_INTTM*n*EQ0 TMPn_UserInit TMQ0_UserInit TMMn_UserInit
	CG_timer.h	-
Watch Timer	CG_wt.c	WT_Init WT_Start WT_Stop
	CG_wt_user.c	MD_INTWT MD_INTWTI WT_UserInit
	CG_wt.h	-
RTC	CG_rtc.c	RTC_Init RTC_CounterEnable RTC_CounterDisable RTC_SetHourSystem RTC_CounterSet RTC_CounterGet RTC_ConstPeriodInterruptEnable RTC_ConstPeriodInterruptDisable RTC_AlarmEnable RTC_AlarmDisable RTC_AlarmSet RTC_AlarmGet RTC_IntervalStart RTC_IntervalStop RTC_IntervalInterruptEnable RTC_IntervalInterruptDisable RTC_RC1CK1HZ_OutputEnable RTC_RC1CK1HZ_OutputDisable RTC_RC1CKO_OutputEnable RTC_RC1CKO_OutputDisable RTC_RC1CKDIV_OutputEnable RTC_RC1CKDIV_OutputDisable RTC_ChangeCorrectionValue
	CG_rtc_user.c	MD_INTRTC*n* RTC_UserInit RTC_ConstPeriodInterruptCallback RTC_AlarmInterruptCallback

Peripheral Function	File Name	Names of API Functions Included
RTC	CG_rtc.h	-
Real-Time Output	CG_rto.c	RTOn_Init RTOn_Enable RTOn_Disable RTOn_Set2BitData RTOn_Set4BitData RTOn_Set6BitData RTOn_GetValue
	CG_rto_user.c	RTOn_UserInit
	CG_rto.h	-
DMA	CG_dma.c	DMAn_Init DMAn_Enable DMAn_Disable DMAn_CheckStatus DMAn_SetData DMAn_SoftwareTriggerOn
	CG_dma_user.c	MD_INTDMA*n* DMAn_UserInit
	CG_dma.h	-
LVI	CG_lvi.c	LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Stop
	CG_lvi_user.c	MD_INTLVI LVI_UserInit
	CG_lvi.h	-

CHAPTER 3 API FUNCTIONS

This appendix describes the API functions output by Applilet3.

3.1 Overview

Below are the naming conventions for API functions output by the Applilet3.

- Macro names are in ALL CAPS.
- The number in front of the macro name is a hexadecimal value; this is the same value as the macro value.
- Local variable names are in all lower case.
- Global variable names start with a "g" and use Camel Case.
- Names of pointers to local variables start with a "p" and are in all lower case.
- Names of pointers to global variables start with a "gp" and use Camel Case.
- Names of elements in enum statements are in ALL CAPS.

3.2 Output Function

Below is a list of API functions output by Applilet3.

Table 3-1. API Function List

Peripheral Function	API Function Name	Function
System	CLOCK_Init	Performs initialization necessary to control clock functions.
	CLOCK_UserInit	Performs user-defined initialization relating to the clock.
	CG_ReadResetSource	Performs processing in response to a reset signal.
	CG_ChangeClockMode	Changes the CPU clock.
	CG_ChangeFrequency	Changes the CPU clock division ratio.
	CG_SelectPowerSaveMode	Configures the CPU's standby function.
	CG_SelectStabTime	Selects the oscillation stabilization time for the X1 oscillator. This will become necessary when STOP mode is released.
	CG_SelectPllMode	Selects the operation mode of the PLL function.
	CG_SelectSSCGMode	Selects the operation mode of the SSCG (Spread Spectrum Clock Generator).
	WDT2_Restart	Clears the watchdog timer counter and resumes counting.
	CRC_Start	Begins detection of data-block errors.
	CRC_SetData	Sets data in the CRC input register (CRCIN).
	CRC_GetResult	Reads the results of the calculation stored in the CRC data register (CRCD).
External Bus	BUS_Init	Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).
	BUS_UserInit	Performs user-defined initialization relating to the external bus interface.
Port	PORT_Init	Performs initialization necessary to control port functions.

Peripheral Function	API Function Name	Function
Port	PORT_UserInit	Performs user-defined initialization relating to the port.
	PORT_ChangePmnInput	Switches the pin's I/O mode from output mode to input mode.
	PORT_ChangePmnOutput	Switches the pin's I/O mode from input mode to output mode.
INT	INTP_Init	Performs initialization necessary to control the external interrupt INTP*n* functions.
	INTP_UserInit	Performs user-defined initialization relating to the external interrupt INTP*n* functions.
	KEY_Init	Performs initialization necessary to control the key interrupt INTKR functions.
	KEY_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
	INT_MaskableInterruptEnable	Disables/enables the acceptance of the maskable interrupts.
	INTPn_Disable	Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
	INTPn_Enable	Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
	KEY_Disable	Disables the acceptance of the key interrupts INTKR.
	KEY_Enable	Enables the acceptance of the key interrupts INTKR.
Serial	UARTAn_Init	Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions.
	UARTAn_UserInit	Performs user-defined initialization relating to the asynchronous serial interface A (UARTA).
	UARTAn_Start	Enables asynchronous serial interface A (UARTA).
	UARTAn_Stop	Disables asynchronous serial interface A (UARTA).
	UARTAn_SendData	Starts UARTA*n* data transmission.
	UARTAn_ReceiveData	Starts UARTA*n* data reception.
	UARTAn_SendEndCallback	Performs processing in response to the UARTA*n* consecutive transmission enable interrupt INTUA*n*T.
	UARTAn_ReceiveEndCallback	Performs processing in response to the UARTA*n* reception completion interrupt INTUA*n*R.
	UARTAn_ErrorCallback	Performs processing in response to the UARTA*n* reception error interrupt INTUA*n*R (overrun error, framing error, parity error).
	UARTAn_SoftOverRunCallback	Performs processing in response to detection of overrun error.
	UARTBn_Init	Performs initialization necessary to control the asynchronous serial interface A (UARTB) functions.
	UARTBn_UserInit	Performs user-defined initialization relating to the asynchronous serial interface A (UARTB).
	UARTBn_Start	Enables asynchronous serial interface A (UARTB).
	UARTBn_Stop	Disables asynchronous serial interface A (UARTB).
	UARTBn_SendData	Starts UARTB*n* data transmission.

Peripheral Function	API Function Name	Function
Serial	UARTBn_ReceiveData	Starts UARTB*n* data reception.
	UARTBn_SendEndCallback	Performs processing consequent to the transmission enable interrupt INTUB*n*TIT and the FIFO transmission completion interrupt INTUB*n*TIF.
	UARTBn_ReceiveEndCallback	Performs processing in response to the reception completion interrupt INTUB*n*TIR.
	UARTBn_SingleErrorCallback	Performs processing in response to the reception error interrupt INTUB*n*TIRE (overrun error, framing error, parity error).
	UARTBn_FIFOErrorCallback	Performs processing in response to the reception error interrupt INTUB*n*TIRE (overrun error, framing error, parity error).
	UARTBn_TimeoutErrorCallback	Performs processing in response to the reception timeout error interrupt INTUB*n*TITO.
	UARTBn_SoftOverRunCallback	Performs processing in response to detection of overrun error.
	CSIBn_Init	Performs initialization necessary to control the 3-wire variable-length serial I/O (CSIB) functions.
	CSIBn_UserInit	Performs user-defined initialization relating to the 3-wire variable-length serial I/O (CSIB).
	CSIBn_Start	Enables 3-wire variable-length serial I/O (CSIB).
	CSIBn_Stop	Disables 3-wire variable-length serial I/O (CSIB).
	CSIBn_SendData	Starts CSIB data transmission.
	CSIBn_ReceiveData	Starts CSIB data reception.
	CSIBn_SendReceiveData	Starts CSIB data transmission/reception.
	CSIBn_SendEndCallback	Performs processing in response to the CSIB*n* reception completion interrupt INTCB*n*R or the CSIB*n* consecutive transmission write enable interrupt INTCB*n*T.
	CSIBn_ReceiveEndCallback	Performs processing in response to the CSIB*n* reception completion interrupt INTCB*n*R.
	CSIBn_ErrorCallback	Performs processing in response to the CSIB*n* reception error interrupt INTCB*n*R (overrun error).
	CSIEn_Init	Performs initialization necessary to control the 3-wire variable-length serial I/O (CSIE) functions.
	CSIEn_UserInit	Performs user-defined initialization relating to the 3-wire variable-length serial I/O (CSIE).
	CSIEn_Start	Enables 3-wire variable-length serial I/O (CSIE).
	CSIEn_Stop	Disables 3-wire variable-length serial I/O (CSIE).
	CSIEn_SendData	Starts CSIE data transmission.
	CSIEn_ReceiveData	Starts CSIE data reception.
	CSIEn_SendReceiveData	Starts CSIE data transmission/reception.
	CSIEn_SendEndCallback	Performs processing in response to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.
	CSIEn_ReceiveEndCallback	Performs processing in response to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.

Peripheral Function	API Function Name	Function
Serial	CSIEn_ErrorCallback	Performs processing in response to the CSIE*n*BUF overflow interrupt INTCE*n*TIOF.
	IIC0n_Init	Performs initialization necessary to control the IIC bus functions.
	IIC0n_UserInit	Performs user-defined initialization relating to the IIC bus.
	IIC0n_Stop	Ends IIC0*n* communication.
	IIC0n_StopCondition	Generates a stop condition.
	IIC0n_MasterSendStart	Starts IIC0*n* master transmission.
	IIC0n_MasterReceiveStart	Starts IIC0*n* master reception.
	IIC0n_MasterSendEndCallback	Performs processing in response to the IIC*n* master transfer completion interrupt INTIIC*n*.
	IIC0n_MasterReceiveEndCallback	Performs processing in response to the IIC*n* master transfer completion interrupt INTIIC*n*.
	IIC0n_MasterErrorCallback	Performs processing in response to detection of error in IIC*n* master communication.
	IIC0n_SlaveSendStart	Starts IIC0*n* slave transmission.
	IIC0n_SlaveReceiveStart	Starts IIC0*n* slave reception.
	IIC0n_SlaveSendEndCallback	Performs processing in response to the IIC*n* slave transfer completion interrupt INTIIC*n*.
	IIC0n_SlaveReceiveEndCallback	Performs processing in response to the IIC*n* slave transfer completion interrupt INTIIC*n*.
	IIC0n_SlaveErrorCallback	Performs processing in response to detection of error in IIC*n* slave communication.
	IIC0n_GetStopConditionCallback	Performs processing in response to detection of stop condition.
A/D	AD_Init	Performs initialization necessary to control A/D converter functions.
	AD_UserInit	Performs user-defined initialization relating to the A/D converter.
	AD_Start	Starts A/D conversion.
	AD_Stop	Ends A/D conversion.
	AD_SelectADChannel	Configures the analog voltage input pin for A/D conversion.
	AD_SetPFTCondition	Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger).
	AD_Read	Reads the results of A/D conversion (10 bits).
	AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).
D/A	DAn_Init	Performs initialization necessary to control D/A converter functions.
	DAn_UserInit	Performs user-defined initialization relating to the D/A converter.
	DAn_Start	Starts D/A conversion.

Peripheral Function	API Function Name	Function
D/A	DAn_Stop	Ends D/A conversion.
	DAn_SetValue	Sets the analog voltage output to the ANO*n* pin.
Timer	TMPn_Init	Performs initialization necessary to control timer "TMP*n*" functions.
	TMPn_UserInit	Performs user-defined initialization relating to the timer "TMP*n*".
	TMPn_Start	Starts the count for timer "TMP*n*".
	TMPn_Stop	Ends the count for timer "TMP*n*".
	TMPn_ChangeTimerCondition	Changes the counter value for timer "TMP*n*".
	TMPn_GetPulseWidth	Reads the pulse width of timer TMP*n* (high/low level width).
	TMPn_GetFreeRunningValue	Reads the value captured by timer TMP*n*.
	TMPn_ChangeDuty	Changes the duty ratio of the PWM signal.
	TMPn_SoftwareTriggerOn	Generates the trigger (software trigger) for timer output.
	TMQ0_Init	Performs initialization necessary to control timer "TMQ0" functions.
	TMQ0_UserInit	Performs user-defined initialization relating to the timer "TMQ0".
	TMQ0_Start	Starts the count for timer "TMQ0".
	TMQ0_Stop	Ends the count for timer "TMQ0".
	TMQ0_ChangeTimerCondition	Changes the counter value for timer "TMQ0".
	TMQ0_GetPulseWidth	Reads the pulse width of timer TMQ0 (high/low level width).
	TMQ0_GetFreeRunningValue	Reads the value captured by timer TMQ0.
	TMQ0_ChangeDuty	Changes the duty ratio of the PWM signal.
	TMQ0_SoftwareTriggerOn	Generates the trigger (software trigger) for timer output.
	TMMn_Init	Performs initialization necessary to control timer "TMM*n*" functions.
	TMMn_UserInit	Performs user-defined initialization relating to the timer "TMM*n*".
	TMMn_Start	Starts the count for timer "TMM*n*".
	TMMn_Stop	Ends the count for timer "TMM*n*".
	TMMn_ChangeTimerCondition	Changes the counter value for timer "TMM*n*".
Watch Timer	WT_Init	Performs initialization necessary to control watch timer functions.
	WT_UserInit	Performs user-defined initialization relating to the watch timer.
	WT_Start	Clears the watch timer counter and resumes counting.
	WT_Stop	Ends the count for watch timer.
RTC	RTC_Init	Performs initialization necessary to control real-time counter functions.

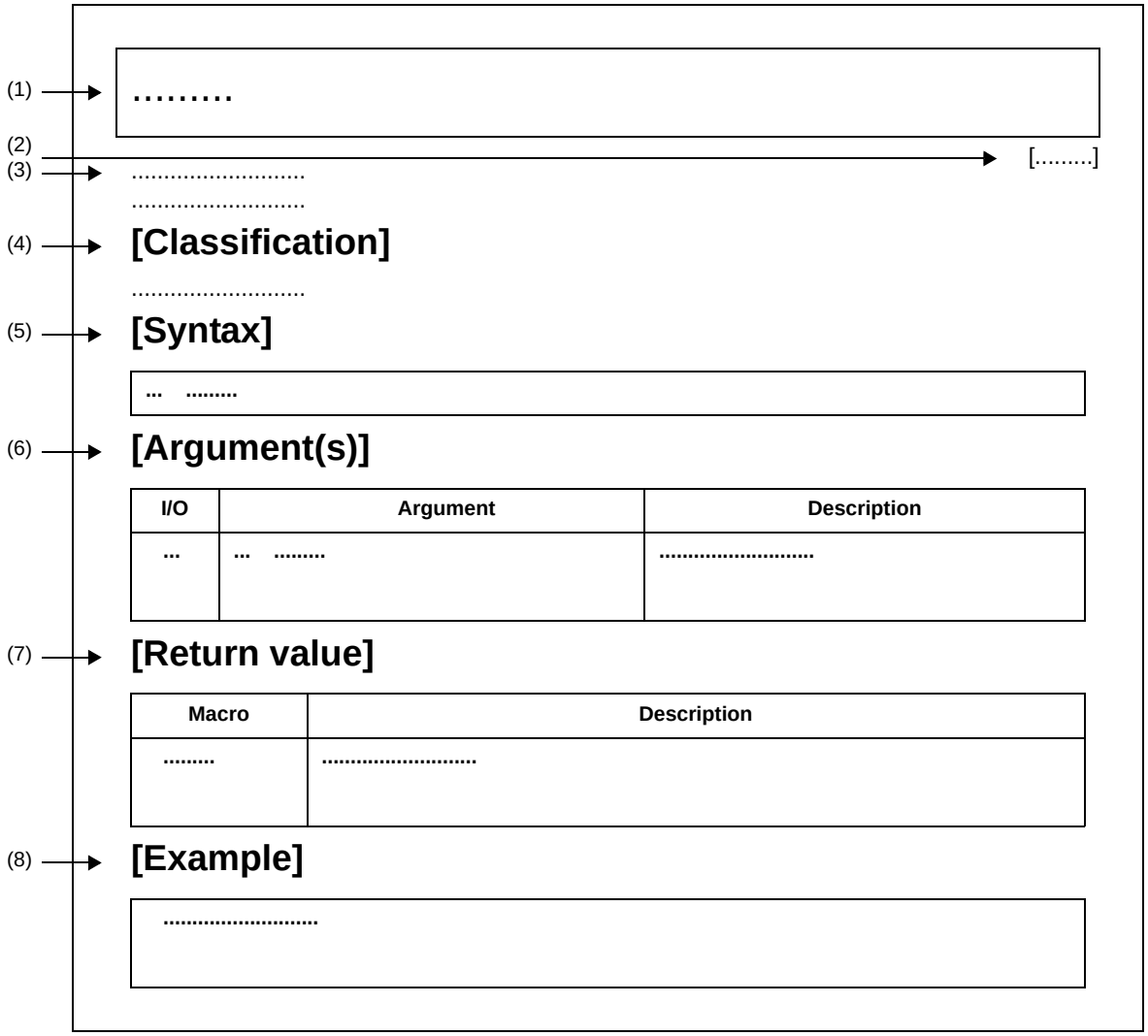
Peripheral Function	API Function Name	Function
RTC	RTC_UserInit	Performs user-defined initialization relating to the real-time counter.
	RTC_CounterEnable	Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).
	RTC_CounterDisable	Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).
	RTC_SetHourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time counter.
	RTC_CounterSet	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_CounterGet	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_ConstPeriodInterruptEnable	Sets the cycle of the interrupts INTRTC0, then starts the cyclic interrupt function.
	RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
	RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC0.
	RTC_AlarmEnable	Starts the alarm interrupt function.
	RTC_AlarmDisable	Ends the alarm interrupt function.
	RTC_AlarmSet	Sets the alarm conditions (weekday, hour, minute).
	RTC_AlarmGet	Reads the alarm conditions (weekday, hour, minute).
	RTC_AlarmInterruptCallback	Performs processing in response to the alarm interrupt INTRTC1.
	RTC_IntervalStart	Starts the interval interrupt function.
	RTC_IntervalStop	Ends the interval interrupt function.
	RTC_IntervalInterruptEnable	Sets the cycle of the interrupts INTRTC2, then starts the interval interrupt function.
	RTC_IntervalInterruptDisable	Ends the interval interrupt function.
	RTC_RC1CK1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
	RTC_RC1CK1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
	RTC_RC1CKO_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RC1CKO_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RC1CKDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
	RTC_RC1CKDIV_OutputDisable	Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
	RTC_ChangeCorrectionValue	Changes the timing and correction value for correcting clock errors.
Real-Time Output	RTOn_Init	Performs initialization necessary to control real-time output functions.
	RTOn_UserInit	Performs user-defined initialization relating to the real-time output.

Peripheral Function	API Function Name	Function
Real-Time Output	RTOn_Enable	Enables (validates) real-time output.
	RTOn_Disable	Disables (invalidates) real-time output.
	RTOn_Set2BitData	Sets 2-bit data for real-time output.
	RTOn_Set4BitData	Sets 4-bit data for real-time output.
	RTOn_Set6BitData	Sets 6-bit data for real-time output.
	RTOn_GetValue	Reads data from real-time output.
DMA	DMAn_Init	Performs initialization necessary to control DMA controller functions.
	DMAn_UserInit	Performs user-defined initialization relating to the DMA controller.
	DMAn_Enable	Enables operation of channel *n*.
	DMAn_Disable	Disables operation of channel *n*.
	DMAn_CheckStatus	Reads the transfer status (transfer complete/transfer ongoing).
	DMAn_SetData	Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.
	DMAn_SoftwareTriggerOn	Uses a software trigger as a DMA transfer start trigger.
LVI	LVI_Init	Performs initialization necessary to control low-voltage detector functions.
	LVI_UserInit	Performs user-defined initialization relating to the low-voltage detector.
	LVI_InterruptModeStart	Starts low-voltage detection (when in interrupt generation mode).
	LVI_ResetModeStart	Starts low-voltage detection (when in internal reset mode).
	LVI_Stop	Stops low-voltage detection.

3.3 Function Reference

This section describes the API functions output by the Applilet3, using the following notation format.

Figure 3-1. Notation Format of API Functions



(1) Name

Indicates the name of the API function.

(2) Taget device

Indicates the name of the device targeted by API function output.

(3) Outline

Outlines the functions of the API function.

(4) [Classification]

Indicates the name of the C source file to which the API function is output.

(5) [Syntax]

Indicates the format to be used when describing an API function to be called in C language.

(6) [Argument(s)]

API function arguments are explained in the following format.

I/O	Argument	Description
(a)	(b)	(c)

(a) I/O

Argument classification

I ... input argument

O ... Output argument

(b) Argument

Argument data type

(c) Description

Description of argument

(7) [Return value]

Indicates an API function call's return value using a macro and value.

(8) [Example]

Shows an example of the API function in use.

Caution The sample programs in the [Example] section assume that NEC Electronics' PM+ is being used as the integrated development environment for the application system.

3.3.1 System

Below is a list of API functions output by Applilet3 for system use.

Table 3-2. API Functions: [System]

API Function Name	Function
CLOCK_Init	Performs initialization necessary to control clock functions.
CLOCK_UserInit	Performs user-defined initialization relating to the clock.
CG_ReadResetSource	Performs processing in response to a reset signal.
CG_ChangeClockMode	Changes the CPU clock.
CG_ChangeFrequency	Changes the CPU clock division ratio.
CG_SelectPowerSaveMode	Configures the CPU's standby function.
CG_SelectStabTime	Selects the oscillation stabilization time for the X1 oscillator. This will become necessary when STOP mode is released.
CG_SelectPllMode	Selects the operation mode of the PLL function.
CG_SelectSSCGMode	Selects the operation mode of the SSCG (Spread Spectrum Clock Generator).
WDT2_Restart	Clears the watchdog timer counter and resumes counting.
CRC_Start	Begins detection of data-block errors.
CRC_SetData	Sets data in the CRC input register (CRCIN).
CRC_GetResult	Reads the results of the calculation stored in the CRC data register (CRCD).

CLOCK_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control clock functions.

[Classification]

CG_system.c

[Syntax]

```
void    CLOCK_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

CLOCK_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the clock.

Remark This API function is called as the CLOCK_Init callback routine.

[Classification]

CG_system_user.c

[Syntax]

```
void    CLOCK_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

CG_ReadResetSource

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to a reset signal.

[Classification]

CG_system_user.c

[Syntax]

```
void    CG_ReadResetSource ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example]

Below are examples of the different processes executing depending on the reset signal trigger.

[CG_Systeminit.c]

```
void systeminit ( void ) {
    CG_ReadResetSource ();  /* Perform process according to reset signal trigger */
    ......
}
```

[CG_system_user.c]

```
#include    "CG_macrodriver.h"
void CG_ReadResetSource ( void ) {
    UCHAR   resetflag = RESF;   /* Reset control flag register: Obtain RESF contents */
    if ( resetflag & 0x1 ) {            /* Trigger identification: Check LVIRF flag */
        ......  /* Process performed when low-voltage detector detects low voltage */
    } else if ( resetflag & 0x2 ) {     /* Trigger identification: Check CLMRF flag */
        ......  /* Process performed when clock monitor oscillation stopped */
    } else if ( resetflag & 0x10 ) {    /* Trigger identification: Check WDT2RF flag */
        ......  /* Process performed when watchdog timer 2 overflows */
    }
    ......
}
```

CG_ChangeClockMode

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Changes the CPU clock.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_ChangeClockMode ( enum ClockMode mode );
```

[Argument(s)]

I/O	Argument	Description
I	`enum ClockMode mode;`	CPU clock type MAINOSCCLK: Main clock oscillator (fXX) SUBCLK: Subclock oscillator (fXT)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend) - Cannot change from the subclock oscillator (fXT) to the main clock oscillator (fXX).
MD_ERROR2	Exit with error (abend) - Cannot change from the main clock oscillator (fXX) to the subclock oscillator (fXT).
MD_ARGERROR	Invalid argument specification

CG_ChangeFrequency

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Changes the CPU clock division ratio.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_ChangeFrequency ( enum CPUClock clock );
```

[Argument(s)]

I/O	Argument	Description
I	`enum CPUClock clock;`	Division ratio type SYSTEMCLOCK: fxx SYSONEHALF: fxx/2 SYSONEFOURTH: fxx/4 SYSONEEIGHTH: fxx/8 SYSONESIXTEENTH: fxx/16 SYSONETHIRTYSECOND: fxx/32

Remark "fxx" signifies the frequency of the main clock oscillator.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend)
MD_ARGERROR	Invalid argument specification

CG_SelectPowerSaveMode

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Configures the CPU's standby function.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectPowerSaveMode ( enum PSLevel level );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PSLevel level;`	Standby function type PSSTOP: STOP mode PSHALT: HALT mode PSIDLE1: IDLE1 mode PSIDLE2: IDLE2 mode [ES/Jx3] [ES/Jx3-L] PSSTOP: STOP mode PSHALT: HALT mode

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CG_SelectStabTime

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Selects the oscillation stabilization time for the X1 oscillator. This will become necessary when STOP mode is released.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectStabTime ( enum StabTime waittime );
```

[Argument(s)]

I/O	Argument	Description
I	`enum StabTime waittime;`	Oscillation stabilization time type STLEVEL0: 2^10/fx STLEVEL1: 2^11/fx STLEVEL2: 2^12/fx STLEVEL3: 2^13/fx STLEVEL4: 2^14/fx STLEVEL5: 2^15/fx STLEVEL6: 2^16/fx

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CG_SelectPllMode

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Selects the operation mode of the PLL function.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectPllMode ( enum PllMode pllmode );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PllMode pllmode;`	Operation mode type SYSPLLOFF: Clock-through mode SYS4PLL: x4 (When PLL function is used) SYS8PLL: x8 (When PLL function is used)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) [ES/Jx3] [ES/Jx3-L]
MD_ERROR1	Exit with error (abend) [E/Sx3-H] - Cannot change the operation mode.
MD_ERROR2	Exit with error (abend) [E/Sx3-H] - Cannot change to the x4.
MD_ERROR3	Exit with error (abend) [E/Sx3-H] - Cannot change to the x8.
MD_ARGERROR	Invalid argument specification

CG_SelectSSCGMode

[E/Sx3-H]

Selects the operation mode of the SSCG (Spread Spectrum Clock Generator).

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectSSCGMode ( enum SSCGMode sscgmode );
```

[Argument(s)]

I/O	Argument	Description
I	`enum SSCGMode sscgmode;`	Operation mode type SYSSSCGON: SSCG operation enabled SYSSSCGOFF: SSCG operation stopped

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend)
MD_ARGERROR	Invalid argument specification

WDT2_Restart

[]E/Sx3-H [ES/Jx3] [ES/Jx3-L]

Clears the watchdog timer counter and resumes counting.

[Classification]

CG_system.c

[Syntax]

```
void    WDT2_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

CRC_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Begins detection of data-block errors.

[Classification]

CG_system.c

[Syntax]

```
void    CRC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

CRC_SetData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Sets data in the CRC input register (CRCIN).

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    CRC_SetData ( UCHAR data );
```

[Argument(s)]

I/O	Argument	Description
I	UCHAR *data*;	Data to set

[Return value]

None.

CRC_GetResult

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the results of the calculation stored in the CRC data register (CRCD).

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    CRC_GetResult ( USHORT *result );
```

[Argument(s)]

I/O	Argument	Description
O	`USHORT *result;`	Pointer to area in which to store read calculation results

[Return value]

None.

3.3.2 External Bus

Below is a list of API functions output by Applilet3 for external bus interface use.

Table 3-3. API Functions: [External Bus]

API Function Name	Function
BUS_Init	Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).
BUS_UserInit	Performs user-defined initialization relating to the external bus interface.

BUS_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).

[Classification]

CG_bus.c

[Syntax]

```
void    BUS_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

BUS_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the external bus interface.

Remark This API function is called as the BUS_Init callback routine.

[Classification]

CG_bus_user.c

[Syntax]

```
void    BUS_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.3 Port

Below is a list of API functions output by Applilet3 for port use.

Table 3-4. API Functions: [Port]

API Function Name	Function
PORT_Init	Performs initialization necessary to control port functions.
PORT_UserInit	Performs user-defined initialization relating to the port.
PORT_ChangePmnInput	Switches the pin's I/O mode from output mode to input mode.
PORT_ChangePmnOutput	Switches the pin's I/O mode from input mode to output mode.

PORT_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control port functions.

[Classification]

CG_port.c

[Syntax]

```
void    PORT_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

PORT_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the port.

Remark This API function is called as the PORT_Init callback routine.

[Classification]

CG_port_user.c

[Syntax]

```
void    PORT_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

PORT_ChangeP*mn*Input

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Switches the pin's I/O mode from output mode to input mode.

[Classification]

CG_port.c

[Syntax]

```
void    PORT_ChangeP mn Input ( void );
```

Remark *mn* is the port number.

[Argument(s)]

None.

[Return value]

None.

[Example]

Below is an example of switching the P00 pin's I/O mode from output mode to input mode.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Input ( TRUE );   /* Switch I/O mode */
    ......
}
```

PORT_ChangeP*mn*Output

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Switches the pin's I/O mode from input mode to output mode.

[Classification]

CG_port.c

[Syntax]

The format for specifying this API function differs according to whether the target pin conducts N-ch open drain output.

[N-ch open drain output: none]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL initialvalue );
```

[N-ch open drain output: yes]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL enablench, BOOL initialvalue );
```

Remark *nm* is the port number.

[Argument(s)]

I/O	Argument	Description
I	`BOOL enablench;`	Output mode type MD_TRUE: N-ch open drain output (V_{DD} withstand voltage) mode MD_FALSE: Normal output mode
I	`BOOL initialvalue;`	Initial output value MD_SET: Output HIGH level "1" MD_CLEAR: Output LOW level "0"

[Return value]

None.

[Example 1]

Below is shown an example where pin P00 (N-ch open drain output: none) is changed as follows:

I/O mode type: Output mode
Initial output value: Output HIGH level "1"

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Output ( MD_SET );    /* Switch I/O mode */
```

```
    ......
}
```

[Example 2]

Below is shown an example where pin P04 (N-ch open drain output: yes) is changed as follows:

I/O mode type: Output mode
Output mode type: N-ch open drain output (V_{DD} withstand voltage) mode
Initial output value: Output LOW level "0"

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP04Output ( MD_TRUE, MD_CLEAR ); /* Switch I/O mode */
    ......
}
```

3.3.4 INT

Below is a list of API functions output by Applilet3 for interrupt and key interrupt use.

Table 3-5. API Functions: [INT]

API Function Name	Function
INTP_Init	Performs initialization necessary to control the external interrupt INTP*n* functions.
INTP_UserInit	Performs user-defined initialization relating to the external interrupt INTP*n* functions.
KEY_Init	Performs initialization necessary to control the key interrupt INTKR functions.
KEY_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
INT_MaskableInterruptEnable	Disables/enables the acceptance of the maskable interrupts.
INTPn_Disable	Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
INTPn_Enable	Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
KEY_Disable	Disables the acceptance of the key interrupts INTKR.
KEY_Enable	Enables the acceptance of the key interrupts INTKR.

INTP_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control the external interrupt INTP*n* functions.

[Classification]

CG_int.c

[Syntax]

```
void    INTP_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

INTP_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the external interrupt INTP*n* functions.

Remark This API function is called as the INTP_Init callback routine.

[Classification]

CG_int_user.c

[Syntax]

```
void    INTP_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

KEY_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control the key interrupt INTKR functions.

[Classification]

CG_int.c

[Syntax]

```
void    KEY_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

KEY_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the key interrupt INTKR functions.

Remark This API function is called as the KEY_Init callback routine.

[Classification]

CG_int_user.c

[Syntax]

```
void    KEY_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

INT_MaskableInterruptEnable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Disables/enables the acceptance of the maskable interrupts.

[Classification]

CG_int.c

[Syntax]

- [E/Sx3-H]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
MD_STATUS   INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

- [ES/Jx3] [ES/Jx3-L]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void    INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

[Argument(s)]

I/O	Argument	Description
I	`enum MaskableSource name;`	Maskable interrupt type INT_*xxx*: Maskable interrupt
I	`BOOL enableflag;`	Acceptance enabled/disabled MD_TRUE: Acceptance enabled MD_FALSE: Acceptance disabled

Remark See the header file CG_int.h for details about the maskable interrupt type INT_*xxx*.

[Return value]

- [E/Sx3-H]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

- [ES/Jx3] [ES/Jx3-L]

None.

[Example 1]

Below is an example of disabling acceptance of the maskable interrupt INTP0.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ......
    INT_MaskableInterruptEnable ( INT_INTP0, MD_FALSE );  /* Disable acceptance of maskable
interrupt INTP0 */
    ......
}
```

[Example 2]

Below is an example of enabling acceptance of the maskable interrupt INTP0.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ......
    INT_MaskableInterruptEnable ( INT_INTP0, MD_TRUE );   /* Enable acceptance of maskable
interrupt INTP0 */
    ......
}
```

INTP*n*_Disable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).

[Classification]

CG_int.c

[Syntax]

```
void    INTPn_Disable ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

INTP*n*_Enable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).

[Classification]

CG_int.c

[Syntax]

```
void    INTPn_Enable ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

KEY_Disable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Disables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

```
void    KEY_Disable ( void );
```

[Argument(s)]

None.

[Return value]

None.

KEY_Enable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Enables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

```
void    KEY_Enable ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.5 Serial

Below is a list of API functions output by Applilet3 for serial use.

Table 3-6. API Functions: [Serial]

API Function Name	Function
UARTAn_Init	Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions.
UARTAn_UserInit	Performs user-defined initialization relating to the asynchronous serial interface A (UARTA).
UARTAn_Start	Enables asynchronous serial interface A (UARTA).
UARTAn_Stop	Disables asynchronous serial interface A (UARTA).
UARTAn_SendData	Starts UARTA*n* data transmission.
UARTAn_ReceiveData	Starts UARTA*n* data reception.
UARTAn_SendEndCallback	Performs processing in response to the UARTA*n* consecutive transmission enable interrupt INTUA*n*T.
UARTAn_ReceiveEndCallback	Performs processing in response to the UARTA*n* reception completion interrupt INTUA*n*R.
UARTAn_ErrorCallback	Performs processing in response to the UARTA*n* reception error interrupt INTUA*n*R (overrun error, framing error, parity error).
UARTAn_SoftOverRunCallback	Performs processing in response to detection of overrun error.
UARTBn_Init	Performs initialization necessary to control the asynchronous serial interface A (UARTB) functions.
UARTBn_UserInit	Performs user-defined initialization relating to the asynchronous serial interface A (UARTB).
UARTBn_Start	Enables asynchronous serial interface A (UARTB).
UARTBn_Stop	Disables asynchronous serial interface A (UARTB).
UARTBn_SendData	Starts UARTB*n* data transmission.
UARTBn_ReceiveData	Starts UARTB*n* data reception.
UARTBn_SendEndCallback	Performs processing consequent to the transmission enable interrupt INTUB*n*TIT and the FIFO transmission completion interrupt INTUB*n*TIF.
UARTBn_ReceiveEndCallback	Performs processing in response to the reception completion interrupt INTUB*n*TIR.
UARTBn_SingleErrorCallback	Performs processing in response to the reception error interrupt INTUB*n*TIRE (overrun error, framing error, parity error).
UARTBn_FIFOErrorCallback	Performs processing in response to the reception error interrupt INTUB*n*TIRE (overrun error, framing error, parity error).
UARTBn_TimeoutErrorCallback	Performs processing in response to the reception timeout error interrupt INTUB*n*TITO.
UARTBn_SoftOverRunCallback	Performs processing in response to detection of overrun error.
CSIBn_Init	Performs initialization necessary to control the 3-wire variable-length serial I/O (CSIB) functions.
CSIBn_UserInit	Performs user-defined initialization relating to the 3-wire variable-length serial I/O (CSIB).
CSIBn_Start	Enables 3-wire variable-length serial I/O (CSIB).
CSIBn_Stop	Disables 3-wire variable-length serial I/O (CSIB).
CSIBn_SendData	Starts CSIB data transmission.

API Function Name	Function
CSIBn_ReceiveData	Starts CSIB data reception.
CSIBn_SendReceiveData	Starts CSIB data transmission/reception.
CSIBn_SendEndCallback	Performs processing in response to the CSIB*n* reception completion interrupt INTCB*n*R or the CSIB*n* consecutive transmission write enable interrupt INTCB*n*T.
CSIBn_ReceiveEndCallback	Performs processing in response to the CSIB*n* reception completion interrupt INTCB*n*R.
CSIBn_ErrorCallback	Performs processing in response to the CSIB*n* reception error interrupt INTCB*n*R (overrun error).
CSIEn_Init	Performs initialization necessary to control the 3-wire variable-length serial I/O (CSIE) functions.
CSIEn_UserInit	Performs user-defined initialization relating to the 3-wire variable-length serial I/O (CSIE).
CSIEn_Start	Enables 3-wire variable-length serial I/O (CSIE).
CSIEn_Stop	Disables 3-wire variable-length serial I/O (CSIE).
CSIEn_SendData	Starts CSIE data transmission.
CSIEn_ReceiveData	Starts CSIE data reception.
CSIEn_SendReceiveData	Starts CSIE data transmission/reception.
CSIEn_SendEndCallback	Performs processing in response to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.
CSIEn_ReceiveEndCallback	Performs processing in response to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.
CSIEn_ErrorCallback	Performs processing in response to the CSIE*n*BUF overflow interrupt INTCE*n*TIOF.
IIC0n_Init	Performs initialization necessary to control the IIC bus functions.
IIC0n_UserInit	Performs user-defined initialization relating to the IIC bus.
IIC0n_Stop	Ends IIC0*n* communication.
IIC0n_StopCondition	Generates a stop condition.
IIC0n_MasterSendStart	Starts IIC0*n* master transmission.
IIC0n_MasterReceiveStart	Starts IIC0*n* master reception.
IIC0n_MasterSendEndCallback	Performs processing in response to the IIC0*n* master transfer completion interrupt INTIIC*n*.
IIC0n_MasterReceiveEndCallback	Performs processing in response to the IIC0*n* master transfer completion interrupt INTIIC*n*.
IIC0n_MasterErrorCallback	Performs processing in response to detection of error in IIC0*n* master communication.
IIC0n_SlaveSendStart	Starts IIC0*n* slave transmission.
IIC0n_SlaveReceiveStart	Starts IIC0*n* slave reception.
IIC0n_SlaveSendEndCallback	Performs processing in response to the IIC0*n* slave transfer completion interrupt INTIIC*n*.
IIC0n_SlaveReceiveEndCallback	Performs processing in response to the IIC0*n* slave transfer completion interrupt INTIIC*n*.
IIC0n_SlaveErrorCallback	Performs processing in response to detection of error in IIC0*n* slave communication.

API Function Name	Function
IIC0n_GetStopConditionCallback	Performs processing in response to detection of stop condition.

UARTA*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTAn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTA*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the asynchronous serial interface A (UARTA).

Remark This API function is called as the UARTAn_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTAn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTA*n*_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Enables asynchronous serial interface A (UARTA).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTA*n*_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Disables asynchronous serial interface A (UARTA).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTA*n*_SendData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts UARTA*n* data transmission.

Remarks 1. This API function repeats the byte-level UARTA*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. When performing a UARTA*n* transmission, UARTAn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTAn_SendData ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *txbuf;	Pointer to a buffer storing the transmission data
I	USHORT txnum;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification
MD_DATAEXISTS	Exit with error (abend) - Hold a next transmission data in the UA*n*TX register.

UARTA*n*_ReceiveData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts UARTA*n* data reception.

Remarks 1. This API function performs byte-level UARTA*n* reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. Actual UARTA*n* reception starts after this API function is called, and UARTAn_Start is then called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTAn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

UARTA*n*_SendEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the UARTA*n* consecutive transmission enable interrupt INTUA*n*T.

Remark This API function is called as the callback routine of interrupt process MD_INTUA*n*T corresponding to the UARTA*n* consecutive transmission enable interrupt INTUA*n*T (performed when total number of UARTA*n* transmissions specified by UARTAn_SendData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTAn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTA*n*_ReceiveEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the UARTA*n* reception completion interrupt INTUA*n*R.

Remark This API function is called as the callback routine of interrupt process MD_INTUA*n*R corresponding to the UARTA*n* reception completion interrupt INTUA*n*R (performed when total number of UARTA*n* receptions specified by UARTAn_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTAn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTA*n*_ErrorCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the UARTA*n* reception error interrupt INTUA*n*R (overrun error, framing error, parity error).

Remark This API function is called as the callback routine of interrupt process MD_INTUA*n*R corresponding to the UARTA*n* reception error interrupt INTUA*n*R.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTAn_ErrorCallback ( UCHAR err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	UCHAR *err_type*;	Trigger for UARTA*n* reception error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Framing error 000001*xx*B: Parity error

[Return value]

None.

UARTA*n*_SoftOverRunCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process MD_INTUA*n*R corresponding to the UARTA*n* reception error interrupt INTUA*n*R (process performed when the amount of data received is greater than the parameter *rxnum* specified for UARTAn_ReceiveData).

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTAn_SoftOverRunCallback ( UCHAR rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR rx_data;`	Received data

[Return value]

None.

UARTB*n*_Init

[E/Sx3-H]

Performs initialization necessary to control the asynchronous serial interface A (UARTB) functions.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTBn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB*n*_UserInit

[E/Sx3-H]

Performs user-defined initialization relating to the asynchronous serial interface A (UARTB).

Remark This API function is called as the UARTBn_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTBn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB*n*_Start

[E/Sx3-H]

Enables asynchronous serial interface A (UARTB).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTBn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB*n*_Stop

[E/Sx3-H]

Disables asynchronous serial interface A (UARTB).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTBn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB*n*_SendData

[E/Sx3-H]

Starts UARTB*n* data transmission.

Remarks 1. This API function repeats the byte-level UARTB*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. When performing a UARTB*n* transmission (single mode), UARTBn_Start must be called before this API function is called.

3. When performing a UARTB*n* transmission (FIFO mode), UARTBn_Start must be called after this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTBn_SendData ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *txbuf;	Pointer to a buffer storing the transmission data
I	USHORT txnum;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification
MD_DATAEXISTS	Exit with error (abend) - Hold a next transmission data in the UB*n*TX register.

UARTB*n*_ReceiveData

[E/Sx3-H]

Starts UARTB*n* data reception.

Remarks 1. This API function performs byte-level UARTB*n* reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. Actual UARTB*n* reception starts after this API function is called, and UARTBn_Start is then called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTBn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

UARTB*n*_SendEndCallback

[E/Sx3-H]

Performs processing consequent to the transmission enable interrupt INTUB*n*TIT and the FIFO transmission completion interrupt INTUB*n*TIF.

Remark This API function is called as a callback routine of the interrupt process MD_INTUB*n*TIT corresponding to a transmission enable interrupt INTUB*n*TIT of UARTB*n* (single mode), and interrupt process MD_INTUB*n*TIF corresponding to a FIFO transmission completion interrupt INTUB*n*TIF of UARTB*n* (FIFO mode) (performed when total number of UARTB*n* transmissions specified by UARTBn_SendData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTBn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB*n*_ReceiveEndCallback

[E/Sx3-H]

Performs processing in response to the reception completion interrupt INTUB*n*TIR.

Remark This API function is called as the callback routine of interrupt process MD_INTUB*n*TIR corresponding to the reception completion interrupt INTUB*n*TIR (performed when total number of UARTB*n* receptions specified by UARTBn_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTBn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB*n*_SingleErrorCallback

[E/Sx3-H]

Performs processing in response to the reception error interrupt INTUB*n*TIRE (overrun error, framing error, parity error).

Remark This API function is called as a callback routine of the interrupt process MD_INTUB*n*TIRE corresponding to a reception error interrupt INTUBA*n*TIRE of UARTB*n* (single mode).

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTBn_SingleErrorCallback ( UCHAR err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	UCHAR *err_type*;	Trigger for reception error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Framing error 000001*xx*B: Parity error

[Return value]

None.

UARTB*n*_FIFOErrorCallback

[E/Sx3-H]

Performs processing in response to the reception error interrupt INTUB*n*TIRE (overrun error, framing error, parity error).

Remark This API function is called as a callback routine of the interrupt process MD_INTUB*n*TIRE corresponding to a reception error interrupt INTUBA*n*TIRE of UARTB*n* (FIFO mode).

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTBn_FIFOErrorCallback ( UCHAR err_type1, UCHAR err_type2 );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	UCHAR *err_type1*;	Trigger for reception error interrupt 00001000B: Overrun error
O	UCHAR *err_type2*;	Trigger for reception error interrupt 000000*x*1B: Framing error 0000001*x*B: Parity error

[Return value]

None.

UARTB*n*_TimeoutErrorCallback

[E/Sx3-H]

Performs processing in response to the reception timeout error interrupt INTUB*n*TITO.

Remark This API function is called as a callback routine of the interrupt process MD_INTUB*n*TITO corresponding to a reception timeout interrupt INTUBA*n*TITO of UARTB*n* (FIFO mode).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTBn_TimeoutErrorCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB*n*_SoftOverRunCallback

[E/Sx3-H]

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process MD_INTUB*n*TIRE corresponding to the reception error interrupt INTUB*n*TIRE (process performed when the amount of data received is greater than the parameter *rxnum* specified for UARTBn_ReceiveData).

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTBn_SoftOverRunCallback ( UCHAR rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR rx_data;`	Received data

[Return value]

None.

CSIB*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control the 3-wire variable-length serial I/O (CSIB) functions.

[Classification]

CG_serial.c

[Syntax]

```
void    CSIBn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the 3-wire variable-length serial I/O (CSIB).

Remark This API function is called as the CSIBn_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSIBn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB*n*_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Enables 3-wire variable-length serial I/O (CSIB).

[Classification]

CG_serial.c

[Syntax]

```
void    CSIBn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB*n*_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Disables 3-wire variable-length serial I/O (CSIB).

[Classification]

CG_serial.c

[Syntax]

```
void    CSIBn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB*n*_SendData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts CSIB*n* data transmission.

Remarks 1. This API function repeats the byte-level CSIB*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. When performing a CSIB*n* transmission, CSIBn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    “CG_macrodriver.h“
MD_STATUS   CSIBn_SendData ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *txbuf;	Pointer to a buffer storing the transmission data
I	USHORT txnum;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CSIB*n*_ReceiveData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts CSIB*n* data reception.

Remarks 1. This API function performs byte-level CSIB*n* reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. When performing a CSIB*n* reception, CSIBn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIBn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CSIB*n*_SendReceiveData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts CSIB*n* data transmission/reception.

Remarks 1. This API function repeats the byte-level CSIB*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. This API function performs byte-level CSIB*n* reception the number of times specified by the parameter *txnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

3. When performing a CSIB*n* transmission/reception, CSIBn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIBn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *txbuf;`	Pointer to a buffer to store the received data
I	`USHORT txnum;`	Total amount of data to receive
I	`UCHAR *rxbuf;`	Pointer to a buffer storing the transmission data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CSIB*n*_SendEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the CSIB*n* reception completion interrupt INTCB*n*R or the CSIB*n* consecutive transmission write enable interrupt INTCB*n*T.

Remark This API function is called as the callback routine (process performed when the total number of CSIB*n* transmissions specified in the parameter *txnum* for CSIBn_SendData or CSIBn_SendReceiveData has been completed) of interrupt process MD_INTCB*n*R corresponding to the CSIB*n* reception completion interrupt INTCB*n*R, and interrupt process MD_INTCB*n*T corresponding to the CSIB*n* consecutive transmission write enable interrupt INTCB*n*T.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSIBn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB*n*_ReceiveEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the CSIB*n* reception completion interrupt INTCB*n*R.

Remark This API function is called as the callback routine (process performed when the total number of CSIB*n* receptions specified in the parameter *rxnum* for CSIBn_ReceiveData or CSIBn_SendReceiveData has been completed) of interrupt process MD_INTCB*n*R corresponding to the CSIB*n* reception completion interrupt INTCB*n*R.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSIBn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB*n*_ErrorCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the CSIB*n* reception error interrupt INTCB*n*R (overrun error).

Remark This API function is called as the callback routine of interrupt process MD_INTCB*n*R corresponding to the CSIB*n* reception error interrupt INTCB*n*R.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    CSIBn_ErrorCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE*n*_Init

[E/Sx3-H]

Performs initialization necessary to control the 3-wire variable-length serial I/O (CSIE) functions.

[Classification]

CG_serial.c

[Syntax]

```
void    CSIEn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE*n*_UserInit

[E/Sx3-H]

Performs user-defined initialization relating to the 3-wire variable-length serial I/O (CSIE).

Remark This API function is called as the CSIEn_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSIEn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE*n*_Start

[E/Sx3-H]

Enables 3-wire variable-length serial I/O (CSIE).

[Classification]

CG_serial.c

[Syntax]

```
void    CSIEn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE*n*_Stop

[E/Sx3-H]

Disables 3-wire variable-length serial I/O (CSIE).

[Classification]

CG_serial.c

[Syntax]

```
void    CSIEn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE*n*_SendData

[E/Sx3-H]

Starts CSIE*n* data transmission.

Remarks 1. This API function repeats the byte-level CSIE*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. When performing a CSIE*n* transmission, CSIEn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    “CG_macrodriver.h“
MD_STATUS   CSIEn_SendData ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR *txbuf;`	Pointer to a buffer storing the transmission data
I	`USHORT txnum;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CSIE*n*_ReceiveData

[E/Sx3-H]

Starts CSIE*n* data reception.

Remarks 1. This API function performs byte-level CSIE*n* reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. When performing a CSIE*n* reception, CSIEn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIEn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CSIE*n*_SendReceiveData

[E/Sx3-H]

Starts CSIE*n* data transmission/reception.

Remarks 1. This API function repeats the byte-level CSIE*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. This API function performs byte-level CSIE*n* reception the number of times specified by the parameter *txnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

3. When performing a CSIE*n* transmission/reception, CSIEn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIEn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *txbuf;`	Pointer to a buffer to store the received data
I	`USHORT txnum;`	Total amount of data to receive
I	`UCHAR *rxbuf;`	Pointer to a buffer storing the transmission data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CSIE*n*_SendEndCallback

[E/Sx3-H]

Performs processing in response to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.

Remark This API function is called as the callback routine (process performed when the total number of CSIE*n* transmissions specified in the parameter *txnum* for CSIEn_SendData or CSIEn_SendReceiveData has been completed) of interrupt process MD_INTCE*n*T corresponding to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSIEn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE*n*_ReceiveEndCallback

[E/Sx3-H]

Performs processing in response to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.

Remark This API function is called as the callback routine (process performed when the total number of CSIE*n* receptions specified in the parameter *rxnum* for CSIEn_ReceiveData or CSIEn_SendReceiveData has been completed) of interrupt process MD_INTCE*n*T corresponding to the CSIE*n* transmission/reception completion interrupt INTCE*n*T.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSIEn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE*n*_ErrorCallback

[E/Sx3-H]

Performs processing in response to the CSIE*n*BUF overflow interrupt INTCE*n*TIOF.

Remark This API function is called as the callback routine of interrupt process MD_INTCE*n*TIOF corresponding to the CSIE*n*BUF overflow interrupt INTCE*n*TIOF.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    CSIEn_ErrorCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control the IIC bus functions.

[Classification]

CG_serial.c

[Syntax]

```
void    IIC0n_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the IIC bus.

Remark This API function is called as the IIC0n_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IIC0n_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Ends IIC0*n* communication.

[Classification]

CG_serial.c

[Syntax]

```
void    IIC0n_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_StopCondition

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Generates a stop condition.

[Classification]

CG_serial.c

[Syntax]

```
void    IIC0n_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_MasterSendStart

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts IIC0*n* master transmission.

Remark This API function repeats the byte-level IIC0*n* master transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

CG_serial.c

[Syntax]

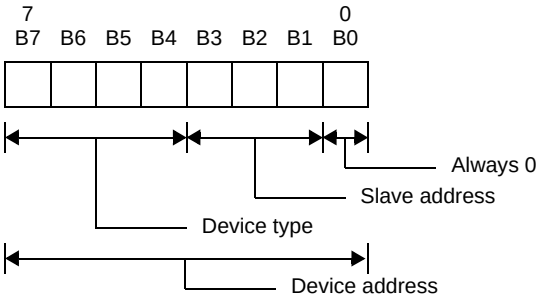
```
#include    “CG_macrodriver.h“
MD_STATUS   IIC0n_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *adr*;	Device address
I	UCHAR **txbuf*;	Pointer to a buffer storing the transmission data
I	USHORT *txnum*;	Total amount of data to send
I	UCHAR *wait*;	Setup time of start conditions

Remark Device address *adr* consists of a device type and slave address.



[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

IIC0*n*_MasterReceiveStart

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts IIC0*n* master reception.

Remark This API function performs byte-level simple IIC0*n* master reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

[Classification]

CG_serial.c

[Syntax]

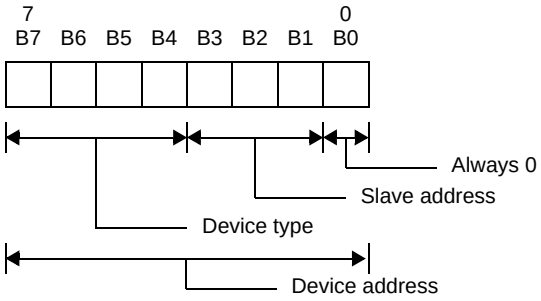
```
#include    "CG_macrodriver.h"
MD_STATUS   IIC0n_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *adr*;	Device address
O	UCHAR **rxbuf*;	Pointer to a buffer to store the received data
I	USHORT *rxnum*;	Total amount of data to receive
I	UCHAR *wait*;	Setup time of start conditions

Remark Device address *adr* consists of a device type and slave address.



[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

IIC0*n*_MasterSendEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the IIC0*n* master transfer completion interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the simple IIC0*n* master transfer completion interrupt INTIIC*n* (performed when total number of simple IIC0*n* master transmissions specified by IIC0n_MasterSendStart parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    IIC0n_MasterSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_MasterReceiveEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the IIC0*n* master transfer completion interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the simple IIC0*n* master transfer completion interrupt INTIIC*n* (performed when total number of simple IIC0*n* master transmissions specified by IIC0n_MasterReceiveStart parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    IIC0n_MasterReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_MasterErrorCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to detection of error in IIC0*n* master communication.

Remark This API function is called as the callback routine (process carried out when an IIC0*n* master communication error is detected) of interrupt process MD_INTIIC*n* corresponding to the IIC0*n* master transfer complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IIC0n_MasterErrorCallback ( MD_STATUS flag );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`MD_STATUS flag;`	Cause of IIC0*n* master communication error MD_SPT: Stop condition detected MD_NACK: Acknowledge detected

[Return value]

None.

IIC0*n*_SlaveSendStart

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts IIC0*n* slave transmission.

Remark This API function repeats the byte-level IIC0*n* slave transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

CG_serial.c

[Syntax]

```
#include    “CG_macrodriver.h“
void    IIC0n_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR *txbuf;`	Pointer to a buffer storing the transmission data
I	`USHORT txnum;`	Total amount of data to send

[Return value]

None.

IIC0*n*_SlaveReceiveStart

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts IIC0*n* slave reception.

Remark This API function performs byte-level IIC0*n* slave reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IIC0n_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

None.

IIC0*n*_SlaveSendEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the IIC0*n* slave transfer completion interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the simple IIC0*n* slave transfer completion interrupt INTIIC*n* (performed when total number of simple IIC0*n* slave transmissions specified by IIC0n_SlaveSendStart parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    IIC0n_SlaveSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_SlaveReceiveEndCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to the IIC0*n* slave transfer completion interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the simple IIC0*n* slave transfer completion interrupt INTIIC*n* (performed when total number of simple IIC0*n* slave transmissions specified by IIC0n_SlaveReceiveStart parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    IIC0n_SlaveReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_SlaveErrorCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to detection of error in IIC0*n* slave communication.

Remark This API function is called as the callback routine (process carried out when an IIC0*n* slave communication error is detected) of interrupt process MD_INTIIC*n* corresponding to the IIC0*n* slave transfer complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IIC0n_SlaveErrorCallback ( MD_STATUS flag );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`MD_STATUS flag;`	Cause of IIC0*n* master communication error MD_ERROR: Address mismatch deteced MD_NACK: Acknowledge detected

[Return value]

None.

IIC0*n*_GetStopConditionCallback

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs processing in response to detection of stop condition.

Remark This API function is called as the callback routine (process carried out when a stop condition is detected) of interrupt process MD_INTIIC*n* corresponding to the IIC0*n* transfer completion interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IIC0n_GetStopConditionCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.3.6 A/D

Below is a list of API functions output by Applilet3 for D/A converter use.

Table 3-7. API Functions: [A/D]

API Function Name	Function
AD_Init	Performs initialization necessary to control A/D converter functions.
AD_UserInit	Performs user-defined initialization relating to the A/D converter.
AD_Start	Starts A/D conversion.
AD_Stop	Ends A/D conversion.
AD_SelectADChannel	Configures the analog voltage input pin for A/D conversion.
AD_SetPFTCondition	Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger).
AD_Read	Reads the results of A/D conversion (10 bits).
AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

AD_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control A/D converter functions.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the A/D converter.

Remark This API function is called as the AD_Init callback routine.

[Classification]

CG_ad_user.c

[Syntax]

```
void    AD_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts A/D conversion.

Remark A/D conversion is performed repeatedly between the call to this API function and a call to AD_Stop.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows A/D conversion of analog voltage.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D conversion complete flag */
void main ( void ) {
    USHORT  buffer = 0;
    int     wait = 100;
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    ......
    AD_Start ();                        /* Start A/D conversion */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_Stop ();                         /* End A/D conversion */
    ......
}
```

[CG_ad_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* A/D conversion complete flag */
__interrupt void MD_INTAD ( void ) {    /* Interrupt processing for INTAD */
    gFlag = 0;                          /* Set A/D conversion complete flag */
}
```

AD_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Ends A/D conversion.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_SelectADChannel

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Configures the analog voltage input pin for A/D conversion.

[Classification]

CG_ad.c

[Syntax]

```
#include    "CG_ad.h"
MD_STATUS   AD_SelectADChannel ( enum ADChannel channel );
```

[Argument(s)]

I/O	Argument	Description
I	`enum ADChannel channel;`	Analog voltage input pin ADCHANNEL*n*: Input pin

Remark See the header file CG_ad.h for details about the analog voltage input pin ADCHANNEL*n*.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

AD_SetPFTCondition

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger).

[Classification]

CG_ad.c

[Syntax]

```
#include    "CG_ad.h"
MD_STATUS   AD_SetPFTCondition ( UCHAR pftvalue, enum ADPFTMode mode );
```

[Argument(s)]

I/O	Argument	Description
I	`UCHAR pftvalue;`	Comparison value
I	`enum ADPFTMode mode;`	Cause of A/D conversion end interrupt INTAD EACHEND: Generate INTAD when A/D is complete PFTHIGHER: Generate INTAD if ADA0CRnH ≥ ADA0PFT PFTLOWER: Generate INTAD if ADA0CRnH < ADA0PFT

Remark If the parameter *mode* is set to PFTHIGHER or PFTLOWER, then the value set in parameter *pftvalue* is set in the power-fail compare threshold value register (ADA0PFT), and used for comparison with the A/D conversion result register*n*H (ADA0CR*n*H).

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

AD_Read

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the results of A/D conversion (10 bits).

[Classification]

CG_ad.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   AD_Read ( USHORT *buffer );
```

[Argument(s)]

I/O	Argument	Description
O	USHORT *buffer;	Pointer to area in which to store read results of A/D conversion (10 bits)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend)

AD_ReadByte

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

[Classification]

CG_ad.c

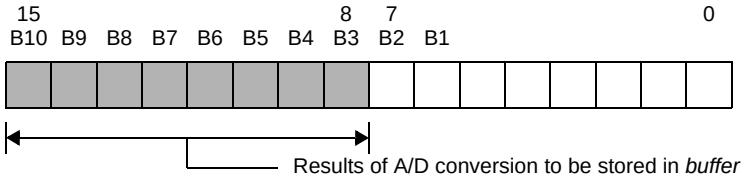
[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   AD_ReadByte ( UCHAR *buffer );
```

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *buffer;`	Pointer to area in which to store the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution)

Remark Below is an example of the results of A/D conversion to be stored in *buffer*.



[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend)

3.3.7 D/A

Below is a list of API functions output by Applilet3 for D/A converter use.

Table 3-8. API Functions: [D/A]

API Function Name	Function
DAn_Init	Performs initialization necessary to control D/A converter functions.
DAn_UserInit	Performs user-defined initialization relating to the D/A converter.
DAn_Start	Starts D/A conversion.
DAn_Stop	Ends D/A conversion.
DAn_SetValue	Sets the analog voltage output to the ANO*n* pin.

DA*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control D/A converter functions.

[Classification]

CG_da.c

[Syntax]

```
void    DAn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the D/A converter.

Remark This API function is called as the DAn_Init callback routine.

[Classification]

CG_da_user.c

[Syntax]

```
void    DAn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts D/A conversion.

[Classification]

CG_da.c

[Syntax]

```
void    DAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Ends D/A conversion.

[Classification]

CG_da.c

[Syntax]

```
void    DAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_SetValue

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Sets the analog voltage output to the ANO*n* pin.

[Classification]

CG_da.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    DAn_SetValue ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *value*;	Analog voltage (0x0 to 0xff)

[Return value]

None.

[Example]

Below is an example of setting "analog voltage" to channels 0 and 1

[CG_main.c]

```
void main ( void ) {
    ......
    DA0_Start ();                       /* Start D/A conversion */
    DA1_Start ();                       /* Start D/A conversion */
    ......
    DA0_SetValue ( 0x7f );              /* Set analog voltage */
    ......
}
```

[CG_tau_user.c]

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* Interrupt processing for INTTM05 */
    DA1_SetValue ( gValue++ );          /* Set analog voltage */
}
```

3.3.8 Timer

Below is a list of API functions output by Applilet3 for timer use.

Table 3-9. API Functions: [Timer]

API Function Name	Function
TMPn_Init	Performs initialization necessary to control timer "TMP*n*" functions.
TMPn_UserInit	Performs user-defined initialization relating to the timer "TMP*n*".
TMPn_Start	Starts the count for timer "TMP*n*".
TMPn_Stop	Ends the count for timer "TMP*n*".
TMPn_ChangeTimerCondition	Changes the counter value for timer "TMP*n*".
TMPn_GetPulseWidth	Reads the pulse width of timer TMPn (high/low level width).
TMPn_GetFreeRunningValue	Reads the value captured by timer TMP*n*.
TMPn_ChangeDuty	Changes the duty ratio of the PWM signal.
TMPn_SoftwareTriggerOn	Generates the trigger (software trigger) for timer output.
TMQ0_Init	Performs initialization necessary to control timer "TMQ0" functions.
TMQ0_UserInit	Performs user-defined initialization relating to the timer "TMQ0".
TMQ0_Start	Starts the count for timer "TMQ0".
TMQ0_Stop	Ends the count for timer "TMQ0".
TMQ0_ChangeTimerCondition	Changes the counter value for timer "TMQ0".
TMQ0_GetPulseWidth	Reads the pulse width of timer TMQ0 (high/low level width).
TMQ0_GetFreeRunningValue	Reads the value captured by timer TMQ0.
TMQ0_ChangeDuty	Changes the duty ratio of the PWM signal.
TMQ0_SoftwareTriggerOn	Generates the trigger (software trigger) for timer output.
TMMn_Init	Performs initialization necessary to control timer "TMM*n*" functions.
TMMn_UserInit	Performs user-defined initialization relating to the timer "TMM*n*".
TMMn_Start	Starts the count for timer "TMM*n*".
TMMn_Stop	Ends the count for timer "TMM*n*".
TMMn_ChangeTimerCondition	Changes the counter value for timer "TMM*n*".

TMP*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control timer "TMP*n*" functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMP*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the timer "TMP*n*".

Remark This API function is called as the TMPn_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TMPn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMP*n*_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts the count for timer "TMP*n*".

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMP*n*_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Ends the count for timer "TMP*n*".

Remark The length of time between the call to this API function and the end of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMP*n*_ChangeTimerCondition

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Changes the counter value for timer "TMP*n*".

Remark The value specified in parameter *arrar_reg* is set in TMP*n* capture/compare register *m* (TP*n*CCR*m*).

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   TMPn_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num )
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`USHORT *array_reg;`	Pointer to an area storing the count value (0x0 to 0xffff)
I	`UCHAR array_num;`	Register to change 1: TP*n*CCR0 2: TP*n*CCR0, TP*n*CCR1

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

The example below shows changing the interval time to one half.
In this example, channel 0 has been selected for the interval timer.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    int     flag_finish = 1;
    USHORT  array_reg = TMP_TP0CCR0_VALUE >> 1;            /* TMP_TP0CCR0_VALUE: Current
interval time */
    UCHAR   array_num = 1;
    ......
    TMP0_Start ();                                         /* Start count */
    while ( flag_finish );                                 /* Check for time up */
    ......
    TMP0_ChanneTimerCondition ( &array_reg, array_num );    /* Change counter value */
```

```
    ......
}
```

TMP*n*_GetPulseWidth

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the pulse width of timer TMP*n* (high/low level width).

Remarks 1. This API function can only be called when TMP*n* is being used for pulse width measurement.

2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TMPn_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`ULONG *activewith;`	Pointer to an area storing the high level width that was read (0x0 to 0x1ffff)
O	`ULONG *inactivewidth;`	Pointer to an area storing the low level width that was read (0x0 to 0x1ffff)

[Return value]

None.

TMP*n*_GetFreeRunningValue

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the value captured by timer TMP*n*.

Remark This API function can only be called when TMP*n* is used as a free-running timer, and TMP*n*CCR*m* capture/compare register m (TMP*n*CCR*m*) is selected as the capture register.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMPn_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`ULONG *count;`	Pointer to area in which to store the width that was read
i	`enum TMChannel channel;`	Channel to read TMCHANNEL0: Channel 0 (TP*n*CCR0) TMCHANNEL1: Channel 1 (TP*n*CCR1)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

TMP*n*_ChangeDuty

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Changes the duty ratio of the PWM signal.

Remark This API function can only be called when TMP*n* is used for external trigger pulse output / PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TMPn_ChangeDuty ( UCHAR array_duty );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *array_duty*;	Duty ratio (0 to 100, unit: %)

Remark The value set to duty ratio *array_duty* must be in base 10 notation.

[Return value]

None.

[Example]

The example below shows changing the duty ratio to 25%.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    int     flagStatus = 1;
    UCHAR   array_duty = 25;
    ......
    TMP0_Start ();                      /* Start count */
    while ( flagStatus );
    TMP0_ChangeDuty ( array_duty );     /* Change duty ratio */
    ......
}
```

TMP*n*_SoftwareTriggerOn

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Generates the trigger (software trigger) for timer output.

Remark This API function can only be called when TMP*n* is used for external trigger pulse output / one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_SoftwareTriggerOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMQ0_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control timer "TMQ0" functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the timer "TMQ0".

Remark This API function is called as the TMQ0_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TMQ0_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts the count for timer "TMQ0".

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Ends the count for timer "TMQ0".

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_ChangeTimerCondition

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Changes the counter value for timer "TMQ0".

Remark The value specified in parameter *arrar_reg* is set to TMQ0 capture/compare register *m* "TQ0CCR*m*".

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   TMQ0_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num )
```

[Argument(s)]

I/O	Argument	Description
I	`USHORT *array_reg;`	Pointer to an area storing the count value (0x0 to 0xffff)
I	`UCHAR array_num;`	Register to change 1: TQ0CCR0 2: TQ0CCR0, TQ0CCR1 3: TQ0CCR0, TQ0CCR1, TQ0CCR2 4: TQ0CCR0, TQ0CCR1, TQ0CCR2, TQ0CCR3

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

The example below shows changing the interval time to one half.
In this example, channel 0 has been selected for the interval timer.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    int     flag_finish = 1;
    USHORT  array_reg = TMQ_TQ0CCR0_VALUE >> 1;            /* TMQ_TQ0CCR0_VALUE: Current
interval time */
    UCHAR   array_num = 1;
    ......
    TMQ0_Start ();                                         /* Start count */
    while ( flag_finish );                                 /* Check for time up */
    ......
    TMQ0_ChanneTimerCondition ( &array_reg, array_num );    /* Change counter value */
```

```
    ......
}
```

TMQ0_GetPulseWidth

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the pulse width of timer TMQ0 (high/low level width).

Remarks 1. This API function can only be called when TMQ0 is being used for pulse width measurement.

2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TMQ0_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

[Argument(s)]

I/O	Argument	Description
O	`ULONG *activewith;`	Pointer to an area storing the high level width that was read (0x0 to 0x1ffff)
O	`ULONG *inactivewidth;`	Pointer to an area storing the low level width that was read (0x0 to 0x1ffff)

[Return value]

None.

TMQ0_GetFreeRunningValue

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the value captured by timer TMQ0.

Remark This API function can only be called when TMQ0 is used as a free-running timer, and TQ0CCR*m* capture/compare register m (TQ0CCR*m*) is selected as the capture register.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMQ0_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

[Argument(s)]

I/O	Argument	Description
O	`ULONG *count;`	Pointer to area in which to store the width that was read
i	`enum TMChannel channel;`	Channel to read TMCHANNEL0: Channel 0 (TQ0CCR0) TMCHANNEL1: Channel 1 (TQ0CCR1) TMCHANNEL2: Channel 2 (TQ0CCR2) TMCHANNEL3: Channel 3 (TQ0CCR3)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

TMQ0_ChangeDuty

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Changes the duty ratio of the PWM signal.

Remark This API function can only be called when TMQ0 is used for external trigger pulse output / PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   TMQ0_ChangeDuty ( UCHAR *array_duty, UCHAR array_num );
```

[Argument(s)]

I/O	Argument	Description
I	UCHAR *array_duty;	Pointer to an area storing the duty ratio (0 to 100; in percent)
I	UCHAR array_num;	Register to change 1: TQ0CCR1 2: TQ0CCR1, TQ0CCR2 3: TQ0CCR1, TQ0CCR2, TQ0CCR3

Remark The value set to duty ratio *array_duty* must be in base 10 notation.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

The example below shows changing the duty ratio to 25%.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    int     flagStatus = 1;
    UCHAR   array_duty = 25;
    UCHAR   array_num = 1;
    ......
    TMQ0_Start ();                                  /* Start count */
    while ( flagStatus );
    TMQ0_ChangeDuty ( &array_duty, array_num );     /*Change duty ratio */
```

```
    ......
}
```

TMQ0_SoftwareTriggerOn

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Generates the trigger (software trigger) for timer output.

Remark This API function can only be called when TMQ0 is used for external trigger pulse output / one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_SoftwareTriggerOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMM*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control timer “TMM*n*“ functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMMn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMM*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the timer “TMM*n*“.

Remark This API function is called as the TMMn_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TMMn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMM*n*_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts the count for timer “TMM*n*”.

[Classification]

CG_timer.c

[Syntax]

```
void    TMMn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMM*n*_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Ends the count for timer “TMM*n*”.

[Classification]

CG_timer.c

[Syntax]

```
void    TMMn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMM*n*_ChangeTimerCondition

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Changes the counter value for timer “TMM*n*”.

Remark The value specified in parameter *regvalue* is set to TMM*n* control register 0”TM*n*CMP0”.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TMMn_ChangeTimerCondition ( USHORT regvalue )
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`USHORT regvalue;`	Counter value (0x0 to 0xffff)

[Return value]

None.

3.3.9 Watch Timer

Below is a list of API functions output by Applilet3 for watch timer use.

Table 3-10. API Functions: [Watch Timer]

API Function Name	Function
WT_Init	Performs initialization necessary to control watch timer functions.
WT_UserInit	Performs user-defined initialization relating to the watch timer.
WT_Start	Clears the watch timer counter and resumes counting.
WT_Stop	Ends the count for watch timer.

WT_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control watch timer functions.

[Classification]

CG_wt.c

[Syntax]

```
void    WT_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

WT_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the watch timer.

Remark This API function is called as the WT_Init callback routine.

[Classification]

CG_wt_user.c

[Syntax]

```
void    WT_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

WT_Start

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Clears the watch timer counter and resumes counting.

[Classification]

CG_wt.c

[Syntax]

```
void    WT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

WT_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Ends the count for watch timer.

[Classification]

CG_wt.c

[Syntax]

```
void    WT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below illustrates use of the watch timer function.

[CG_main.c]

```
#include    "CG_macrodriver.h"
ULONG   INT_flg = 0;
void main ( void ) {
    WT_Start ();            /* Start count */
    while ( !INT_flg );
    WT_Stop ();             /* End count */
    ......
}
```

[CG_wt_user.c]

```
#include    "CG_macrodriver.h"
extern  ULONG   INT_flg;
__interrupt void MD_INTWT ( void ) {    /* Interrupt processing for INTWT */
    INT_flg = 1;
}
```

3.3.10 RTC

Below is a list of API functions output by Applilet3 for real-time counter use.

Table 3-11. API Functions: [RTC]

API Function Name	Function
RTC_Init	Performs initialization necessary to control real-time counter functions.
RTC_UserInit	Performs user-defined initialization relating to the real-time counter.
RTC_CounterEnable	Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).
RTC_CounterDisable	Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).
RTC_SetHourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time counter.
RTC_CounterSet	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
RTC_CounterGet	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
RTC_ConstPeriodInterruptEnable	Sets the cycle of the interrupts INTRTC0, then starts the cyclic interrupt function.
RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC0.
RTC_AlarmEnable	Starts the alarm interrupt function.
RTC_AlarmDisable	Ends the alarm interrupt function.
RTC_AlarmSet	Sets the alarm conditions (weekday, hour, minute).
RTC_AlarmGet	Reads the alarm conditions (weekday, hour, minute).
RTC_AlarmInterruptCallback	Performs processing in response to the alarm interrupt INTRTC1.
RTC_IntervalStart	Starts the interval interrupt function.
RTC_IntervalStop	Ends the interval interrupt function.
RTC_IntervalInterruptEnable	Sets the cycle of the interrupts INTRTC2, then starts the interval interrupt function.
RTC_IntervalInterruptDisable	Ends the interval interrupt function.
RTC_RC1CK1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RC1CK1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RC1CKO_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RC1CKO_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RC1CKDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
RTC_RC1CKDIV_OutputDisable	Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
RTC_ChangeCorrectionValue	Changes the timing and correction value for correcting clock errors.

RTC_Init

[E/Sx3-H]

Performs initialization necessary to control real-time counter functions.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_UserInit

[E/Sx3-H]

Performs user-defined initialization relating to the real-time counter.

Remark This API function is called as the RTC_Init callback routine.

[Classification]

CG_rtc_user.c

[Syntax]

```
void    RTC_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_CounterEnable

[E/Sx3-H]

Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_CounterEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_CounterDisable

[E/Sx3-H]

Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_CounterDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_SetHourSystem

[E/Sx3-H]

Sets the clock type (12-hour or 24-hour clock) of the real-time counter.

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_SetHourSystem ( enum RTCHourSystem hoursystem );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCHourSystem hoursystem;`	Clock type HOUR12: 12-hour clock HOUR24: 24-hour clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)
MD_ARGERROR	Invalid argument specification

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "CG_rtc.h" larger.

[Example]

Below is an example of setting the clock type to the 24-hour clock.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_CounterEnable ();           /* Start count */
    ......
    RTC_SetHourSystem ( HOUR24 );   /* Set clock type */
    ......
}
```

RTC_CounterSet

[E/Sx3-H]

Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterSet ( struct RTCCounterValue counterwriteval );
```

[Argument(s)]

I/O	Argument	Description
I	`struct RTCCounterValue counterwriteval;`	Counter value

Remark Below is an example of the structure RTCCounterValue (counter value) for the real-time counter.

```
struct  RTCCounterValue {
    UCHAR   Sec;    /* second */
    UCHAR   Min;    /* Minute */
    UCHAR   Hour;   /* Hour */
    UCHAR   Day;    /* Day */
    UCHAR   Week;   /* Weekday (0: Sunday, 6: Saturday) */
    UCHAR   Month;  /* Month */
    UCHAR   Year;   /* Year */
};
```

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "CG_rtc.h" larger.

[Example]

The example below shows the counter value of the real-time counter being set to "2008/12/25 (Thu.) 17:30:00".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( main ) {
    struct  RTCCounterValue counterwriteval;
    ......
    RTC_CounterEnable ();               /* Start count */
    ......
    counterwriteval.Year = 0x08;
    counterwriteval.Month = 0x12;
    counterwriteval.Day = 0x25;
    counterwriteval.Week = 0x05;
    counterwriteval.Hour = 0x17;
    counterwriteval.Min = 0x30;
    counterwriteval.Sec = 0;
    RTC_SetHourSystem ( HOUR24 );       /* Set clock type */
    RTC_CounterSet ( counterwriteval ); /* Set counter value */
    ......
}
```

RTC_CounterGet

[E/Sx3-H]

Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterGet ( struct RTCCounterValue *counterreadval );
```

[Argument(s)]

I/O	Argument	Description
O	`struct RTCCounterValue *counterreadval;`	Pointer to structure in which to store the counter value being read

Remark See RTC_CounterSet for details about the RTCCounterValue counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before reading)
MD_BUSY2	Stopping count process (after reading)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "CG_rtc.h" larger.

[Example]

Below is an example of reading the counter value of the real-time counter.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCCounterValue counterreadval;
    ......
    RTC_CounterEnable ();               /* Start count */
    ......
    RTC_CounterGet ( &counterreadval ); /* Read count value */
    ......
}
```

RTC_ConstPeriodInterruptEnable

[E/Sx3-H]

Sets the cycle of the interrupts INTRTC0, then starts the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_rtc.h"
MD_STATUS   RTC_ConstPeriodInterruptEnable ( enum RTCINTPeriod period );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCINTPeriod period;`	Interrupt INTRTC0 cycle HALFSEC: 0.5 seconds ONESEC: 1 second ONEMIN: 1 minute ONEHOUR: 1 hour ONEDAY: 1 day ONEMONTH: 1 month

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of setting the cycle of the interrupts INTRTC0, then starting the cyclic interrupt function.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_ConstPeriodInterruptDisable ();         /* End of cyclic interrupt function */
    ......
    RTC_ConstPeriodInterruptEnable ( HALFSEC ); /* Start of cyclic interrupt function */
    ......
}
```

RTC_ConstPeriodInterruptDisable

[E/Sx3-H]

Ends the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_ConstPeriodInterruptDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_ConstPeriodInterruptCallback

[E/Sx3-H]

Performs processing in response to the cyclic interrupt INTRTC0.

Remark This API function is called as the callback routine of interrupt process MD_INTRTC0 corresponding to the cyclic interrupt INTRTC0.

[Classification]

CG_rtc_user.c

[Syntax]

```
void    RTC_ConstPeriodInterruptCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmEnable

[E/Sx3-H]

Starts the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_AlarmEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmDisable

[E/Sx3-H]

Ends the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_AlarmDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmSet

[E/Sx3-H]

Sets the alarm conditions (weekday, hour, minute).

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_rtc.h"
void    RTC_AlarmSet ( struct RTCAlarmValue alarmval );
```

[Argument(s)]

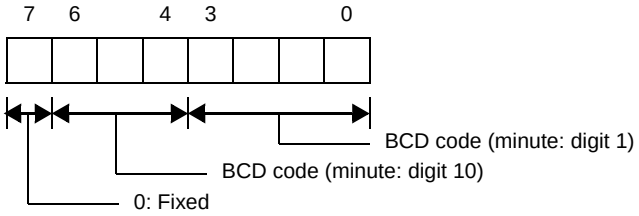
I/O	Argument	Description
I	`struct RTCAlarmValue alarmval;`	Alarm conditions (weekday, hour, minute)

Remark Below is shown the structure RTCAlarmValue (alarm conditions).

```
struct  RTCAlarmValue {
    UCHAR   Alarmwm;    /* Minute */
    UCHAR   Alarmwh;    /* Hour */
    UCHAR   Alarmww;    /* Weekday */
};
```

- Alarmwm (Minute)

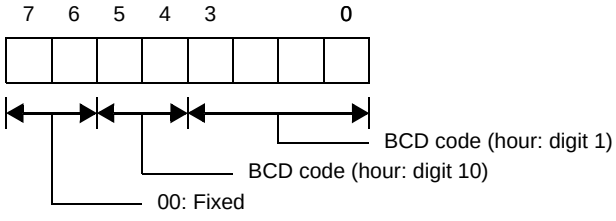
Below are shown the meanings of each bit of the structure member Alarmwm.



- Alarmwh (Hour)

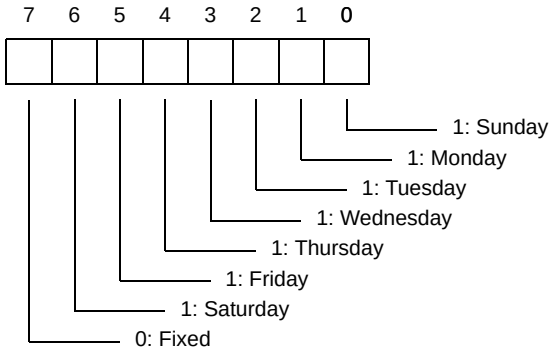
Below are shown the meanings of each bit of the structure member Alarmwh. If the real-time counter is set to the 12-hour clock, then bit 5 has the following meaning.

0: AM
1: PM



- Alarmww (Weekday)

  Below are shown the meanings of each bit of the structure member Alarmww.



[Return value]

None.

[Example 1]

The example below shows the alarm conditions being set to "Monday/Tuesday/Wednesday at 17:30".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();             /* Start alarm interrupt function */
    RTC_CounterEnable ();           /* Start count */
    ......
    RTC_SetHourSystem ( HOUR24 );   /* Set clock type */
    alarmval.Alarmww = 0xe;
    alarmval.Alarmwh = 0x17;
    alarmval.Alarmwm = 0x30;
    RTC_AlarmSet ( alarmval );      /* Set conditions */
    ......
}
```

[Example 2]

The example below shows the alarm conditions being set to "Saturday/Sunday (time left unchanged)".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();             /* Start alarm interrupt function */
    ......
    alarmval.Alarmww = 0x41;
```

```
    RTC_AlarmSet ( alarmval );      /* Change conditions */
    ......
}
```

RTC_AlarmGet

[E/Sx3-H]

Reads the alarm conditions (weekday, hour, minute).

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_rtc.h"
void    RTC_AlarmGet ( struct RTCAlarmValue *alarmval );
```

Remark See RTC_AlarmSet for details about RTCAlarmValue (alarm conditions).

[Argument(s)]

I/O	Argument	Description
O	`struct RTCAlarmValue *alarmval;`	Pointer to structure in which to store the conditions being read

[Return value]

None.

[Example]

The example below shows the alarm conditions being read.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();         /* Start alarm interrupt function */
    ......
    RTC_AlarmGet ( &alarmval ); /* Read conditions */
    ......
}
```

RTC_AlarmInterruptCallback

[E/Sx3-H]

Performs processing in response to the alarm interrupt INTRTC1.

Remark This API function is called as the callback routine of interrupt process MD_INTRTC1 corresponding to the alarm interrupt INTRTC1.

[Classification]

CG_rtc_user.c

[Syntax]

```
void    RTC_AlarmInterruptCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalStart

[E/Sx3-H]

Starts the interval interrupt function.

Remark After setting the cycle of the interrupts INTRTC2, call RTC_IntervalInterruptEnable to start the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_IntervalStart ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalStop

[E/Sx3-H]

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_IntervalStop ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalInterruptEnable

[E/Sx3-H]

Sets the cycle of the interrupts INTRTC2, then starts the interval interrupt function.

Remark Call RTC_IntervalStart to start the interval interrupt function without setting the cycle of the interrupts INTRTC2.

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_rtc.h"
MD_STATUS   RTC_IntervalInterruptEnable ( enum RTCINTInterval interval );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCINTInterval interval;`	Interrupt INTRTC2 cycle INTERVAL0: 2^6/fRTC INTERVAL1: 2^7/fRTC INTERVAL2: 2^8/fRTC INTERVAL3: 2^9/fRTC INTERVAL4: 2^10/fRTC INTERVAL5: 2^11/fRTC INTERVAL6: 2^12/fRTC

Remark fSUB is the frequency of the subsystem clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of changing the interval, the restarting the interval interrupt function.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_IntervalStart ();                    /* Start interval interrupt function */
    ......
    RTC_IntervalStop ();                     /* End interval interrupt function */
    ......
```

```
    RTC_IntervalInterruptEnable ( INTERVAL6 );  /* Start interval interrupt function */
    ......
}
```

RTC_IntervalInterruptDisable

[E/Sx3-H]

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_IntervalInterruptDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CK1HZ_OutputEnable

[E/Sx3-H]

Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CK1HZ_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CK1HZ_OutputDisable

[E/Sx3-H]

Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CK1HZ_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKO_OutputEnable

[E/Sx3-H]

Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKO_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKO_OutputDisable

[E/Sx3-H]

Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKO_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKDIV_OutputEnable

[E/Sx3-H]

Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKDIV_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKDIV_OutputDisable

[E/Sx3-H]

Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKDIV_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_ChangeCorrectionValue

[E/Sx3-H]

Changes the timing and correction value for correcting clock errors.

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_ChangeCorrectionValue ( enum RTCCorectionTiming timing, UCHAR corectval );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCCorectionTiming timing;`	When clock errors are corrected EVERY20S: When the seconds digits are 00, 20 or 40 EVERY60S: When the seconds digits are 00
I	`UCHAR corectval;`	Clock error correction value

Remark This API function does not correct clock errors if correction value *corectVal* is set to 0x0, 0x1, 0x40 or 0x41.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

3.3.11 Real-Time Output

Below is a list of API functions output by Applilet3 as the real-time output function.

Table 3-12. API Functions: [Real-Time Output]

API Function Name	Function
RTOn_Init	Performs initialization necessary to control real-time output functions.
RTOn_UserInit	Performs user-defined initialization relating to the real-time output.
RTOn_Enable	Enables (validates) real-time output.
RTOn_Disable	Disables (invalidates) real-time output.
RTOn_Set2BitData	Sets 2-bit data for real-time output.
RTOn_Set4BitData	Sets 4-bit data for real-time output.
RTOn_Set6BitData	Sets 6-bit data for real-time output.
RTOn_GetValue	Reads data from real-time output.

RTO*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control real-time output functions.

[Classification]

CG_rto.c

[Syntax]

```
void    RTOn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the real-time output.

Remark This API function is called as the RTOn_Init callback routine.

[Classification]

CG_rto_user.c

[Syntax]

```
void    RTOn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO*n*_Enable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Enables (validates) real-time output.

[Classification]

CG_rto.c

[Syntax]

```
void    RTOn_Enable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO*n*_Disable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Disables (invalidates) real-time output.

[Classification]

CG_rto.c

[Syntax]

```
void    RTOn_CounterDisable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO*n*_Set2BitData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Sets 2-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

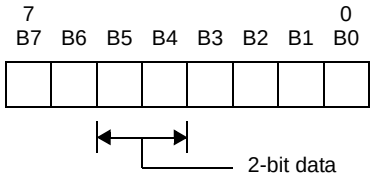
```
#include    "CG_macrodriver.h"
void    RTOn_Set2BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *data*;	2-bit data

Remark The API functions treat values set in bits 4 to 5 as 2-bit data.



[Return value]

None.

RTO*n*_Set4BitData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Sets 4-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

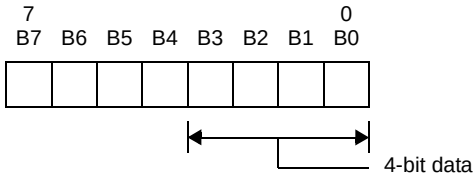
```
#include    "CG_macrodriver.h"
void    RTOn_Set4BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *data*;	4-bit data

Remark The API functions treat values set in bits 0 to 3 as 4-bit data.



[Return value]

None.

RTO*n*_Set6BitData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Sets 6-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

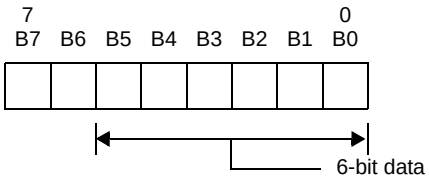
```
#include    "CG_macrodriver.h"
void    RTOn_Set6BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *data*;	6-bit data

Remark The API functions treat values set in bits 0 to 5 as 6-bit data.



[Return value]

None.

RTO*n*_GetValue

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads data from real-time output.

[Classification]

CG_rto.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    RTOn_GetValue ( UCHAR *value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	UCHAR *value;	Pointer to area in which to store the value that was read

[Return value]

None.

[Example]

Below is an example of reading the counter value of the real-time counter.

[CG_main.c]

```
#include    "CG_rto.h"
void main ( void ) {
    RTO0_Set2BitData ( 0x30 );          /* Set output data */
    RTO0_Enable ();                     /* Enable real-time output */
    ......
    RTO0_Disable ();                    /* Disable real-time output */
    ......
}
```

[CG_timer_user.c]

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTTP4CC0 ( void ) {  /* Interrupt processing for INTTP4CC0 interrupt */
    UCHAR   value = 0;
    RTO0_GetValue ( &value );             /* Read output data */
    value = ~value;
    RTO0_Set2BitData ( value );           /* Set output data */
}
```

3.3.12 DMA

Below is a list of API functions output by Applilet3 for DMA (Direct Memory Access) controller use.

Table 3-13. API Functions: [DMA]

API Function Name	Function
DMAn_Init	Performs initialization necessary to control DMA controller functions.
DMAn_UserInit	Performs user-defined initialization relating to the DMA controller.
DMAn_Enable	Enables operation of channel *n*.
DMAn_Disable	Disables operation of channel *n*.
DMAn_CheckStatus	Reads the transfer status (transfer complete/transfer ongoing).
DMAn_SetData	Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.
DMAn_SoftwareTriggerOn	Uses a software trigger as a DMA transfer start trigger.

DMA*n*_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control DMA controller functions.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the DMA controller.

Remark This API function is called as the DMAn_Init callback routine.

[Classification]

CG_dma_user.c

[Syntax]

```
void    DMAn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Enable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Enables operation of channel *n*.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Enable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Disable

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Disables operation of channel *n*.

Remarks 1. This API function does not forcibly terminate DMA transfer.

2. Before using this API function, you must confirm that transmission has ended via DMAn_CheckStatus.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Disable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows setting the operation mode of channel 0 to "disabled".

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    while ( MD_COMPLETED == DMA0_CheckStatus () ); /* Check transfer status */
    DMA0_Disable ();                               /* Change to operation disabled status */
    ......
}
```

DMA*n*_CheckStatus

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Reads the transfer status (transfer complete/transfer ongoing).

[Classification]

CG_dma.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_CheckStatus ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

Macro	Description
MD_UNDEREXEC	Transfer ongoing
MD_COMPLETED	Transfer complete

DMA*n*_SetData

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.

Remark Calling this API function while a transfer is ongoing will end the transfer.

[Classification]

CG_dma.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( UINT srcaddr, UINT dstaddr, UINT count );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UINT srcaddr;`	RAM address of source
I	`UINT dstaddr;`	RAM address of destination
I	`UINT count;`	Number of data transmissions (1 to 1024)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

DMA*n*_SoftwareTriggerOn

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Uses a software trigger as a DMA transfer start trigger.

Remark After this API function is called, DMA transfer will begin if the start DMA transfer software trigger flag STG*n* is set to "1," or the interrupt (e.g. INTP*n* or INTAD) occurs.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_SoftwareTriggerOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

Below is an example of software trigger as a DMA transfer start trigger.

[CG_main.c]

```
void main ( void ) {
    ......
    DMA0_Enable ();             /* Change to operation enabled status */
    DMA0_SoftwareTriggerOn ();  /* Start DMA transfer */
    ......
}
```

3.3.13 LVI

Below is a list of API functions output by Applilet3 for low-voltage detector use.

Table 3-14. API Functions: [LVI]

API Function Name	Function
LVI_Init	Performs initialization necessary to control low-voltage detector functions.
LVI_UserInit	Performs user-defined initialization relating to the low-voltage detector.
LVI_InterruptModeStart	Starts low-voltage detection (when in interrupt generation mode).
LVI_ResetModeStart	Starts low-voltage detection (when in internal reset mode).
LVI_Stop	Stops low-voltage detection.

LVI_Init

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs initialization necessary to control low-voltage detector functions.

[Classification]

CG_lvi.c

[Syntax]

```
void    LVI_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

LVI_UserInit

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Performs user-defined initialization relating to the low-voltage detector.

Remark This API function is called as the LVI_Init callback routine.

[Classification]

CG_lvi_user.c

[Syntax]

```
void    LVI_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

LVI_InterruptModeStart

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts low-voltage detection (when in interrupt generation mode).

[Classification]

CG_lvi.c

[Syntax]

```
void    LVI_InterruptModeStart ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows the detection of low voltage when the operation mode is interrupt generation mode (generate the interrupt INTLVI).

[CG_main.c]

```
void main ( void ) {
    ......
    LVI_InterruptModeStart ( );         /* Start low-voltage detection */
    ......
}
```

[CG_lvi_user.c]

```
__interrupt void MD_INTLVI ( void ) {   /* Interrupt processing for INTLVI */
    if ( LVIF == 1 ) {                  /* Trigger identification: Check LVIF flag */
        ...... /* Handle case when "power voltage (VDD) < detected voltage (VLVI)" detected */
    } else {
        ...... /* Handle case when "power voltage (VDD) >= detected voltage (VLVI)" detected */
    }
}
```

LVI_ResetModeStart

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Starts low-voltage detection (when in internal reset mode).

[Classification]

CG_lvi.c

[Syntax]

```
MD_STATUS   LVI_ResetModeStart ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - The program is configured to not use the low-voltage detector function. - The object of low voltage detection is external voltage (V_{DD}), and power voltage (V_{DD}) <= detected voltage (V_{LVI}). - The object of low voltage detection is external input voltage (EXLVI), and external input voltage (EXLVI) <= detected voltage (V_{EXLVI}).

LVI_Stop

[E/Sx3-H] [ES/Jx3] [ES/Jx3-L]

Stops low-voltage detection.

[Classification]

CG_lvi.c

[Syntax]

```
void    LVI_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

APPENDIX A INDEX

A

A/D ... 122
- AD_Init ... 123
- AD_Read ... 129
- AD_ReadByte ... 130
- AD_SelectADChannel ... 127
- AD_SetPFTCondition ... 128
- AD_Start ... 125
- AD_Stop ... 126
- AD_UserInit ... 124

AD_Init ... 123
AD_Read ... 129
AD_ReadByte ... 130
AD_SelectADChannel ... 127
AD_SetPFTCondition ... 128
AD_Start ... 125
AD_Stop ... 126
AD_UserInit ... 124
API functions ... 19
- A/D ... 122
- D/A ... 131
- DMA ... 209
- External Bus ... 42
- INT ... 51
- LVI ... 217
- Port ... 45
- Real-Time Output ... 200
- RTC ... 169
- Serial ... 62
- System ... 28
- Timer ... 137
- Watch Timer ... 164

B

BUS_Init ... 43
BUS_UserInit ... 44

C

CG_ChangeClockMode ... 32
CG_ChangeFrequency ... 33
CG_ReadResetSource ... 31
CG_SelectPllMode ... 36, 37
CG_SelectPowerSaveMode ... 34
CG_SelectStabTime ... 35
CLOCK_Init ... 29
CLOCK_UserInit ... 30
CRC_GetResult ... 41
CRC_SetData ... 40
CRC_Start ... 39
CSIBn_ErrorCallback ... 96
CSIBn_Init ... 87
CSIBn_ReceiveData ... 92
CSIBn_ReceiveEndCallback ... 95
CSIBn_SendData ... 91
CSIBn_SendEndCallback ... 94
CSIBn_SendReceiveData ... 93
CSIBn_Start ... 89
CSIBn_Stop ... 90
CSIBn_UserInit ... 88
CSIEn_ErrorCallback ... 106
CSIEn_Init ... 97
CSIEn_ReceiveData ... 102
CSIEn_ReceiveEndCallback ... 105
CSIEn_SendData ... 101
CSIEn_SendEndCallback ... 104
CSIEn_SendReceiveData ... 103
CSIEn_Start ... 99
CSIEn_Stop ... 100
CSIEn_UserInit ... 98

D

D/A ... 131
- DAn_Init ... 132
- DAn_SetValue ... 136
- DAn_Start ... 134
- DAn_Stop ... 135
- DAn_UserInit ... 133

DAn_Init ... 132
DAn_SetValue ... 136
DAn_Start ... 134
DAn_Stop ... 135
DAn_UserInit ... 133
DMA ... 209
DMAn_CheckStatus ... 214
DMAn_Disable ... 213
DMAn_Enable ... 212
DMAn_Init ... 210
DMAn_SetData ... 215
DMAn_SoftwareTriggerOn ... 216
DMAn_UserInit ... 211
DMAn_CheckStatus ... 214
DMAn_Disable ... 213
DMAn_Enable ... 212
DMAn_Init ... 210
DMAn_SetData ... 215
DMAn_SoftwareTriggerOn ... 216
DMAn_UserInit ... 211

E

External Bus ... 42
BUS_Init ... 43
BUS_UserInit ... 44

I

IIC0n_GetStopConditionCallback ... 121
IIC0n_Init ... 107
IIC0n_MasterErrorCallback ... 115
IIC0n_MasterReceiveEndCallback ... 114
IIC0n_MasterReceiveStart ... 112
IIC0n_MasterSendEndCallback ... 113
IIC0n_MasterSendStart ... 111
IIC0n_SlaveErrorCallback ... 120
IIC0n_SlaveReceiveEndCallback ... 119
IIC0n_SlaveReceiveStart ... 117
IIC0n_SlaveSendEndCallback ... 118
IIC0n_SlaveSendStart ... 116
IIC0n_Stop ... 109
IIC0n_StopCondition ... 110
IIC0n_UserInit ... 108
INT ... 51
INT_MaskableInterruptEnable ... 56
INTP_Init ... 52
INTPn_Disable ... 58
INTPn_Enable ... 59
INTP_UserInit ... 53
KEY_Disable ... 60
KEY_Enable ... 61
KEY_Init ... 54
KEY_UserInit ... 55
INT_MaskableInterruptEnable ... 56
INTP_Init ... 52
INTPn_Disable ... 58
INTPn_Enable ... 59
INTP_UserInit ... 53

K

KEY_Disable ... 60
KEY_Enable ... 61
KEY_Init ... 54
KEY_UserInit ... 55

L

LVI ... 217
LVI_Init ... 218
LVI_InterruptModeStart ... 220
LVI_ResetModeStart ... 221
LVI_Stop ... 222
LVI_UserInit ... 219
LVI_Init ... 218
LVI_InterruptModeStart ... 220
LVI_ResetModeStart ... 221
LVI_Stop ... 222
LVI_UserInit ... 219

P

Port ... 45
PORT_ChangePmnInput ... 48
PORT_ChangePmnOutput ... 49
PORT_Init ... 46
PORT_UserInit ... 47
PORT_ChangePmnInput ... 48
PORT_ChangePmnOutput ... 49
PORT_Init ... 46
PORT_UserInit ... 47

R

Real-Time Output ... 200
- RTOn_Disable ... 204
- RTOn_Enable ... 203
- RTOn_GetValue ... 208
- RTOn_Init ... 201
- RTOn_Set2BitData ... 205
- RTOn_Set4BitData ... 206
- RTOn_Set6BitData ... 207
- RTOn_UserInit ... 202

RTC ... 169
- RTC_AlarmDisable ... 182
- RTC_AlarmEnable ... 181
- RTC_AlarmGet ... 186
- RTC_AlarmInterruptCallback ... 187
- RTC_AlarmSet ... 183
- RTC_ChangeCorrectionValue ... 199
- RTC_ConstPeriodInterruptCallback ... 180
- RTC_ConstPeriodInterruptDisable ... 179
- RTC_ConstPeriodInterruptEnable ... 178
- RTC_CounterDisable ... 173
- RTC_CounterEnable ... 172
- RTC_CounterGet ... 177
- RTC_CounterSet ... 175
- RTC_Init ... 170
- RTC_IntervalInterruptDisable ... 192
- RTC_IntervalInterruptEnable ... 190
- RTC_IntervalStart ... 188
- RTC_IntervalStop ... 189
- RTC_RC1CK1HZ_OutputDisable ... 194
- RTC_RC1CK1HZ_OutputEnable ... 193
- RTC_RC1CKDIV_OutputDisable ... 198
- RTC_RC1CKDIV_OutputEnable ... 197
- RTC_RC1CKO_OutputDisable ... 196
- RTC_RC1CKO_OutputEnable ... 195
- RTC_SetHourSystem ... 174
- RTC_UserInit ... 171

RTC_AlarmDisable ... 182

RTC_AlarmEnable ... 181

RTC_AlarmGet ... 186

RTC_AlarmInterruptCallback ... 187

RTC_AlarmSet ... 183

RTC_ChangeCorrectionValue ... 199

RTC_ConstPeriodInterruptCallback ... 180

RTC_ConstPeriodInterruptDisable ... 179

RTC_ConstPeriodInterruptEnable ... 178

RTC_CounterDisable ... 173

RTC_CounterEnable ... 172

RTC_CounterGet ... 177

RTC_CounterSet ... 175

RTC_Init ... 170

RTC_IntervalInterruptDisable ... 192

RTC_IntervalInterruptEnable ... 190

RTC_IntervalStart ... 188

RTC_IntervalStop ... 189

RTC_RC1CK1HZ_OutputDisable ... 194

RTC_RC1CK1HZ_OutputEnable ... 193

RTC_RC1CKDIV_OutputDisable ... 198

RTC_RC1CKDIV_OutputEnable ... 197

RTC_RC1CKO_OutputDisable ... 196

RTC_RC1CKO_OutputEnable ... 195

RTC_SetHourSystem ... 174

RTC_UserInit ... 171

RTOn_Disable ... 204

RTOn_Enable ... 203

RTOn_GetValue ... 208

RTOn_Init ... 201

RTOn_Set2BitData ... 205

RTOn_Set4BitData ... 206

RTOn_Set6BitData ... 207

RTOn_UserInit ... 202

S

Serial ... 62
- CSIBn_ErrorCallback ... 96
- CSIBn_Init ... 87
- CSIBn_ReceiveData ... 92
- CSIBn_ReceiveEndCallback ... 95
- CSIBn_SendData ... 91
- CSIBn_SendEndCallback ... 94
- CSIBn_SendReceiveData ... 93
- CSIBn_Start ... 89
- CSIBn_Stop ... 90
- CSIBn_UserInit ... 88
- CSIEn_ErrorCallback ... 106
- CSIEn_Init ... 97

CSIEn_ReceiveData ... 102
CSIEn_ReceiveEndCallback ... 105
CSIEn_SendData ... 101
CSIEn_SendEndCallback ... 104
CSIEn_SendReceiveData ... 103
CSIEn_Start ... 99
CSIEn_Stop ... 100
CSIEn_UserInit ... 98
IIC0n_GetStopConditionCallback ... 121
IIC0n_Init ... 107
IIC0n_MasterErrorCallback ... 115
IIC0n_MasterReceiveEndCallback ... 114
IIC0n_MasterReceiveStart ... 112
IIC0n_MasterSendEndCallback ... 113
IIC0n_MasterSendStart ... 111
IIC0n_SlaveErrorCallback ... 120
IIC0n_SlaveReceiveEndCallback ... 119
IIC0n_SlaveReceiveStart ... 117
IIC0n_SlaveSendEndCallback ... 118
IIC0n_SlaveSendStart ... 116
IIC0n_Stop ... 109
IIC0n_StopCondition ... 110
IIC0n_UserInit ... 108
UARTAn_ErrorCallback ... 73
UARTAn_Init ... 65
UARTAn_ReceiveData ... 70
UARTAn_ReceiveEndCallback ... 72
UARTAn_SendData ... 69
UARTAn_SendEndCallback ... 71
UARTAn_SoftOverRunCallback ... 74
UARTAn_Start ... 67
UARTAn_Stop ... 68
UARTAn_UserInit ... 66
UARTBn_FIFOErrorCallback ... 84
UARTBn_Init ... 75
UARTBn_ReceiveData ... 80
UARTBn_ReceiveEndCallback ... 82
UARTBn_SendData ... 79
UARTBn_SendEndCallback ... 81
UARTBn_SingleErrorCallback ... 83
UARTBn_SoftOverRunCallback ... 86
UARTBn_Start ... 77
UARTBn_Stop ... 78
UARTBn_TimeoutErrorCallback ... 85
UARTBn_UserInit ... 76
System ... 28
CG_ChangeClockMode ... 32
CG_ChangeFrequency ... 33
CG_ReadResetSource ... 31
CG_SelectPllMode ... 36, 37
CG_SelectPowerSaveMode ... 34
CG_SelectStabTime ... 35
CLOCK_Init ... 29
CLOCK_UserInit ... 30
CRC_GetResult ... 41
CRC_SetData ... 40
CRC_Start ... 39
WDT2_Restart ... 38

T

Timer ... 137
TMMn_ChangeTimerCondition ... 163
TMMn_Init ... 159
TMMn_Start ... 161
TMMn_Stop ... 162
TMMn_UserInit ... 160
TMPn_ChangeDuty ... 146
TMPn_ChangeTimerCondition ... 142
TMPn_GetFreeRunningValue ... 145
TMPn_GetPulseWidth ... 144
TMPn_Init ... 138
TMPn_SoftwareTriggerOn ... 147
TMPn_Start ... 140
TMPn_Stop ... 141
TMPn_UserInit ... 139
TMQ0_ChangeDuty ... 156
TMQ0_ChangeTimerCondition ... 152
TMQ0_GetFreeRunningValue ... 155
TMQ0_GetPulseWidth ... 154
TMQ0_Init ... 148
TMQ0_SoftwareTriggerOn ... 158
TMQ0_Start ... 150
TMQ0_Stop ... 151
TMQ0_UserInit ... 149
TMMn_ChangeTimerCondition ... 163
TMMn_Init ... 159

TMMn_Start ... 161
TMMn_Stop ... 162
TMMn_UserInit ... 160
TMPn_ChangeDuty ... 146
TMPn_ChangeTimerCondition ... 142
TMPn_GetFreeRunningValue ... 145
TMPn_GetPulseWidth ... 144
TMPn_Init ... 138
TMPn_SoftwareTriggerOn ... 147
TMPn_Start ... 140
TMPn_Stop ... 141
TMPn_UserInit ... 139
TMQ0_ChangeDuty ... 156
TMQ0_ChangeTimerCondition ... 152
TMQ0_GetFreeRunningValue ... 155
TMQ0_GetPulseWidth ... 154
TMQ0_Init ... 148
TMQ0_SoftwareTriggerOn ... 158
TMQ0_Start ... 150
TMQ0_Stop ... 151
TMQ0_UserInit ... 149

U

UARTAn_ErrorCallback ... 73
UARTAn_Init ... 65
UARTAn_ReceiveData ... 70
UARTAn_ReceiveEndCallback ... 72
UARTAn_SendData ... 69
UARTAn_SendEndCallback ... 71
UARTAn_SoftOverRunCallback ... 74
UARTAn_Start ... 67
UARTAn_Stop ... 68
UARTAn_UserInit ... 66
UARTBn_FIFOErrorCallback ... 84
UARTBn_Init ... 75
UARTBn_ReceiveData ... 80
UARTBn_ReceiveEndCallback ... 82
UARTBn_SendData ... 79
UARTBn_SendEndCallback ... 81
UARTBn_SingleErrorCallback ... 83
UARTBn_SoftOverRunCallback ... 86
UARTBn_Start ... 77
UARTBn_Stop ... 78
UARTBn_TimeoutErrorCallback ... 85
UARTBn_UserInit ... 76

W

Watch Timer ... 164
    WT_Init ... 165
    WT_Start ... 167
    WT_Stop ... 168
    WT_UserInit ... 166
WDT2_Restart ... 38
WT_Init ... 165
WT_Start ... 167
WT_Stop ... 168
WT_UserInit ... 166

For further information,
please contact:

NEC Electronics Corporation

1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/