NEC

Preliminary User's Manual

Applilet3

Device Driver Configurator

API Reference

Target Device

78K0 Microcontroller

Document No. ZUD-CD-09-0241-01 (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 ... 18

APPENDIX A INDEX ... 182

- 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 ... 18

3.1 Overview ... 18
3.2 Output Function ... 18
3.3 Function Reference ... 24
3.3.1 System ... 26
3.3.2 Port ... 36
3.3.3 INT ... 43
3.3.4 Serial ... 54
3.3.5 OPAMP ... 93
3.3.6 A/D ... 101
3.3.7 Timer ... 111
3.3.8 Watchdog Timer ... 135
3.3.9 RTC ... 137
3.3.10 Clock Output ... 169
3.3.11 LVI ... 175

APPENDIX A INDEX ... 182

LIST OF FIGURES

Figure No. Title, Page

3-1 Notation Format of API Functions ... 24

LIST OF TABLES

Table No.	Title, Page
2-1	File List ... 14
3-1	API Function List ... 18
3-2	API Functions: [System] ... 26
3-3	API Functions: [Port] ... 36
3-4	API Functions: [INT] ... 43
3-5	API Functions: [Serial] ... 54
3-6	API Functions: [OPAMP] ... 93
3-7	API Functions: [A/D] ... 101
3-8	API Functions: [Timer] ... 111
3-9	API Functions: [Watchdog Timer] ... 135
3-10	API Functions: [RTC] ... 137
3-11	API Functions: [Clock Output] ... 169
3-12	API Functions: [LVI] ... 175

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+ 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_system_user.c	CLOCK_UserInit CG_ReadResetSource
	CG_system.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_INTP*n* MD_INTKR INTP_UserInit KEY_UserInit
	CG_int.h	-
Serial	CG_serial.c	UART6_Init UART6_Start UART6_Stop UART6_SendData UART6_ReceiveData CSI1n_Init CSI1n_Start CSI1n_Stop CSI1n_ReceiveData CSI1n_SendReceiveData IICA_Init IICA_Stop IICA_MasterSendStart IICA_MasterReceiveStart IICA_StopCondition IICA_SlaveSendStart IICA_SlaveReceiveStart
	CG_serial_user.c	MD_INTSR6

Peripheral Function	File Name	Names of API Functions Included
Serial	CG_serial_user.c	MD_INTSRE6 MD_INTST6 MD_INTCSI1*n* MD_INTIICA0 UART6_UserInit UART6_SendEndCallback UART6_ReceiveEndCallback UART6_SoftOverRunCallback UART6_ErrorCallback CSI1n_UserInit CSI1n_SendEndCallback CSI1n_ReceiveEndCallback IICA_UserInit IICA_MasterSendEndCallback IICA_MasterReceiveEndCallback IICA_MasterErrorCallback IICA_SlaveSendEndCallback IICA_SlaveReceiveEndCallback IICA_SlaveErrorCallback IICA_GetStopConditionCallback
	CG_serial.h	-
OPAMP	CG_opamp.c	OPAMP_Init PGA_Start PGA_Stop PGA_ChangePGAFactor AMPn_Start AMPn_Stop
	CG_opamp_user.c	OPAMP_UserInit
	CG_opamp.h	-
A/D	CG_ad.c	AD_Init AD_ComparatorOn AD_ComparatorOff AD_Start AD_Stop AD_SelectADChannel AD_Read AD_ReadByte
	CG_ad_user.c	MD_INTAD AD_UserInit
	CG_ad.h	-
Timer	CG_timer.c	TM00_Init TM00_Start TM00_Stop TM00_ChangeTimerCondition TM00_GetFreeRunningValue TM00_SoftwareTriggerOn TM00_ChangeDuty TM00_GetPulseWidth

Peripheral Function	File Name	Names of API Functions Included
Timer	CG_timer.c	TM5n_Init TM5n_Start TM5n_Stop TM5n_ChangeTimerCondition TM5n_ChangeDuty TMHn_Init TMHn_Start TMHn_Stop TMHn_ChangeTimerCondition TMHn_ChangeDuty TMH1_CarrierOutputEnable TMH1_CarrierOutputDisable
	CG_timer_user.c	MD_INTTM0*n*0 TM00_UserInit TM5n_UserInit TMHn_UserInit
	CG_timer.h	-
Watchdog Timer	CG_wdt.c	WDT_Restart
	CG_wdt.h	-
RTC	CG_rtc.c	RTC_Init RTC_PowerOff 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_RTC1HZ_OutputEnable RTC_RTC1HZ_OutputDisable RTC_RTCCL_OutputEnable RTC_RTCCL_OutputDisable RTC_RTCDIV_OutputEnable RTC_RTCDIV_OutputDisable RTC_ChangeCorrectionValue
	CG_rtc_user.c	MD_INTRTC MD_INTRTCI RTC_UserInit RTC_ConstPeriodInterruptCallback RTC_AlarmInterruptCallback

Peripheral Function	File Name	Names of API Functions Included
RTC	CG_rtc.h	-
Clock Output	CG_pcl.c	PCL_Init PCL_Start PCL_Stop PCL_ChangeFreq
	CG_pcl_user.c	PCL_UserInit
	CG_pcl.h	-
LVI	CG_lvi.c	LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Stop LVI_SetLVILevel
	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 required to control the clock generator, on-chip debug, and etc. .
	CLOCK_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .
	CG_ReadResetSource	Performs processing in response to RESET signal.
	CG_ChangeClockMode	Changes the CPU clock/peripheral hardware clock.
	CG_ChangeFrequency	Changes the division ratio of the CPU clock/peripheral hardware clock.
	CG_SelectPowerSaveMode	Configures the CPU's standby function.
	CG_SelectStabTime	Configures the oscillation stabilization time of the X1 clock.
Port	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.
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.

Peripheral Function	API Function Name	Function
INT	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	UART6_Init	Performs initialization of the serial interface (UART6) channel.
	UART6_UserInit	Performs user-defined initialization of the serial interface (UART6).
	UART6_Start	Sets UART communication to standby mode.
	UART6_Stop	Ends UART communication.
	UART6_SendData	Starts UART data transmission.
	UART6_ReceiveData	Starts UART data reception.
	UART6_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST6.
	UART6_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR6.
	UART6_SoftOverRunCallback	Performs processing in response to the UART reception complete interrupt INTSR6.
	UART6_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE6.
	CSI1n_Init	Performs initialization of the serial interface (CSI1*n*) channel.
	CSI1n_UserInit	Performs user-defined initialization of the serial interface (CSI1*n*).
	CSI1n_Start	Sets CSI1*n* communication to standby mode.
	CSI1n_Stop	Ends CSI1*n* communication.
	CSI1n_ReceiveData	Starts CSI1*n* data reception.
	CSI1n_SendReceiveData	Starts CSI1*n* data transmission/reception.
	CSI1n_SendEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.
	CSI1n_ReceiveEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.
	IICA_Init	Performs initialization of the serial interface (IICA).
	IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
	IICA_Stop	Ends IICA communication.
	IICA_MasterSendStart	Starts IICA master transmission.
	IICA_MasterReceiveStart	Starts IICA master reception.
	IICA_StopCondition	Generates stop conditions.
	IICA_MasterSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.

Peripheral Function	API Function Name	Function
Serial	IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
	IICA_MasterErrorCallback	Performs processing in response to detection of error in IICA master communication.
	IICA_SlaveSendStart	Starts IICA slave transmission.
	IICA_SlaveReceiveStart	Starts IICA slave reception.
	IICA_SlaveSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
	IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
	IICA_SlaveErrorCallback	Performs processing in response to detection of error in IICA slave communication.
	IICA_GetStopConditionCallback	Performs processing in response to detection of stop condition in IICA slave communication.
OPAMP	OPAMP_Init	Performs initialization necessary to control operational amplifier functions.
	OPAMP_UserInit	Performs user-defined initialization relating to the operational amplifier.
	PGA_Start	Starts the operation of operational amplifier 0 (PGA mode).
	PGA_Stop	Ends the operation of operational amplifier 0 (PGA mode).
	PGA_ChangePGAFactor	Sets the input voltage amplification factor of a operational amplifier 0 (PGA mode).
	AMPn_Start	Starts the operation of operational amplifier *n* (single AMP mode).
	AMPn_Stop	Ends the operation of operational amplifier *n* (single AMP mode).
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_ComparatorOn	Enables operation of voltage converter.
	AD_ComparatorOff	Disables operation of voltage 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_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).
Timer	TM00_Init	Performs initialization necessary to control 16-bit timer/event counter 00 functions.
	TM00_UserInit	Performs user-defined initialization relating to the 16-bit timer/event counter 00.
	TM00_Start	Starts the count for 16-bit timer/event counter 00.

Peripheral Function	API Function Name	Function
Timer	TM00_Stop	Ends the count for 16-bit timer/event counter 00.
	TM00_ChangeTimerCondition	Changes the value of capture/compare control register 00 (CRC00).
	TM00_GetFreeRunningValue	Captures the content of the capture register (CR0*n*0).
	TM00_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
	TM00_ChangeDuty	Changes the duty ratio of the signal output to the TO00 pin.
	TM00_GetPulseWidth	Captures the high/low-level width measured for the signal (pulses) input to the TI0*n*0 pin.
	TM5n_Init	Performs initialization necessary to control 8-bit timer/ event counter 5*n* functions.
	TM5n_UserInit	Performs user-defined initialization relating to the 8-bit timer/event counter 5*n*.
	TM5n_Start	Starts the count for 8-bit timer/event counter 5*n*.
	TM5n_Stop	Ends the count for 8-bit timer/event counter 5*n*.
	TM5n_ChangeTimerCondition	Changes the value of 8-bit timer compare register 5*n* (CR5*n*).
	TM5n_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO5*n* pin.
	TMHn_Init	Performs initialization necessary to control 8-bit timer H*n* functions.
	TMHn_UserInit	Performs user-defined initialization relating to the 8-bit timer H*n*.
	TMHn_Start	Starts the count for 8-bit timer H*n*.
	TMHn_Stop	Ends the count for 8-bit timer H*n*.
	TMHn_ChangeTimerCondition	Changes the value of 8-bit timer H compare register 0*n*/1*n* (CMP0*n*/CMP1*n*).
	TMHn_ChangeDuty	Changes the duty ratio of the PWM signal output to the TOH*n* pin.
	TMH1_CarrierOutputEnable	Begins carrier pulse output of the 8-bit timer H1 (carrier generator mode).
	TMH1_CarrierOutputDisable	Ends carrier pulse output of the 8-bit timer H1 (carrier generator mode).
Watchdog Timer	WDT_Restart	Clears the watchdog timer counter and resumes counting.
RTC	RTC_Init	Performs initialization necessary to control real-time counter functions.
	RTC_UserInit	Performs user-defined initialization relating to the real-time counter.
	RTC_PowerOff	Halts the clock supplied 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).

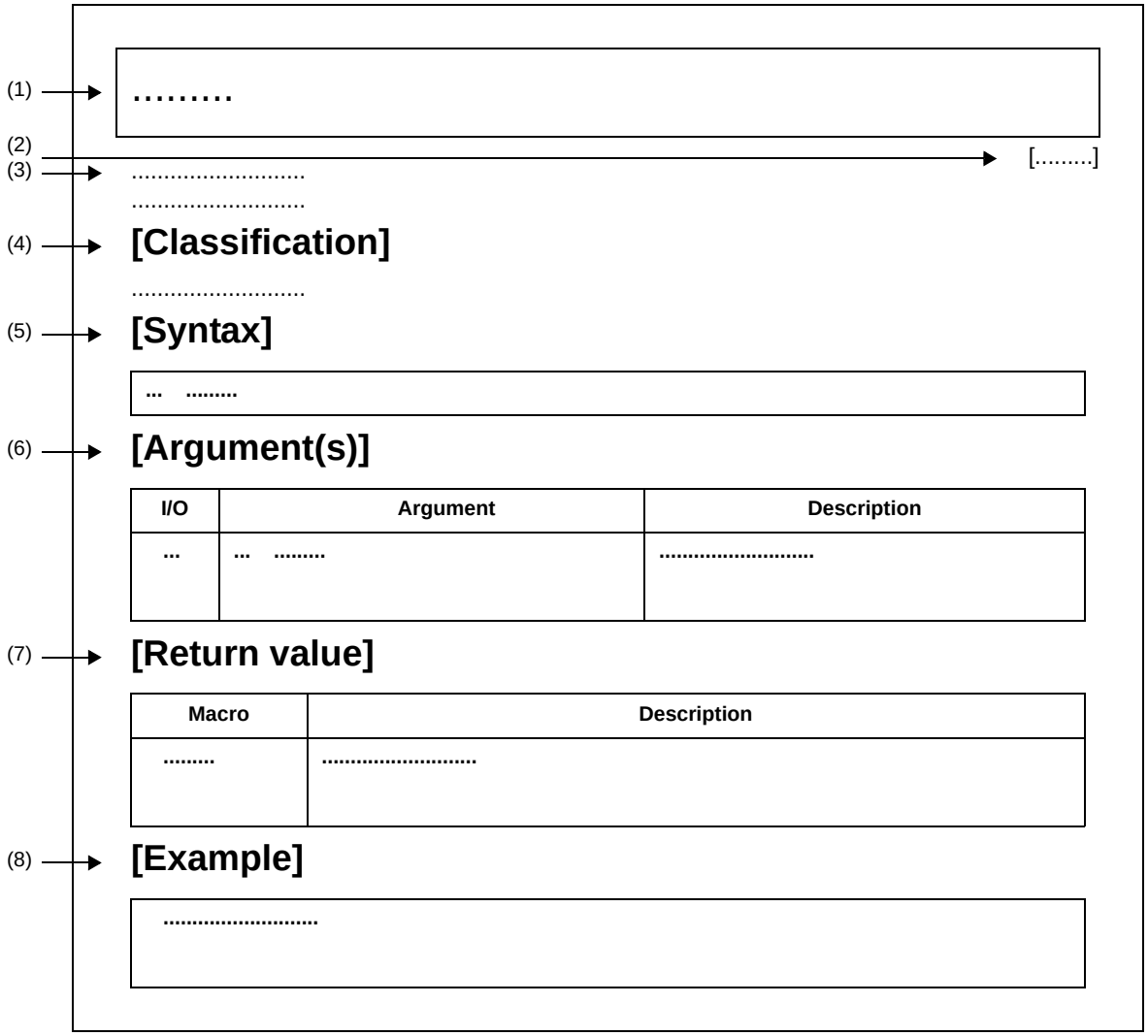
Peripheral Function	API Function Name	Function
RTC	RTC_SetHourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time counter.
	RTC_CounterSet	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_CounterGet	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_ConstPeriodInterruptEnable	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
	RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
	RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC.
	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 INTRTC.
	RTC_IntervalStart	Starts the interval interrupt function.
	RTC_IntervalStop	Ends the interval interrupt function.
	RTC_IntervalInterruptEnable	Sets the cycle of the interrupts, then starts the interval interrupt INTRTCI function.
	RTC_IntervalInterruptDisable	Ends the interval interrupt function.
	RTC_RTC1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
	RTC_RTC1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
	RTC_RTCCL_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCCL_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
	RTC_RTCDIV_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.
Clock Output	PCL_Init	Performs initialization necessary to control clock output control circuit functions.
	PCL_UserInit	Performs user-defined initialization relating to the clock output control circuits.
	PCL_Start	Starts clock output.
	PCL_Stop	Ends clock output.
	PCL_ChangeFreq	Changes the output clock to the PCL pin.
LVI	LVI_Init	Performs initialization necessary to control low-voltage detector functions.

Peripheral Function	API Function Name	Function
LVI	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_SetLVILevel	Sets the low-voltage detection level.

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 required to control the clock generator, on-chip debug, and etc. .
CLOCK_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .
CG_ReadResetSource	Performs processing in response to RESET signal.
CG_ChangeClockMode	Changes the CPU clock/peripheral hardware clock.
CG_ChangeFrequency	Changes the division ratio of the CPU clock/peripheral hardware clock.
CG_SelectPowerSaveMode	Configures the CPU's standby function.
CG_SelectStabTime	Configures the oscillation stabilization time of the X1 clock.

CLOCK_Init

Performs initialization required to control the clock generator, on-chip debug and etc. .

[Classification]

CG_system.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

CLOCK_UserInit

Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .

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

Performs processing in response to 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 ();      /* Processes executed by RESET signal trigger */
    ......
}
```

[CG_system_user.c]

```
#include    "CG_macrodriver.h"
void CG_ReadResetSource ( void ) {
    UCHAR   flag = RESF;        /* Reset control flag register: Obtain RESF contents */
    if ( flag & 0x1 ) {         /* Trigger identification: Check LVIRF flag */
        ......  /* Internal reset request by low-voltage detector */
    } else if ( flag & 0x10 ) { /* Trigger identification: Check WDTRF flag */
        ......  /* Internal reset request by watchdog timer */
    }
    ......
}
```

CG_ChangeClockMode

Changes the CPU clock/peripheral hardware 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 (fCPU)/peripheral hardware clock (fPRS) type HIOCLK: fCPU/fPRS >> Internal high-speed oscillation clock HIOSYSCLK: fCPU >> Internal high-speed oscillation clock, fPRS >> High-spped system clock SYSX1CLK: fCPU/fPRS >> X1 clock SYSEXTCLK: fCPU/fPRS >> External main system clock SUBXT1CLK: fCPU >> XT1 clock SUBEXTCLK: fCPU >> External subsystem clock

Remark SUBXT1CLK and SUBEXTCLK can only be specified when the target device is a 78K0/KC2-L.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend) - Cannot change fCPU/fPRS to the internal high-speed osillation clock.
MD_ERROR2	Exit with error (abend) - Cannot change fCPU to the internal high-speed oscillation clock. - Cannot change fPRS to the high-speed system clock.
MD_ERROR3	Exit with error (abend) - Cannot change fCPU/fPRS to the X1 clock.
MD_ERROR4	Exit with error (abend) - Cannot change fCPU/fPRS to the external main system clock.
MD_ERROR5	Exit with error (abend) - Cannot change fCPU to the XT1 clock.
MD_ERROR6	Exit with error (abend) - Cannot change fCPU to the external subsystem clock.
MD_ARGERROR	Invalid argument specification

Remark The values MD_ERROR5 and MD_ERROR6 will only be returned when the target device is a 78K0/KC2-L.

CG_ChangeFrequency

Changes the division ratio of the CPU clock/peripheral hardware clock.

[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: fMAIN SYSONEHALF: fMAIN/2 SYSONEFOURTH: fMAIN/4 SYSONEEIGHTH: fMAIN/8 SYSONESIXTEENTH: fMAIN/16

Remark "fMAIN" signifies the frequency of the main system clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CG_SelectPowerSaveMode

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - If the CPU is operating by XT1 clock, then STOP mode cannot be specified.
MD_ARGERROR	Invalid argument specification

Remark The value MD_ERROR will only be returned when the target device is a 78K0/KC2-L.

[Example]

Below is an example of changing the standby function to "STOP mode".

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
void main ( void ) {
    MD_STATUS   ret;
    ......
    TM00_Stop ();                               /* Stop count */
    ret = CG_SelectPowerSaveMode ( PSSTOP );    /* Change to STOP mode */
    if ( ret != MD_OK ) {
        while ( 1 );
    }
    TM00_Init ();                               /* Initialize TM00 */
    TM00_Start ();                              /* Start count */
```

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

CG_SelectStabTime

Configures the oscillation stabilization time of the X1 clock.

[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: 2^11/fx STLEVEL1: 2^13/fx STLEVEL2: 2^14/fx STLEVEL3: 2^15/fx STLEVEL4: 2^16/fx

Remark "fx" signifies the frequency of the X1 clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

3.3.2 Port

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

Table 3-3. 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

Performs initialization necessary to control port functions.

[Classification]

CG_port.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PORT_UserInit

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

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

[Classification]

CG_port.c

[Syntax]

The format for specifying this API function differs according to whether the target pin has built-in pull-up resistance/a SMBus input buffer.

[Built-in pull-up resistance: none; SMBus input buffer: none]

```
void    PORT_ChangePmnInput ( void );
```

[Built-in pull-up resistance: yes; SUBus input buffer: none]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnInput ( BOOL enablepu );
```

[Built-in pull-up resistance: yes; SMBus input buffer: yes]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnInput ( BOOL enablepu, BOOL enablesmbus );
```

Remark *mn* is the port number.

[Argument(s)]

I/O	Argument	Description
I	BOOL *enablepu*;	Built-in pull-up resistance used MD_TRUE: Yes MD_FALSE: No
I	BOOL *enablesmbus*;	Input buffer type MD_TRUE: SMBus input buffer MD_FALSE: Normal input buffer

[Return value]

None.

[Example 1]

Below is shown an example where pin P00 (built-in pull-up resistance: yes; SMBus input buffer: none) is changed as follows:

I/O mode type: Input mode
Built-in pull-up resistance used: Yes

[CG_main.c]

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

[Example 2]

Below is shown an example where pin P00 (built-in pull-up resistance: yes; SMBus input buffer: none) is changed as follows:

I/O mode type: Input mode
Built-in pull-up resistance used: No

[CG_main.c]

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

[Example 3]

Below is shown an example where pin P04 (built-in pull-up resistance: yes; SMBus input buffer: yes) is changed as follows:

I/O mode type: Input mode
Built-in pull-up resistance used: No
Input buffer type: SMBus input buffer

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP04Input ( MD_FALSE, MD_TRUE );  /* Switch I/O mode */
    ......
}
```

PORT_ChangeP*mn*Output

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_ChangeP mn Output ( BOOL initialvalue );
```

[N-ch open drain output: yes]

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( 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 */
    ......
}
```

3.3.3 INT

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

Table 3-4. 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

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

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

[KC2-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

[KC2-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

Disables/enables the acceptance of the maskable interrupts.

[Classification]

CG_int.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
MD_STATUS   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]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[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

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

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

[KC2-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

[KC2-L]

Enables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

3.3.4 Serial

Below is a list of API functions output by Applilet3 for serial array unit and serial interface use.

Table 3-5. API Functions: [Serial]

API Function Name	Function
UART6_Init	Performs initialization of the serial interface (UART6) channel.
UART6_UserInit	Performs user-defined initialization of the serial interface (UART6).
UART6_Start	Sets UART communication to standby mode.
UART6_Stop	Ends UART communication.
UART6_SendData	Starts UART data transmission.
UART6_ReceiveData	Starts UART data reception.
UART6_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST6.
UART6_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR6.
UART6_SoftOverRunCallback	Performs processing in response to the serial transfer end interrupt INTSR6.
UART6_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE6.
CSI1n_Init	Performs initialization of the serial interface (CSI1*n*) channel.
CSI1n_UserInit	Performs user-defined initialization of the serial interface (CSI1*n*).
CSI1n_Start	Sets CSI1*n* communication to standby mode.
CSI1n_Stop	Ends CSI1*n* communication.
CSI1n_ReceiveData	Starts CSI1*n* data reception.
CSI1n_SendReceiveData	Starts CSI1*n* data transmission/reception.
CSI1n_SendEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.
CSI1n_ReceiveEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTSCI1*n*.
IICA_Init	Performs initialization of the serial interface (IICA).
IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
IICA_Stop	Ends IICA communication.
IICA_MasterSendStart	Starts IICA master transmission.
IICA_MasterReceiveStart	Starts IICA master reception.
IICA_StopCondition	Generates stop conditions.
IICA_MasterSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
IICA_MasterErrorCallback	Performs processing in response to detection of error in IICA master communication.
IICA_SlaveSendStart	Starts IICA slave transmission.
IICA_SlaveReceiveStart	Starts IICA slave reception.
IICA_SlaveSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.

API Function Name	Function
IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
IICA_SlaveErrorCallback	Performs processing in response to detection of error in IICA slave communication.
IICA_GetStopConditionCallback	Performs processing in response to detection of stop condition in IICA slave communication.

UART6_Init

Performs initialization of the serial interface (UART6) channel.

[Classification]

CG_serial.c

[Syntax]

```
void    UART6_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_UserInit

Performs user-defined initialization of the serial interface (UART6).

Remark This API function is called as the UART6_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UART6_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_Start

Sets UART communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    UART6_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_Stop

Ends UART communication.

[Classification]

CG_serial.c

[Syntax]

```
void    UART6_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_SendData

Starts UART data transmission.

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

2. When performing a UART transmission, UART6_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    “CG_macrodriver.h“
MD_STATUS   UART6_SendData ( UCHAR *txbuf, USHORT txnum );
```

[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

[Example]

Below is an example of sending a UART transmission of four bytes of fixed-length data one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* Transmission complete flag */
void main ( void ) {
    UCHAR   txbuf[] = "ABCD";
    USHORT  txnum = 4;
    gFlag = 1;                          /* Initialize transmission complete flag */
    ......
    UART6_Start ();                     /* Start UART communication*/
    UART6_SendData ( &txbuf, txnum );   /* Start UART data transmission */
    while ( gFlag );                    /* Wait for txnum transmissions */
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* Transmission complete flag */
__interrupt void MD_INTST6 ( void ) {   /* Interrupt processing for INTST6 */
    if ( gUart6TxCnt > 0 ) {
        ......
    } else {
        UART6_SendEndCallback ();       /* Call callback routine */
    }
}

void UART6_SendEndCallback ( void ) {   /* Callback routine for INTST6 */
    gFlag = 0;                          /* Set transmission complete flag */
}
```

UART6_ReceiveData

Starts UART data reception.

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

2. Actual UART reception starts after this API function is called, and UART6_Start is then called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UART6_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

[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

[Example]

Below is an example of UART reception of four bytes of fixed-length data one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                              /* Reception complete flag */
void main ( void ) {
    UCHAR   rxbuf[10];
    USHORT  rxnum = 4;
    gFlag = 1;                              /* Initialize reception complete flag */
    ......
    UART6_ReceiveData ( &rxbuf, rxnum );    /* Start UART data reception */
    UART6_Start ();                         /* Start UART communication */
    while ( gFlag );                        /* Wait for rxnum receptions */
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Reception complete flag */
__interrupt void MD_INTSR6 ( void ) {       /* Interrupt processing for INTSR6 */
    ......
    if ( gUart6RxLen > gUart6RxCnt ) {
        ......
        if ( gUart6RxLen == gUart6RxCnt ) {
            UART6_ReceiveEndCallback ();    /* Call callback routine */
        }
    }
}

void UART6_ReceiveEndCallback ( void ) {    /* Callback routine for INTSR6 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

UART6_SendEndCallback

Performs processing in response to the UART transmission complete interrupt INTST6.

Remark This API function is called as the callback routine of interrupt process MD_INTST6 corresponding to the UART transmission complete interrupt INTST6 (performed when number of transmission data specified by UART6_SendData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UART6_SendEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_ReceiveEndCallback

Performs processing in response to the UART reception complete interrupt INTSR6.

Remark This API function is called as the callback routine of interrupt process MD_INTSR6 corresponding to the UART reception complete interrupt INTSR6 (performed when number of received data specified by UART6_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UART6_ReceiveEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_SoftOverRunCallback

Performs processing in response to the UART reception complete interrupt INTSR6.

Remark This API function is called as the callback routine of interrupt process MD_INTSR6 corresponding to the UART reception complete interrupt INTSR6 (process performed when the amount of data received is greater than the parameter *rxnum* specified for UART6_ReceiveData).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UART6_SoftOverRunCallback ( UCHAR rx_data );
```

[Argument(s)]

I/O	Argument	Description
O	`UCHAR rx_data;`	Received data (data received greater than the number specified in the parameter *rxnum* for UART6_ReceiveData)

[Return value]

None.

UART6_ErrorCallback

Performs processing in response to the UART communication error interrupt INTSRE6.

Remark This API function is called as the callback routine of interrupt process MD_INTSRE6 corresponding to the UART communication error interrupt INTSRE6.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UART6_ErrorCallback ( UCHAR err_type );
```

[Argument(s)]

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

[Return value]

None.

[Example]

Below are examples of callback processing by the trigger for the UART communication error interrupt.

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE6 ( void ) {          /* Interrupt processing for INTSRE6 */
    UCHAR   err_type;
    ......
    UART6_ErrorCallback ( err_type );           /* Call callback routine */
}

void UART6_ErrorCallback ( UCHAR err_type ) {   /* Callback routine for INTSRE6 */
    if ( err_type & 0x1 ) {                     /* Determine trigger */
        ......  /* Callback processing in response to overrun error */
    } else if ( err_type & 0x2 ) {              /* Determine trigger */
        ......  /* Callback processing in response to framing error */
    } else if ( err_type & 0x4 ) {              /* Determine trigger */
        ......  /* Callback processing in response to parity error */
    }
}
```

CSI1*n*_Init

[KB2-L] [KC2-L]

Performs initialization of the serial interface (CSI1*n*) channel.

[Classification]

CG_serial.c

[Syntax]

```
void    CSI1n_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_UserInit

[KB2-L] [KC2-L]

Performs user-defined initialization of the serial interface (CSI1*n*).

Remark This API function is called as the CSI1n_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSI1n_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_Start

[KB2-L] [KC2-L]

Sets CSI1*n* communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    CSI1n_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_Stop

[KB2-L] [KC2-L]

Ends CSI1*n* communication.

[Classification]

CG_serial.c

[Syntax]

```
void    CSI1n_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_ReceiveData

[KB2-L] [KC2-L]

Starts CSI1*n* data reception.

Remarks 1. This API function performs byte-level CSI1*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 CSI1*n* reception, CSI1n_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSI1n_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

[Example]

Below is an example of receiving a CSI10 transmission of four bytes of fixed-length data from channel 10 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                              /* Reception complete flag */
void main ( void ) {
    UCHAR   rxbuf[10];
    USHORT  rxnum = 4;
    gFlag = 1;                              /* Initialize reception complete flag */
    ......
    CSI10_Start ();                         /* Start CSI10 communication */
    CSI10_ReceiveData ( &rxbuf, rxnum );    /* Start CSI10 reception */
    while ( gFlag );                        /* Wait for rxnum receptions */
    ......
```

```
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Reception complete flag */
__interrupt void MD_INTCSI10 ( void ) {     /* Interrupt processing for INTCSI10 */
    if ( gCsi10RxCnt < gCsi10RxLen ) {
        ......
        if ( gCsi10RxCnt == gCsi10RxLen ) {
            CSI10_ReceiveEndCallback ();    /* Call callback routine */
        } else {
            ......
        }
    }
}

void CSI10_ReceiveEndCallback ( void ) {    /* Callback routine for INTCSI10 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

CSI1*n*_SendReceiveData

[KB2-L] [KC2-L]

Starts CSI1*n* data transmission/reception.

Remarks 1. This API function repeats the byte-level CSI1*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. This API function performs byte-level CSI1*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 CSI1*n* reception, CSI1n_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSI1n_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

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/receive
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending and receiving a CSI10 transmission of four bytes of fixed-length data from channel 10 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gSflag;                                     /* Transmission complete flag */
void main ( void ) {
    UCHAR   txbuf[] = "0123";
    USHORT  txnum = 4;
    UCHAR   rxbuf[10];
    gSflag = 1;                                     /* Initialize flag */
```

```
    ......
    CSI10_Start ();                                     /* Start CSI10 communication */
    CSI10_SendReceiveData ( &txbuf, txnum, &rxbuf );    /* Start CSI10 send/receive */
    while ( gSlag );                                    /* Wait for txnum transmissions/
receptions */
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gSflag;                     /* Transmission complete flag */
__interrupt void MD_INTCSI10 ( void ) {     /* Interrupt processing for INTCSI10 */
   if ( gCsi10TxCnt > 0 ) {
        ......
    } else {
        ......
        CSI10_SendEndCallback ();           /* Call callback routine */
    }
    ......
}

void CSI10_SendEndCallback ( void ) {       /* Callback routine for INTCSI10 */
    gSflag = 0;                             /* Set transmission complete flag */
}
```

CSI1*n*_SendEndCallback

[KB2-L] [KC2-L]

Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTCSI1*n* corresponding to the CSI1*n* communication complete interrupt INTCSI1*n* (performed when number of transmission data specified by CSI1n_SendReceiveData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSI1n_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_ReceiveEndCallback

[KB2-L] [KC2-L]

Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTCSI1*n* corresponding to the CSI1*n* communication complete interrupt INTCSI1*n* (performed when number of received data specified by CSI1n_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSI1n_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IICA_Init

Performs initialization of the serial interface (IICA).

[Classification]

CG_serial.c

[Syntax]

```
void    IICA_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_UserInit

Performs user-defined initialization of the serial interface (IICA).

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_Stop

Ends IICA communication.

[Classification]

CG_serial.c

[Syntax]

```
void    IICA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendStart

Starts IICA master transmission.

Remark This API function repeats the byte-level IICA 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   IICA_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

[Argument(s)]

I/O	Argument	Description
I	UCHAR *adr*;	Slave 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

[Return value]

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

IICA_MasterReceiveStart

Starts IICA master reception.

Remark This API function performs byte-level IICA 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   IICA_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

[Argument(s)]

I/O	Argument	Description
I	UCHAR *adr*;	Slave 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

[Return value]

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

IICA_StopCondition

Generates stop conditions.

[Classification]

CG_serial.c

[Syntax]

```
void    IICA_StopCondition ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_MasterSendEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterReceiveEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_MasterReceiveEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterErrorCallback

Performs processing in response to detection of error in IICA master communication.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	MD_STATUS *flag*;	Cause of communication error MD_SPT: Stop condition detected MD_NACK: Acknowledge not detected

[Return value]

None.

IICA_SlaveSendStart

Starts IICA slave transmission.

Remark This API function repeats the byte-level IICA 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    IICA_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

[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.

IICA_SlaveReceiveStart

Starts IICA slave reception.

Remark This API function performs byte-level IICA 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    IICA_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

[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.

IICA_SlaveSendEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_SlaveSendEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveReceiveEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_SlaveReceiveEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveErrorCallback

Performs processing in response to detection of error in IICA slave communication.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	MD_STATUS *flag*;	Cause of communication error MD_ERROR: Address mismatch detected MD_NACK: Acknowledge not detected

[Return value]

None.

IICA_GetStopConditionCallback

Performs processing in response to detection of stop condition in IICA slave communication.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_GetStopConditionCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.5 OPAMP

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

Table 3-6. API Functions: [OPAMP]

API Function Name	Function
OPAMP_Init	Performs initialization necessary to control operational amplifier functions.
OPAMP_UserInit	Performs user-defined initialization relating to the operational amplifier.
PGA_Start	Starts the operation of opeational amplifier 0 (PGA mode).
PGA_Stop	Ends the operation of operational amplifier 0 (PGA mode).
PGA_ChangePGAFactor	Sets the input voltage amplification factor of a operational amplifier 0 (PGA mode).
AMPn_Start	Starts the operation of opeational amplifier *n* (single AMP mode).
AMPn_Stop	Ends the operation of operational amplifier *n* (single AMP mode).

OPAMP_Init

[Products with operational amplifier]

Performs initialization necessary to control operational amplifier functions.

[Classification]

CG_opamp.c

[Syntax]

```
void    OPAMP_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

OPAMP_UserInit

[Products with operational amplifier]

Performs user-defined initialization relating to the operational amplifier.

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

[Classification]

CG_opamp_user.c

[Syntax]

```
void    OPAMP_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

PGA_Start

[Products with operational amplifier]

Starts the operation of operational amplifier 0 (PGA mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    PGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

PGA_Stop

[Products with operational amplifier]

Ends the operation of operational amplifier 0 (PGA mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    PGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

PGA_ChangePGAFactor

[Products with operational amplifier]

Sets the input voltage amplification factor of a operational amplifier 0 (PGA mode).

Remark The value specified in parameter *factor* is set to operational amplifier 0 control register (AMP0M).

[Classification]

CG_opamp.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_opamp.h"
MD_STATUS   PGA_ChangePGAFactor ( enum PGAFactor factor );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PGAFactor factor;`	Input voltage amplification factor PGAFACTOR0: x4 PGAFACTOR1: x8 PGAFACTOR2: x16 PGAFACTOR3: x32

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

AMP*n*_Start

[Products with operational amplifier]

Starts the operation of operational amplifier *n* (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

AMP*n*_Stop

[Products with operational amplifier]

Ends the operation of operational amplifier *n* (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMPn_Stop ( 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 A/D 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_ComparatorOn	Enables operation of voltage converter.
AD_ComparatorOff	Disables operation of voltage 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_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

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

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_ComparatorOn

Enables operation of voltage converter.

Remark About 1 microsecond of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to this API function and the call to AD_Start.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_ComparatorOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_ComparatorOff

Disables operation of voltage converter.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_ComparatorOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_Start

Starts A/D conversion.

Remark About 1 micro second of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to AD_ComparatorOn and the call to this API function.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes. Consequently, to stop the operation of the voltage converter, you must call AD_ComparatorOff after the process of this API function completes.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_SelectADChannel

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

Remark The value specified in parameter *channel* is set to analog input channel specification register (ADS).

[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_Read

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

Remark The contents of the 10-bit A/D conversion result register (ADCR) are stored in the area specified by parameter *buffer*.

[Classification]

CG_ad.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    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

[Return value]

None.

AD_ReadByte

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

Remark The contents of the 8-bit A/D conversion result register H (ADCRH) are stored in the area specified by parameter *buffer*.

[Classification]

CG_ad.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    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

[Return value]

None.

3.3.7 Timer

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

Table 3-8. API Functions: [Timer]

API Function Name	Function
TM00_Init	Performs initialization necessary to control 16-bit timer/event counter 00 functions.
TM00_UserInit	Performs user-defined initialization relating to the 16-bit timer/event counter 00.
TM00_Start	Starts the count for 16-bit timer/event counter 00.
TM00_Stop	Ends the count for 16-bit timer/event counter 00.
TM00_ChangeTimerCondition	Changes the value of capture/compare control register 00 (CRC00).
TM00_GetFreeRunningValue	Captures the content of the capture register (CR0*n*0).
TM00_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
TM00_ChangeDuty	Changes the duty ratio of the signal output to the TO00 pin.
TM00_GetPulseWidth	Captures the high/low-level width measured for the signal (pulses) input to the TI0*n*0 pin.
TM5n_Init	Performs initialization necessary to control 8-bit timer/event counter 5*n* functions.
TM5n_UserInit	Performs user-defined initialization relating to the 8-bit timer/event counter 5*n*.
TM5n_Start	Starts the count for 8-bit timer/event counter 5*n*.
TM5n_Stop	Ends the count for 8-bit timer/event counter 5*n*.
TM5n_ChangeTimerCondition	Changes the value of 8-bit timer compare register 5*n* (CR5*n*).
TM5n_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO5*n* pin.
TMHn_Init	Performs initialization necessary to control 8-bit timer H*n* functions.
TMHn_UserInit	Performs user-defined initialization relating to the 8-bit timer H*n*.
TMHn_Start	Starts the count for 8-bit timer H*n*.
TMHn_Stop	Ends the count for 8-bit timer H*n*.
TMHn_ChangeTimerCondition	Changes the value of 8-bit timer H compare register 0*n*/1*n* (CMP0*n*/CMP1*n*).
TMHn_ChangeDuty	Changes the duty ratio of the PWM signal output to the TOH*n* pin.
TMH1_CarrierOutputEnable	Begins carrier pulse output of the 8-bit timer H1 (carrier generator mode).
TMH1_CarrierOutputDisable	Ends carrier pulse output of the 8-bit timer H1 (carrier generator mode).

TM00_Init

Performs initialization necessary to control 16-bit timer/event counter 00 functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_UserInit

Performs user-defined initialization relating to the 16-bit timer/event counter 00.

Remark This API function is called as the TM00_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TM00_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_Start

Starts the count for 16-bit timer/event counter 00.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, square-wave output, or external event counter).

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_Stop

Ends the count for 16-bit timer/event counter 00.

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_ChangeTimerCondition

Changes the value of capture/compare control register 00 (CRC00).

Remark To change the contents of CRC00, you must call TM00_Stop before calling this API function.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TM00_ChangeTimerCondition ( USHORT *array_reg, USHORT array_num );
```

[Argument(s)]

I/O	Argument	Description
I	`USHORT *array_reg;`	Pointer to the area storing the value to set in the target register
I	`USHORT array_num;`	The register to change 1: CR000 2: CR000, CR010

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Invalid argument "*array_num*" specification

TM00_GetFreeRunningValue

Captures the content of the capture register (CR0*n*0).

Remark This API function can only be called when the 16-bit timer/event counter 00 is running in free-running timer mode, and the capture/compare control register 00 (CR0*n*0) is being used as a capture register.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TM00_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

[Argument(s)]

I/O	Argument	Description
I	`ULONG *count;`	Pointer to area in which to store the captured value
I	`enum TMChannel channel;`	The register to capture TMCHANNEL0: CR010 TMCHANNEL1: CR000

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - CR0*n*0 is operating as a compare register.
MD_ARGERROR	Invalid argument specification

TM00_SoftwareTriggerOn

Generates the trigger (software trigger) for one-shot pulse output.

Remark This API function can only be called when the 16-bit timer/event counter 00 is being used for one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_SoftwareTriggerOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_ChangeDuty

Changes the duty ratio of the signal output to the TO00 pin.

Remark This API function can only be called when the 16-bit timer/event counter 00 is being used for PPG output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TM00_ChangeDuty ( UCHAR ratio );
```

[Argument(s)]

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

Remark The value set to duty ratio *ratio* 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 ) {
    UCHAR   ratio = 25;
    ......
    TM00_Start ();                  /* Start count */
    ......
    TM00_ChangeDuty ( ratio );      /* Change duty ratio */
    ......
}
```

TM00_GetPulseWidth

Captures the high/low-level width measured for the signal (pulses) input to the TI0*n*0 pin.

Remark This API function can only be called when the 16-bit timer/event counter 00 is being used for pulse width measurement.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
void    TM00_GetPulseWidth ( ULONG *highwidth, ULONG *lowwidth, enum TMChannel channel );
```

[Argument(s)]

I/O	Argument	Description
O	`ULONG *highwidth;`	Pointer to area storing the high-level measurement width (0x0 to 0xffff)
O	`ULONG *lowwidth;`	Pointer to area storing the low-level measurement width (0x0 to 0xffff)
I	`enum TMChannel channel;`	The pin to measure TMCHANNEL0: TI000 pin TMCHANNEL1: TI010 pin

[Return value]

None.

TM5*n*_Init

Performs initialization necessary to control 8-bit timer/event counter 5*n* functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TM5n_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_UserInit

Performs user-defined initialization relating to the 8-bit timer/event counter 5*n*.

Remark This API function is called as the TM5n_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TM5n_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_Start

Starts the count for 8-bit timer/event counter 5*n*.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, or external event counter).

[Classification]

CG_timer.c

[Syntax]

```
void    TM5n_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_Stop

Ends the count for 8-bit timer/event counter 5*n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TM5n_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_ChangeTimerCondition

Changes the value of 8-bit timer compare register 5*n* (CR5*n*).

Remark To change the contents of CR5*n*, you must call TM5n_Stop before calling this API function.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TM5n_ChangeTimerCondition ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR valueg;`	The value to set in CR5*n*

[Return value]

None.

TM5*n*_ChangeDuty

Changes the duty ratio of the PWM signal output to the TO5*n* pin.

Remark This API function can only be called when the 8-bit timer/event counter 5*n* is being used for PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TM5n_ChangeDuty ( UCHAR ratio );
```

Remark *n* is the channel number.

[Argument(s)]

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

Remark The value set to duty ratio *ratio* 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 ) {
    UCHAR   ratio = 25;
    ......
    TM50_Start ();                  /* Start count */
    ......
    TM50_ChangeDuty ( ratio );      /* Change duty ratio */
    ......
}
```

TMH*n*_Init

Performs initialization necessary to control 8-bit timer H*n* functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMHn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_UserInit

Performs user-defined initialization relating to the 8-bit timer H*n*.

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

[Classification]

CG_timer_user.c

[Syntax]

```
void    TMHn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_Start

Starts the count for 8-bit timer H*n*.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, square-wave output, or PWM output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMHn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_Stop

Ends the count for 8-bit timer H*n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TMHn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_ChangeTimerCondition

Changes the value of 8-bit timer H compare register 0*n*/1*n* (CMP0*n*/CMP1*n*).

Remark To change the contents of CMP0*n*/CMP1*n*, you must call TMHn_Stop before calling this API function.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMHn_ChangeTimerCondition ( UCHAR *array_reg, UCHAR array_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR *array_reg;`	Pointer to the area storing the value to set in the target register
I	`UCHAR array_num;`	The register to change 1: CMP0*n* 2: CMP0*n*, CMP1*n*

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

TMH*n*_ChangeDuty

Changes the duty ratio of the PWM signal output to the TOH*n* pin.

Remark This API function can only be called when the 8-bit timer H*n* is being used for PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TMHn_ChangeDuty ( UCHAR ratio );
```

Remark *n* is the channel number.

[Argument(s)]

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

Remark The value set to duty ratio *ratio* 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 ) {
    UCHAR   ratio = 25;
    ......
    TMH0_Start ();                  /* Start count */
    ......
    TMH0_ChangeDuty ( ratio );      /* Change duty ratio */
    ......
}
```

TMH1_CarrierOutputEnable

Begins carrier pulse output of the 8-bit timer H1 (carrier generator mode).

[Classification]

CG_timer.c

[Syntax]

```
void    TMH1_CarrierOutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMH1_CarrierOutputDisable

Ends carrier pulse output of the 8-bit timer H1 (carrier generator mode).

[Classification]

CG_timer.c

[Syntax]

```
void    TMH1_CarrierOutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.8 Watchdog Timer

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

Table 3-9. API Functions: [Watchdog Timer]

API Function Name	Function
WDT_Restart	Clears the watchdog timer counter and resumes counting.

WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Classification]

CG_wdt.c

[Syntax]

```
void    WDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.9 RTC

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

Table 3-10. 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_PowerOff	Halts the clock supplied 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 INTRTC, then starts the cyclic interrupt function.
RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC.
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 INTRTC.
RTC_IntervalStart	Starts the interval interrupt function.
RTC_IntervalStop	Ends the interval interrupt function.
RTC_IntervalInterruptEnable	Sets the cycle of the interrupts INTRTCI, then starts the interval interrupt function.
RTC_IntervalInterruptDisable	Ends the interval interrupt function.
RTC_RTC1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RTC1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RTCCL_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RTCCL_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RTCDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
RTC_RTCDIV_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

[KC2-L]

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

[KC2-L]

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_PowerOff

[KC2-L]

Halts the clock supplied to the real-time counter.

Remark Calling this API function changes the real-time counter to reset status. For this reason, writes to the control registers (e.g. real-time counter control register 0: RTCC0) after this API function is called are ignored.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_CounterEnable

[KC2-L]

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

[KC2-L]

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

[KC2-L]

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

[KC2-L]

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

[KC2-L]

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

[KC2-L]

Sets the cycle of the interrupts INTRTC, 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 INTRTC 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 INTRTC, 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

[KC2-L]

Ends the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_ConstPeriodInterruptCallback

[KC2-L]

Performs processing in response to the cyclic interrupt INTRTC.

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

[Classification]

CG_rtc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmEnable

[KC2-L]

Starts the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmDisable

[KC2-L]

Ends the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmSet

[KC2-L]

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

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

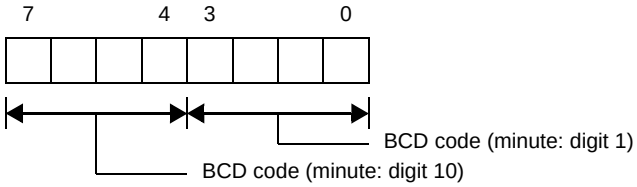
I/O	Argument	Description
I	`struct RTCAlarmValue alarmval;`	Alarm conditions (weekday, hour, minute)

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

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

- Alarmwm (Minute)

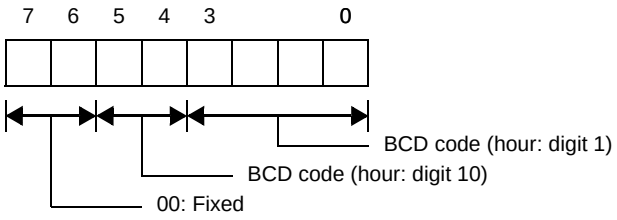
Below are shown the meanings of each bit of the structure member Alarmwm.



- Alarmwh (Hour)

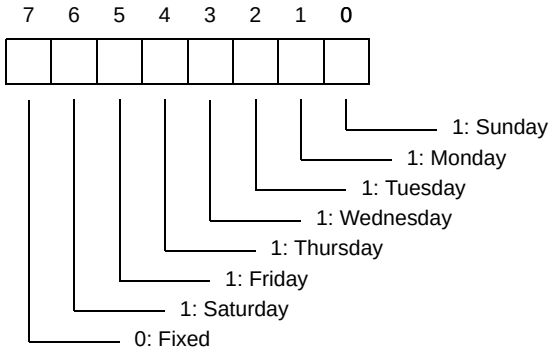
Below are shown the meanings of each bit of the structure member Alarmwh. If the real-time counter is set to the 12-hour clock, then bit 5 has the following meaning.

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

[KC2-L]

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

[KC2-L]

Performs processing in response to the alarm interrupt INTRTC.

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

[Classification]

CG_rtc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalStart

[KC2-L]

Starts the interval interrupt function.

Remark After setting the cycle of the interrupts INTRTCI, 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

[KC2-L]

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalInterruptEnable

[KC2-L]

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

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

[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 INTRTCI cycle INTERVAL0: 2^6/fSUB INTERVAL1: 2^7/fSUB INTERVAL2: 2^8/fSUB INTERVAL3: 2^9/fSUB INTERVAL4: 2^10/fSUB INTERVAL5: 2^11/fSUB INTERVAL6: 2^12/fSUB

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

[KC2-L]

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTC1HZ_OutputEnable

[KC2-L]

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTC1HZ_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTC1HZ_OutputDisable

[KC2-L]

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTC1HZ_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCCL_OutputEnable

[KC2-L]

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCCL_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCCL_OutputDisable

[KC2-L]

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCCL_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCDIV_OutputEnable

[KC2-L]

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCDIV_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCDIV_OutputDisable

[KC2-L]

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCDIV_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_ChangeCorrectionValue

[KC2-L]

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

3.3.10 Clock Output

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

Table 3-11. API Functions: [Clock Output]

API Function Name	Function
PCL_Init	Performs initialization necessary to control clock output control circuit functions.
PCL_UserInit	Performs user-defined initialization relating to the clock output control circuits.
PCL_Start	Starts clock output.
PCL_Stop	Ends clock output.
PCL_ChangeFreq	Changes the output clock to the PCL pin.

PCL_Init

[KC2-L (48 pin)]

Performs initialization necessary to control clock output control circuit functions.

[Classification]

CG_pcl.c

[Syntax]

```
void    PCL_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_UserInit

[KC2-L (48 pin)]

Performs user-defined initialization relating to the clock output control circuits.

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

[Classification]

CG_pcl_user.c

[Syntax]

```
void    PCL_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_Start

[KC2-L (48 pin)]

Starts clock output.

[Classification]

CG_pcl.c

[Syntax]

```
void    PCL_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_Stop

[KC2-L (48 pin)]

Ends clock output.

[Classification]

CG_pcl.c

[Syntax]

```
void    PCL_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_ChangeFreq

[KC2-L (48 pin)]

Changes the output clock to the PCL pin.

Remark The value specified in parameter *clock* is set to clock output select register (CKS).

[Classification]

CG_pcl.c

[Syntax]

```
#include    "CG_pclbuz.h"
MD_STATUS   PCL_ChangeFreq ( enum PCLclock clock );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PCLclock clock;`	Output clock type FPRS: fPRS FPRS2: fPRS/2 FPRS4: fPRS/4 FPRS8: fPRS/8 FPRS16: fPRS/16 FPRS32: fPRS/2048 FPRS64: fPRS/4096 FPRS128: fPRS/8192 SUBCLOCK: fSUB

Remark fPRS is the main system clock frequency; fSUB is the subsystem clock frequency.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

3.3.11 LVI

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

Table 3-12. 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_SetLVILevel	Sets the low-voltage detection level.

LVI_Init

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

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

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

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 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

Stops low-voltage detection.

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

LVI_SetLVILevel

Sets the low-voltage detection level.

Remarks 1. To change the low-voltage detection level, you must call LVI_Stop before calling this API function.

2. The value specified in parameter *level* is set to low-voltage detection level select register (LVIS).

[Classification]

CG_lvi.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_lvi.h"
MD_STATUS   LVI_SetLVILevel ( enum LVILevel level );
```

[Argument(s)]

I/O	Argument	Description
I	`enum LVILevel level;`	Voltage level to detect as low voltage LVILEVEL0: 4.22 V ± 0.1 V LVILEVEL1: 4.07 V ± 0.1 V LVILEVEL2: 3.92 V ± 0.1 V LVILEVEL3: 3.76 V ± 0.1 V LVILEVEL4: 3.61 V ± 0.1 V LVILEVEL5: 3.45 V ± 0.1 V LVILEVEL6: 3.30 V ± 0.1 V LVILEVEL7: 3.15 V ± 0.1 V LVILEVEL8: 2.99 V ± 0.1 V LVILEVEL9: 2.84 V ± 0.1 V LVILEVEL10: 2.68 V ± 0.1 V LVILEVEL11: 2.53 V ± 0.1 V LVILEVEL12: 2.38 V ± 0.1 V LVILEVEL13: 2.22 V ± 0.1 V LVILEVEL14: 2.07 V ± 0.1 V LVILEVEL15: 1.91 V ± 0.1 V

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - The target of low-voltage detection is external input voltage (EXLVI) from the external input pin.
MD_ARGERROR	Invalid argument specification

Remark The value MD_ERROR will only be returned when the target device is a 78K0/KB2-L or 78K0/KC2-L.

APPENDIX A INDEX

A

A/D ... 101
- AD_ComparatorOff ... 105
- AD_ComparatorOn ... 104
- AD_Init ... 102
- AD_Read ... 109
- AD_ReadByte ... 110
- AD_SelectADChannel ... 108
- AD_Start ... 106
- AD_Stop ... 107
- AD_UserInit ... 103

AD_ComparatorOff ... 105
AD_ComparatorOn ... 104
AD_Init ... 102
AD_Read ... 109
AD_ReadByte ... 110
AD_SelectADChannel ... 108
AD_Start ... 106
AD_Stop ... 107
AD_UserInit ... 103
AMPn_Start ... 99
AMPn_Stop ... 100
API functions ... 18
- A/D ... 101
- Clock Output ... 169
- INT ... 43
- LVI ... 175
- OPAMP ... 93
- Port ... 36
- RTC ... 137
- Serial ... 54
- System ... 26
- Timer ... 111
- Watchdog Timer ... 135

C

CG_ChangeClockMode ... 30
CG_ChangeFrequency ... 32
CG_ReadResetSource ... 29
CG_SelectPowerSaveMode ... 33
CG_SelectStabTime ... 35
CLOCK_Init ... 27
Clock Output ... 169
- PCL_ChangeFreq ... 174
- PCL_Init ... 170
- PCL_Start ... 172
- PCL_Stop ... 173
- PCL_UserInit ... 171

CLOCK_UserInit ... 28
CSI1n_Init ... 68
CSI1n_ReceiveData ... 72
CSI1n_ReceiveEndCallback ... 77
CSI1n_SendEndCallback ... 76
CSI1n_SendReceiveData ... 74
CSI1n_Start ... 70
CSI1n_Stop ... 71
CSI1n_UserInit ... 69

I

IICA_GetStopConditionCallback ... 92
IICA_Init ... 78
IICA_MasterErrorCallback ... 86
IICA_MasterReceiveEndCallback ... 85
IICA_MasterReceiveStart ... 82
IICA_MasterSendEndCallback ... 84
IICA_MasterSendStart ... 81
IICA_SlaveErrorCallback ... 91
IICA_SlaveReceiveEndCallback ... 90
IICA_SlaveReceiveStart ... 88
IICA_SlaveSendEndCallback ... 89
IICA_SlaveSendStart ... 87
IICA_Stop ... 80
IICA_StopCondition ... 83
IICA_UserInit ... 79
INT ... 43
- INT_MaskableInterruptEnable ... 48

INTP_Init ... 44
INTPn_Disable ... 50
INTPn_Enable ... 51
INTP_UserInit ... 45
KEY_Disable ... 52
KEY_Enable ... 53
KEY_Init ... 46
KEY_UserInit ... 47
INT_MaskableInterruptEnable ... 48
INTP_Init ... 44
INTPn_Disable ... 50
INTPn_Enable ... 51
INTP_UserInit ... 45

K

KEY_Disable ... 52
KEY_Enable ... 53
KEY_Init ... 46
KEY_UserInit ... 47

L

LVI ... 175
LVI_Init ... 176
LVI_InterruptModeStart ... 178
LVI_ResetModeStart ... 179
LVI_SetLVILevel ... 181
LVI_Stop ... 180
LVI_UserInit ... 177
LVI_Init ... 176
LVI_InterruptModeStart ... 178
LVI_ResetModeStart ... 179
LVI_SetLVILevel ... 181
LVI_Stop ... 180
LVI_UserInit ... 177

O

OPAMP ... 93
AMPn_Start ... 99
AMPn_Stop ... 100
OPAMP_Init ... 94
OPAMP_UserInit ... 95
PGA_ChangePGAFactor ... 98
PGA_Start ... 96
PGA_Stop ... 97
OPAMP_Init ... 94
OPAMP_UserInit ... 95

P

PCL_ChangeFreq ... 174
PCL_Init ... 170
PCL_Start ... 172
PCL_Stop ... 173
PCL_UserInit ... 171
PGA_ChangePGAFactor ... 98
PGA_Start ... 96
PGA_Stop ... 97
Port ... 36
PORT_ChangePmnInput ... 39
PORT_ChangePmnOutput ... 41
PORT_Init ... 37
PORT_UserInit ... 38
PORT_ChangePmnInput ... 39
PORT_ChangePmnOutput ... 41
PORT_Init ... 37
PORT_UserInit ... 38

R

RTC ... 137
RTC_AlarmDisable ... 151
RTC_AlarmEnable ... 150
RTC_AlarmGet ... 155
RTC_AlarmInterruptCallback ... 156
RTC_AlarmSet ... 152
RTC_ChangeCorrectionValue ... 168
RTC_ConstPeriodInterruptCallback ... 149
RTC_ConstPeriodInterruptDisable ... 148
RTC_ConstPeriodInterruptEnable ... 147
RTC_CounterDisable ... 142
RTC_CounterEnable ... 141
RTC_CounterGet ... 146
RTC_CounterSet ... 144
RTC_Init ... 138
RTC_IntervalInterruptDisable ... 161
RTC_IntervalInterruptEnable ... 159
RTC_IntervalStart ... 157
RTC_IntervalStop ... 158
RTC_PowerOff ... 140
RTC_RTC1HZ_OutputDisable ... 163

RTC_RTC1HZ_OutputEnable ... 162
RTC_RTCCL_OutputDisable ... 165
RTC_RTCCL_OutputEnable ... 164
RTC_RTCDIV_OutputDisable ... 167
RTC_RTCDIV_OutputEnable ... 166
RTC_SetHourSystem ... 143
RTC_UserInit ... 139
RTC_AlarmDisable ... 151
RTC_AlarmEnable ... 150
RTC_AlarmGet ... 155
RTC_AlarmInterruptCallback ... 156
RTC_AlarmSet ... 152
RTC_ChangeCorrectionValue ... 168
RTC_ConstPeriodInterruptCallback ... 149
RTC_ConstPeriodInterruptDisable ... 148
RTC_ConstPeriodInterruptEnable ... 147
RTC_CounterDisable ... 142
RTC_CounterEnable ... 141
RTC_CounterGet ... 146
RTC_CounterSet ... 144
RTC_Init ... 138
RTC_IntervalInterruptDisable ... 161
RTC_IntervalInterruptEnable ... 159
RTC_IntervalStart ... 157
RTC_IntervalStop ... 158
RTC_PowerOff ... 140
RTC_RTC1HZ_OutputDisable ... 163
RTC_RTC1HZ_OutputEnable ... 162
RTC_RTCCL_OutputDisable ... 165
RTC_RTCCL_OutputEnable ... 164
RTC_RTCDIV_OutputDisable ... 167
RTC_RTCDIV_OutputEnable ... 166
RTC_SetHourSystem ... 143
RTC_UserInit ... 139

S

Serial ... 54
CSI1n_Init ... 68
CSI1n_ReceiveData ... 72
CSI1n_ReceiveEndCallback ... 77
CSI1n_SendEndCallback ... 76
CSI1n_SendReceiveData ... 74
CSI1n_Start ... 70
CSI1n_Stop ... 71
CSI1n_UserInit ... 69
IICA_GetStopConditionCallback ... 92
IICA_Init ... 78
IICA_MasterErrorCallback ... 86
IICA_MasterReceiveEndCallback ... 85
IICA_MasterReceiveStart ... 82
IICA_MasterSendEndCallback ... 84
IICA_MasterSendStart ... 81
IICA_SlaveErrorCallback ... 91
IICA_SlaveReceiveEndCallback ... 90
IICA_SlaveReceiveStart ... 88
IICA_SlaveSendEndCallback ... 89
IICA_SlaveSendStart ... 87
IICA_Stop ... 80
IICA_StopCondition ... 83
IICA_UserInit ... 79
UART6_ErrorCallback ... 67
UART6_Init ... 56
UART6_ReceiveData ... 62
UART6_ReceiveEndCallback ... 65
UART6_SendData ... 60
UART6_SendEndCallback ... 64
UART6_SoftOverRunCallback ... 66
UART6_Start ... 58
UART6_Stop ... 59
UART6_UserInit ... 57
System ... 26
CG_ChangeClockMode ... 30
CG_ChangeFrequency ... 32
CG_ReadResetSource ... 29
CG_SelectPowerSaveMode ... 33
CG_SelectStabTime ... 35
CLOCK_Init ... 27
CLOCK_UserInit ... 28

T

Timer ... 111
TM00_ChangeDuty ... 119
TM00_ChangeTimerCondition ... 116
TM00_GetFreeRunningValue ... 117
TM00_GetPulseWidth ... 120
TM00_Init ... 112

TM00_SoftwareTriggerOn ... 118
TM00_Start ... 114
TM00_Stop ... 115
TM00_UserInit ... 113
TM5n_ChangeDuty ... 126
TM5n_ChangeTimerCondition ... 125
TM5n_Init ... 121
TM5n_Start ... 123
TM5n_Stop ... 124
TM5n_UserInit ... 122
TMH1_Carrieroutputdisable ... 134
TMH1_CarrierOutputEnable ... 133
TMHn_ChangeDuty ... 132
TMHn_ChangeTimerCondition ... 131
TMHn_Init ... 127
TMHn_Start ... 129
TMHn_Stop ... 130
TMHn_UserInit ... 128
TM00_ChangeDuty ... 119
TM00_ChangeTimerCondition ... 116
TM00_GetFreeRunningValue ... 117
TM00_GetPulseWidth ... 120
TM00_Init ... 112
TM00_SoftwareTriggerOn ... 118
TM00_Start ... 114
TM00_Stop ... 115
TM00_UserInit ... 113
TM5n_ChangeDuty ... 126
TM5n_ChangeTimerCondition ... 125
TM5n_Init ... 121
TM5n_Start ... 123
TM5n_Stop ... 124
TM5n_UserInit ... 122
TMH1_CarrierOutputDisable ... 134
TMH1_CarrierOutputEnable ... 133
TMHn_ChangeDuty ... 132
TMHn_ChangeTimerCondition ... 131
TMHn_Init ... 127
TMHn_Start ... 129
TMHn_Stop ... 130
TMHn_UserInit ... 128

U

UART6_ErrorCallback ... 67
UART6_Init ... 56
UART6_ReceiveData ... 62
UART6_ReceiveEndCallback ... 65
UART6_SendData ... 60
UART6_SendEndCallback ... 64
UART6_SoftOverRunCallback ... 66
UART6_Start ... 58
UART6_Stop ... 59
UART6_UserInit ... 57

W

Watchdog Timer ... 135
WDT_Restart ... 136
WDT_Restart ... 136

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/