NEC

User's Manual

Applilet3

Device Driver Configurator

API Reference

Target Device

78K0R Microcontroller

Document No. ZUD-CD-09-0242-01 (1st edition)
Date Published December 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

- The information in this document is current as of December, 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

LIST OF FIGURES

LIST OF TABLES

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	-
External Bus	CG_bus.c	BUS_Init BUS_PowerOff
	CG_bus_user.c	BUS_UserInit
	CG_bus.h	-
Port	CG_port.c	PORT_Init PORT_ChangePmnInput PORT_ChangePmnOutput
	CG_port_user.c	PORT_UserInit

Peripheral Function	File Name	Names of API Functions Included
Port	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	SAUm_Init SAUm_PowerOff UARTn_Init UARTn_Start UARTn_Stop UARTn_SendData UARTn_ReceiveData CSImn_Init CSImn_Start CSImn_Stop CSImn_SendData CSImn_ReceiveData CSImn_SendReceiveData IICmn_Init IICmn_Stop IICmn_MasterSendStart IICmn_MasterReceiveStart IICmn_StartCondition IICmn_StopCondition UARTFn_Init UARTFn_PowerOff UARTFn_Start UARTFn_Stop UARTFn_SendData UARTFn_ReceiveData UARTFn_SetComparisonData UARTFn_DataComparisonEnable UARTFn_DataComparisonDisable IICn_Init IICn_Stop IICn_MasterSendStart IICn_MasterReceiveStart IICn_SlaveSendStart IICn_SlaveReceiveStart IICA_Init IICA_PowerOff

Peripheral Function	File Name	Names of API Functions Included
Serial	CG_serial.c	IICA_Stop IICA_MasterSendStart IICA_MasterReceiveStart IICA_StopCondition IICA_SlaveSendStart IICA_SlaveReceiveStart
	CG_serial_user.c	MD_INTSR*n* MD_INTSRE*n* MD_INTST*n* MD_INTCSI*mn* MD_INTIIC*mn* MD_INTLT*n* MD_INTLR*n* MD_INTLS*n* MD_INTIIC*n* MD_INTIICA SAUm_UserInit UARTn_SendEndCallback UARTn_ReceiveEndCallback UARTn_SoftOverRunCallback UARTn_ErrorCallback CSImn_SendEndCallback CSImn_ReceiveEndCallback CSImn_ErrorCallback IICmn_MasterSendEndCallback IICmn_MasterReceiveEndCallback IICmn_MasterErrorCallback UARTFn_SendEndCallback UARTFn_ReceiveEndCallback UARTFn_SoftOverRunCallback UARTFn_ExpBitCetectCallback UARTFn_IDMatchCallback UARTFn_ErrorCallback IICn_UserInit IICn_MasterSendEndCallback IICn_MasterReceiveEndCallback IICn_MasterErrorCallback IICn_SlaveSendEndCallback IICn_SlaveReceiveEndCallback IICn_SlaveErrorCallback IICn_GetStopConditionCallback IICA_UserInit IICA_MasterSendEndCallback IICA_MasterReceiveEndCallback IICA_MasterErrorCallback IICA_SlaveSendEndCallback IICA_SlaveReceiveEndCallback IICA_SlaveErrorCallback IICA_GetStopConditionCallback
	CG_serial.h	-

Peripheral Function	File Name	Names of API Functions Included
A/D	CG_ad.c	AD_Init AD_PowerOff 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	-
D/A	CG_da.c	DA_Init DA_PowerOff DAn_Start DAn_Stop DAn_SetValue DAn_Set8BitsValue DAn_Set12BitsValue
	CG_da_user.c	DA_UserInit
	CG_da.h	-
LCD Controller/Driver	CG_lcd.c	LCD_Init LCD_DisplayOn LCD_DisplayOff LCD_VoltageOn LCD_VoltageOff
	CG_lcd_user.c	LCD_UserInit
	CG_lcd.h	-
Timer	CG_tau.c	TAUm_Init TAUm_PowerOff TAUm_Channeln_Start TAUm_Channeln_Stop TAUm_Channeln_ChangeCondition TAUm_Channeln_ChangeTimerCondition TAUm_Channeln_GetPulseWidth TAUm_Channeln_ChangeDuty TAUm_Channeln_SoftWareTriggerOn
	CG_tau_user.c	MD_INTTM*mn* TAUm_UserInit
	CG_tau.h	-
Watchdog Timer	CG_wdt.c	WDT_Init WDT_Restart
	CG_wdt_user.c	MD_INTWDTI WDT_UserInit
	CG_wdt.h	-
RTC	CG_rtc.c	RTC_Init

Peripheral Function	File Name	Names of API Functions Included
RTC	CG_rtc.c	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
	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	-
Clock Output/Buzzer Output	CG_pclbuz.c	PCLBUZn_Init PCLBUZn_Start PCLBUZn_Stop PCLBUZn_ChangeFreq
	CG_pclbuz_user.c	PCLBUZn_UserInit
	CG_pclbuz.h	-
DMA	CG_dma.c	DMAn_Init DMAn_Enable DMAn_Disable DMAn_Hold DMAn_Restart DMAn_CheckStatus DMAn_SetData

Peripheral Function	File Name	Names of API Functions Included
DMA	CG_dma.c	DMAn_SoftwareTriggerOn
	CG_dma_user.c	MD_INTDMA*n* DMAn_UserInit
	CG_dma.h	-
OPAMP	CG_opamp.c	OPAMP_Init AMPn_Start AMPn_Stop
	CG_opamp_user.c	OPAMP_UserInit
	CG_opamp.h	-
Comparator/PGA	CG_cmppga.c	CMPPGA_Init CMPPGA_PowerOff CMPPGA_Start CMPPGA_Stop CMPPGA_ChangeCMPnRefVoltage CMPPGA_ChangePGAFactor
	CG_cmppga_user.c	MD_INTCMP*n* CMPPGA_UserInit
	CG_cmppga.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.
External Bus	BUS_Init	Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).
	BUS_UserInit	Performs user-defined initialization relating to the external bus interface.
	BUS_PowerOff	Halts the clock supplied to the external bus interface.
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.

Peripheral Function	API Function Name	Function
INT	INTP_UserInit	Performs user-defined initialization relating to the external interrupt INTP*n* functions.
	KEY_Init	Performs initialization necessary to control the key interrupt INTKR functions.
	KEY_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
	INT_MaskableInterruptEnable	Disables/enables the acceptance of the maskable interrupts.
	INTPn_Disable	Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
	INTPn_Enable	Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
	KEY_Disable	Disables the acceptance of the key interrupts INTKR.
	KEY_Enable	Enables the acceptance of the key interrupts INTKR.
Serial	SAUm_Init	Performs initialization necessary to control the serial array unit and serial interface functions.
	SAUm_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
	SAUm_PowerOff	Halts the clock supplied to the serial array unit.
	UARTn_Init	Performs initialization of the serial interface (UART) channel.
	UARTn_Start	Sets UART communication to standby mode.
	UARTn_Stop	Ends UART communication.
	UARTn_SendData	Starts UART data transmission.
	UARTn_ReceiveData	Starts UART data reception.
	UARTn_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST*n*.
	UARTn_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR*n*.
	UARTn_SoftOverRunCallback	Performs processing in response to the UART reception complete interrupt INTSR*n*.
	UARTn_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE*n*.
	CSImn_Init	Performs initialization of the serial interface (CSI) channel.
	CSImn_Start	Sets CSI communication to standby mode.
	CSImn_Stop	Ends CSI communication.
	CSImn_SendData	Starts CSI data transmission.
	CSImn_ReceiveData	Starts CSI data reception.
	CSImn_SendReceiveData	Starts CSI data transmission/reception.
	CSImn_SendEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.
	CSImn_ReceiveEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.

Peripheral Function	API Function Name	Function
Serial	CSImn_ErrorCallback	Performs processing in response to the CSI communication error interrupt INTSRE*n*.
	IICmn_Init	Performs initialization of the serial interface (simple IIC) channel.
	IICmn_Stop	Ends simple IIC communication.
	IICmn_MasterSendStart	Starts simple IIC master transmission.
	IICmn_MasterReceiveStart	Starts simple IIC master reception.
	IICmn_StartCondition	Generates start conditions.
	IICmn_StopCondition	Generates stop conditions.
	IICmn_MasterSendEndCallback	Performs processing in response to the IIC*mn* communication complete interrupt INTIIC*mn*.
	IICmn_MasterReceiveEndCallback	Performs processing in response to the IIC*mn* communication complete interrupt INTIIC*mn*.
	IICmn_MasterErrorCallback	Performs processing in response to detection of parity error (ACK error) in simple IIC communication.
	UARTFn_Init	Performs initialization of the serial interface (UARTF*n*).
	UARTFn_PowerOff	Halts the clock supplied to the serial interface (UARTF*n*).
	UARTFn_Start	Sets UARTF communication to standby mode.
	UARTFn_Stop	Ends UARTF communication.
	UARTFn_SendData	Starts UARTF data transmission.
	UARTFn_ReceiveData	Starts UARTF data reception.
	UARTFn_SetComparisonData	Sets the data to compare to the received data.
	UARTFn_DataComparisonEnable	Starts the data comparison.
	UARTFn_DataComparisonDisable	Ends the data comparison.
	UARTFn_SendEndCallback	Performs processing in response to the transmission interrupt INTLT*n*.
	UARTFn_ReceiveEndCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
	UARTFn_SoftOverRunCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
	UARTFn_ExpBitCetectCallback	Performs processing in response to the status interrupt INTLS*n*.
	UARTFn_IDMatchCallback	Performs processing in response to the status interrupt INTLS*n*.
	UARTFn_ErrorCallback	Performs processing in response to the status interrupt INTLS*n*.
	IICn_Init	Performs initialization of the serial interface (IIC*n*).
	IICn_UserInit	Performs user-defined initialization of the serial interface (IIC*n*).
	IICn_Stop	Ends IIC*n* communication.
	IICn_MasterSendStart	Starts IIC*n* master transmission.
	IICn_MasterReceiveStart	Starts IIC*n* master reception.

Peripheral Function	API Function Name	Function
Serial	IICn_MasterSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_MasterReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_MasterErrorCallback	Performs processing in response to detection of error in IIC*n* master communication.
	IICn_SlaveSendStart	Starts IIC*n* slave transmission.
	IICn_SlaveReceiveStart	Starts IIC*n* slave reception.
	IICn_SlaveSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_SlaveReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_SlaveErrorCallback	Performs processing in response to detection of error in IIC*n* slave communication.
	IICn_GetStopConditionCallback	Performs processing in response to detection of stop condition in IIC*n* slave communication.
	IICA_Init	Performs initialization of the serial interface (IICA).
	IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
	IICA_PowerOff	Halts the clock supplied to 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 INTIICA.
	IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.
	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 INTIICA.
	IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.
	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.
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_PowerOff	Halts the clock supplied to the A/D converter.

Peripheral Function	API Function Name	Function
A/D	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.
	AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).
D/A	DA_Init	Performs initialization necessary to control D/A converter functions.
	DA_UserInit	Performs user-defined initialization relating to the D/A converter.
	DA_PowerOff	Halts the clock supplied to the D/A converter.
	DAn_Start	Starts D/A conversion.
	DAn_Stop	Ends D/A conversion.
	DAn_SetValue	Sets the initial analog voltage output to the ANO*n* pin.
	DAn_Set8BitsValue	Sets the initial analog voltage (8 bits) output to the ANO*n* pin.
	DAn_Set12BitsValue	Sets the initial analog voltage (12 bits) output to the ANO*n* pin.
LCD Controller/Driver	LCD_Init	Performs initialization necessary to control LCD controller/driver functions.
	LCD_UserInit	Performs user-defined initialization relating to the LCD controller/driver.
	LCD_DisplayOn	Sets the LCD controller/driver to "display on" status.
	LCD_DisplayOff	Sets the LCD controller/driver to "display off" status.
	LCD_VoltageOn	Eenables operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the deselect signal from the segment pin.
	LCD_VoltageOff	Halts operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the groundlevel signal from the segment/common pin.
Timer	TAUm_Init	Performs initialization necessary to control timer array unit functions.
	TAUm_UserInit	Performs user-defined initialization relating to the timer array unit.
	TAUm_PowerOff	Halts the clock supplied to the timer array unit.
	TAUm_Channeln_Start	Starts the count for channel *n*.
	TAUm_Channeln_Stop	Ends the count for channel *n*.
	TAUm_Channeln_ChangeCondition	Changes the counter value.
	TAUm_Channeln_ChangeTimerCondition	Changes the counter value.
	TAUm_Channeln_GetPulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.

Peripheral Function	API Function Name	Function
Timer	TAUm_ChannelN_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO*mn* pin.
	TAUm_ChannelN_SoftWareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
Watchdog Timer	WDT_Init	Performs initialization necessary to control watchdog timer functions.
	WDT_UserInit	Performs user-defined initialization relating to the 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).
	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.

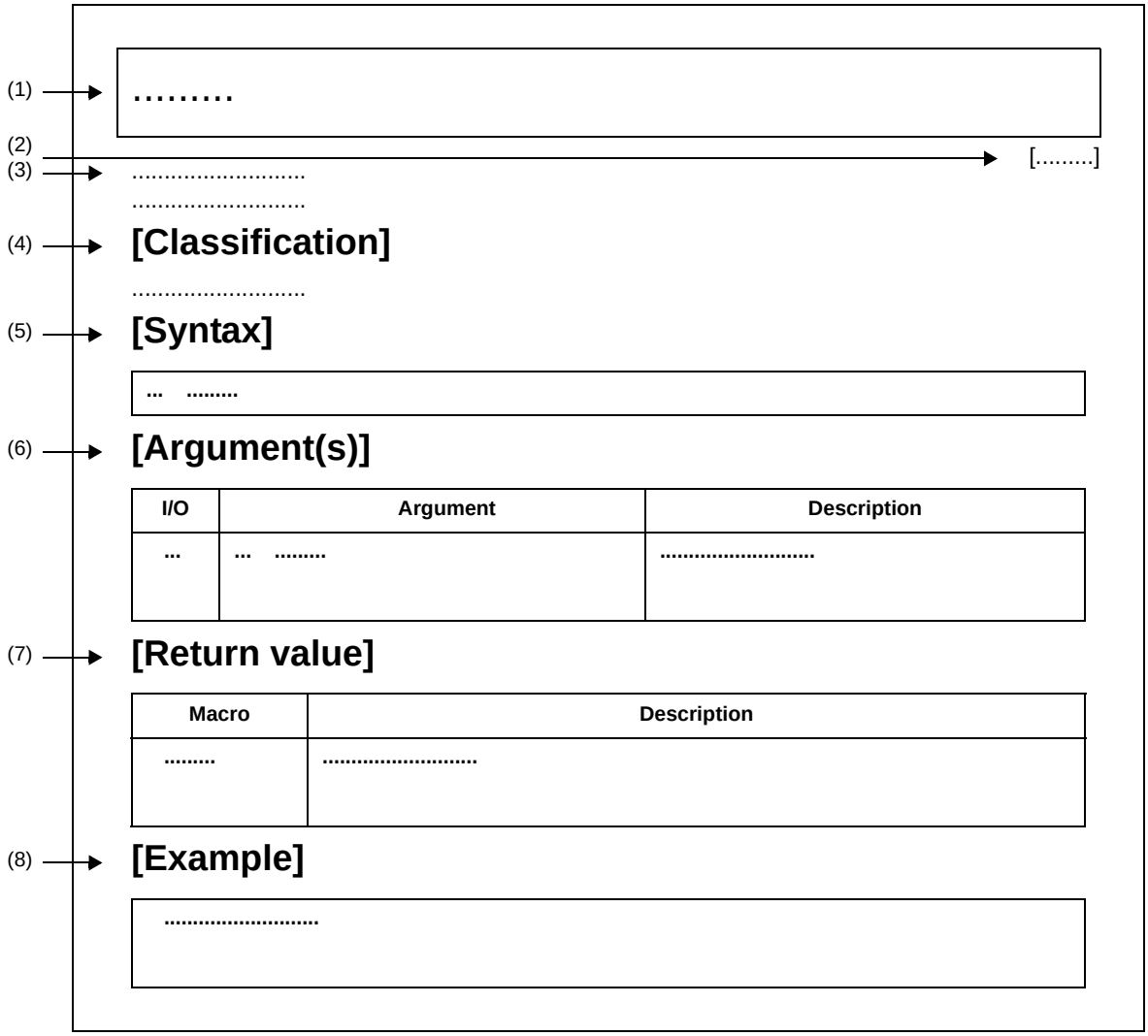
Peripheral Function	API Function Name	Function
RTC	RTC_RTCCL_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCCL_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCDIV_OutputEnable	Enables output of 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.
Clock Output/Buzzer Output	PCLBUZn_Init	Performs initialization necessary to control clock/buzzer output control circuit functions.
	PCLBUZn_UserInit	Performs user-defined initialization relating to the clock/buzzer output control circuits.
	PCLBUZn_Start	Starts clock/buzzer output.
	PCLBUZn_Stop	Ends clock/buzzer output.
	PCLBUZn_ChangeFreq	Changes the output clock to the PCLBUZ*n* pin.
DMA	DMAn_Init	Performs initialization necessary to control DMA controller functions.
	DMAn_UserInit	Performs user-defined initialization relating to the DMA controller.
	DMAn_Enable	Enables operation of channel *n*.
	DMAn_Disable	Disables operation of channel *n*.
	DMAn_Hold	Holds a DMA start request.
	DMAn_Restart	Releases hold on a DMA start request.
	DMAn_CheckStatus	Reads the transfer status (transfer complete/transfer ongoing).
	DMAn_SetData	Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.
	DMAn_SoftwareTriggerOn	Starts DMA transfer when DMA operation is enabled.
OPAMP	OPAMP_Init	Performs initialization necessary to control operational amplifier functions.
	OPAMP_UserInit	Performs user-defined initialization relating to the operational amplifier.
	AMPn_Start	Starts the operation of opeational amplifier *n* (single AMP mode).

Peripheral Function	API Function Name	Function
OPAMP	AMPn_Stop	Ends the operation of operational amplifier *n* (single AMP mode).
Comparator/PGA	CMPPGA_Init	Performs initialization necessary to control comparator/programmable gain amplifiers functions.
	CMPPGA_UserInit	Performs user-defined initialization relating to the comparator/programmable gain amplifiers.
	CMPPGA_PowerOff	Halts the clock supplied to the comparator/programmable gain amplifiers.
	CMPPGA_Start	Starts the operation of comparator/programmable gain amplifier.
	CMPPGA_Stop	Ends the operation of comparator/programmable gain amplifier.
	CMPPGA_ChangeCMPnRefVoltage	Sets comparator *n* internal reference voltage.
	CMPPGA_ChangePGAFactor	Sets the input voltage amplification factor of a programmable gain amplifier.
LVI	LVI_Init	Performs initialization necessary to control low-voltage detector functions.
	LVI_UserInit	Performs user-defined initialization relating to the low-voltage detector.
	LVI_InterruptModeStart	Starts low-voltage detection (when in interrupt generation mode).
	LVI_ResetModeStart	Starts low-voltage detection (when in internal reset mode).
	LVI_Stop	Stops low-voltage detection.
	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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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 WDRF flag */
        ......  /* Internal reset request by watchdog timer */
    } else if ( flag & 0x80 ) { /* Trigger identification: Check TRAP flag */
        ......  /* Internal reset request by execution of illegal instruction */
    }
    ......
}
```

CG_ChangeClockMode

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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;`	Clock generator type [Fx3] HIOCLK: Internal high-speed oscillation clock SYSX1CLK: X1 clock SYSEXTCLK: External main system clock FILCLK: Internal low-speed oscillation clock [Kx3] HIOCLK: Internal high-speed oscillation clock SYSX1CLK: X1 clock SYSEXTCLK: External main system clock SUBCLK: Subsystem clock [Kx3-A] [Kx3-L] [Lx3] HIOCLK: Internal high-speed oscillation clock HIO20CLK: 20 MHz internal high-speed oscillation clock SYSX1CLK: X1 clock SYSEXTCLK: External main system clock SUBCLK: Subsystem clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend) [Fx3] [Kx3] - Cannot change to the X1 clock. Exit with error (abend) [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the 20 MHz internal high-speed oscillation clock.
MD_ERROR2	Exit with error (abend) [Fx3] [Kx3] - Cannot change to the external main system clock. Exit with error (abend) [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the X1 clock.

Macro	Description
MD_ERROR3	Exit with error (abend) [Fx3] - Cannot change to the internal low-speed oscillation clock. Exit with error (abend) [Kx3] - Cannot change to the subsystem clock because the XT1 and XT2 pins are in input mode. Exit with error (abend) [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the external main system clock.
MD_ERROR4	Exit with error (abend) [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the subsystem clock.
MD_ARGERROR	Invalid argument specification

CG_ChangeFrequency

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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 [Fx3] SYSTEMCLOCK: fPLL SYSONEHALF: fPLL/2 SYSONEFOURTH: fPLL/4 SYSONEEIGHTH: fPLL/8 SYSONESIXTEENTH: fPLL/16 SYSONETHIRTYSECOND: fPLL/32 [Kx3] [Kx3-L] SYSTEMCLOCK: fMAIN SYSONEHALF: fMAIN/2 SYSONEFOURTH: fMAIN/4 SYSONEEIGHTH: fMAIN/8 SYSONESIXTEENTH: fMAIN/16 SYSONETHIRTYSECOND: fMAIN/32 [Kx3-A] [Lx3] SYSTEMCLOCK: fMAIN SYSONEHALF: fMAIN/2 SYSONEFOURTH: fMAIN/4 SYSONEEIGHTH: fMAIN/8 SYSONESIXTEENTH: fMAIN/16 SYSONETHIRTYSECOND: fMAIN/32 SUB: fSUB SUBONEHALF: fSUB/2

Remark "fPLL" signifies the frequency of the PLL clock, "fMAIN" signifies the frequency of the main system clock, and "fSUB" signifies the frequency of the subsystem clock.

[Return value]

Macro	Description
MD_OK	Normal completion

Macro	Description
MD_ARGERROR	Invalid argument specification

CG_SelectPowerSaveMode

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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 a subsystem clock (XT1 oscillator), then STOP mode cannot be specified.
MD_ARGERROR	Invalid argument specification

[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;
    ......
    TAU0_PowerOff ();                           /* Stop clock supply */
    ret = CG_SelectPowerSaveMode ( PSSTOP );    /* Change to STOP mode */
    if ( ret != MD_OK ) {
        while ( 1 );
    }
    TAU0_Init ();                               /* Initialize timer array unit */
    TAU0_Channel0_Start ();                     /* Starts the channel 0 count */
```

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

CG_SelectStabTime

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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: 2^8/fx STLEVEL1: 2^9/fx STLEVEL2: 2^10/fx STLEVEL3: 2^11/fx STLEVEL4: 2^13/fx STLEVEL5: 2^15/fx STLEVEL6: 2^17/fx STLEVEL7: 2^18/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 External Bus

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

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

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

BUS_Init

[Kx3]

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

[Classification]

CG_bus.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

BUS_UserInit

[Kx3]

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

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

[Classification]

CG_bus_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

BUS_PowerOff

[Kx3]

Halts the clock supplied to the external bus interface.

Remark Calling this API function changes the external bus interface to reset status. For this reason, writes to the control registers (memory extension mode control register: MEM) after this API function is called are ignored.

[Classification]

CG_bus.c

[Syntax]

```
void    BUS_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.3 Port

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

Table 3-4. API Functions: [Port]

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

PORT_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization necessary to control port functions.

[Classification]

CG_port.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PORT_UserInit

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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 TLL buffer.

- Built-in pull-up resistance: none; TLL buffer: none

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

- Built-in pull-up resistance: yes; TLL buffer: none

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

- Built-in pull-up resistance: yes; TLL buffer: yes

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

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 *enablettl*;	Input buffer type MD_TRUE: TTL 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; TLL 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; TLL 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; TLL buffer: yes) is changed as follows:

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Classification]

CG_port.c

[Syntax]

If the target device is a 78K0R/Fx3, then the format for specifying this API function differs depending on whether N-ch open-drain output is being performed via the target pin, and whether slow mode is specified.

- N-ch open drain output: none; Slow mode: none [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL initialvalue );
```

- N-ch open drain output: yes; Slow mode: none [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL enablench, BOOL initialvalue );
```

- N-ch open drain output: none; Slow mode: yes [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL enableslow, BOOL initialvalue );
```

- N-ch open drain output: yes; Slow mode: yes [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL enablench, BOOL enableslow, BOOL initialvalue );
```

If the target device is 78K0R/Kx3, 78K0R/Kx3-A, 78K0R/Kx3-L or 78K0R/Lx3, then 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 [Kx3] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL initialvalue );
```

- N-ch open drain output: yes [Kx3] [Kx3-A] [Kx3-L] [Lx3]

```
#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 *enableslow*;	Output mode type MD_TRUE: Slow mode MD_FALSE: Normal mode
I	BOOL *initialvalue*;	Initial output value MD_SET: Output HIGH level "1" MD_CLEAR: Output LOW level "0"

[Return value]

None.

[Example 1]

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

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

[CG_main.c]

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

[Example 2]

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

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

[CG_main.c]

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

3.3.4 INT

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

Table 3-5. API Functions: [INT]

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

INTP_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Disables/enables the acceptance of the maskable interrupts.

[Classification]

CG_int.c

[Syntax]

- [Fx3] [Kx3-A] [Kx3-L] [Lx3]

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

- [Kx3]

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

[Argument(s)]

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

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

[Return value]

- [Fx3] [Kx3-A] [Kx3-L] [Lx3]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

- [Kx3]
None.

[Example 1]

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

[CG_main.c]

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

[Example 2]

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

[CG_main.c]

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

INTP*n*_Disable

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-L] [Lx3]

Enables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

3.3.5 Serial

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

Table 3-6. API Functions: [Serial]

API Function Name	Function
SAUm_Init	Performs initialization necessary to control the serial array unit and serial interface functions.
SAUm_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
SAUm_PowerOff	Halts the clock supplied to the serial array unit.
UARTn_Init	Performs initialization of the serial interface (UART) channel.
UARTn_Start	Sets UART communication to standby mode.
UARTn_Stop	Ends UART communication.
UARTn_SendData	Starts UART data transmission.
UARTn_ReceiveData	Starts UART data reception.
UARTn_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST*n*.
UARTn_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR*n*.
UARTn_SoftOverRunCallback	Performs processing in response to the serial transfer end interrupt INTSR*n*.
UARTn_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE*n*.
CSImn_Init	Performs initialization of the serial interface (CSI) channel.
CSImn_Start	Sets CSI communication to standby mode.
CSImn_Stop	Ends CSI communication.
CSImn_SendData	Starts CSI data transmission.
CSImn_ReceiveData	Starts CSI data reception.
CSImn_SendReceiveData	Starts CSI data transmission/reception.
CSImn_SendEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.
CSImn_ReceiveEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.
CSImn_ErrorCallback	Performs processing in response to the CSI communication error interrupt INTSRE*n*.
IICmn_Init	Performs initialization of the serial interface (simple IIC) channel.
IICmn_Stop	Ends simple IIC communication.
IICmn_MasterSendStart	Starts simple IIC master transmission.
IICmn_MasterReceiveStart	Starts simple IIC master reception.
IICmn_StartCondition	Generates start conditions.
IICmn_StopCondition	Generates stop conditions.
IICmn_MasterSendEndCallback	Performs processing in response to the simple IIC*mn* communication complete interrupt INTIIC*mn*.
IICmn_MasterReceiveEndCallback	Performs processing in response to the simple IIC*mn* communication complete interrupt INTIIC*mn*.

API Function Name	Function
IICmn_MasterErrorCallback	Performs processing in response to detection of parity error (ACK error) in simple IIC communication.
UARTFn_Init	Performs initialization of the serial interface (UARTF*n*).
UARTFn_PowerOff	Halts the clock supplied to the serial interface (UARTF*n*).
UARTFn_Start	Sets UARTF communication to standby mode.
UARTFn_Stop	Ends UARTF communication.
UARTFn_SendData	Starts UARTF data transmission.
UARTFn_ReceiveData	Starts UARTF data reception.
UARTFn_SetComparisonData	Sets the data to compare to the received data.
UARTFn_DataComparisonEnable	Starts the data comparison.
UARTFn_DataComparisonDisable	Ends the data comparison.
UARTFn_SendEndCallback	Performs processing in response to the transmission interrupt INTLT*n*.
UARTFn_ReceiveEndCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
UARTFn_SoftOverRunCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
UARTFn_ExpBitCetectCallback	Performs processing in response to the status interrupt INTLS*n*.
UARTFn_IDMatchCallback	Performs processing in response to the status interrupt INTLS*n*.
UARTFn_ErrorCallback	Performs processing in response to the status interrupt INTLS*n*.
IICn_Init	Performs initialization of the serial interface (IIC*n*).
IICn_UserInit	Performs user-defined initialization of the serial interface (IIC*n*).
IICn_Stop	Ends IIC*n* communication.
IICn_MasterSendStart	Starts IIC*n* master transmission.
IICn_MasterReceiveStart	Starts IIC*n* master reception.
IICn_MasterSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_MasterReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_MasterErrorCallback	Performs processing in response to detection of error in IIC*n* master communication.
IICn_SlaveSendStart	Starts IIC*n* slave transmission.
IICn_SlaveReceiveStart	Starts IIC*n* slave reception.
IICn_SlaveSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_SlaveReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_SlaveErrorCallback	Performs processing in response to detection of error in IIC*n* slave communication.
IICn_GetStopConditionCallback	Performs processing in response to detection of stop condition in IIC*n* slave communication.
IICA_Init	Performs initialization of the serial interface (IICA).
IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
IICA_PowerOff	Halts the clock supplied to the serial interface (IICA).
IICA_Stop	Ends IICA communication.

API Function Name	Function
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 INTIICA.
IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.
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 INTIICA.
IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.
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.

SAU*m*_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization necessary to control the serial array unit and serial interface functions.

[Classification]

CG_serial.c

[Syntax]

```
void    SAUm_Init ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

SAU*m*_UserInit

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs user-defined initialization related to the serial array unit and serial interface functions.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    SAUm_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

SAU*m*_PowerOff

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Halts the clock supplied to the serial array unit.

Remark Calling this API function changes the serial array unit to reset status. For this reason, writes to the control registers (e.g. serial clock select register *n*: SPS*n*) after this API function is called are ignored.

[Classification]

CG_serial.c

[Syntax]

```
void    SAUm_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization of the serial interface (UART) channel.

Remark This API function is used as an internal function of SAUm_Init. For this reason, there is normally no need to call it from a user program.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_Start

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Sets UART communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_Stop

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends UART communication.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_SendData

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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, UARTn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending a UART transmission of four bytes of fixed-length data from channel 0 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 */
    ......
    UART0_Start ();                     /* Start UART communication*/
    UART0_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_INTST0 ( void ) {   /* Interrupt processing for INTST0 */
    if ( gUart0TxCnt > 0 ) {
        ......
    } else {
        UART0_SendEndCallback ();       /* Call callback routine */
    }
}

void UART0_SendEndCallback ( void ) {   /* Callback routine for INTST0 */
    gFlag = 0;                          /* Set transmission complete flag */
}
```

UART*n*_ReceiveData

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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 UARTn_Start is then called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTn_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 UART reception of four bytes of fixed-length data from channel 0 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 */
    ......
    UART0_ReceiveData ( &rxbuf, rxnum );    /* Start UART data reception */
    UART0_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_INTSR0 ( void ) {       /* Interrupt processing for INTSR0 */
    ......
    if ( gUart0RxLen > gUart0RxCnt ) {
        ......
        if ( gUart0RxLen == gUart0RxCnt ) {
            UART0_ReceiveEndCallback ();    /* Call callback routine */
        }
    }
}

void UART0_ReceiveEndCallback ( void ) {    /* Callback routine for INTSR0 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

UART*n*_SendEndCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the UART transmission complete interrupt INTST*n*.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_ReceiveEndCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the UART reception complete interrupt INTSR*n*.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_SoftOverRunCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the UART reception complete interrupt INTSR*n*.

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

[Classification]

CG_serial_user.c

[Syntax]

- [Fx3]

```
void    UARTn_SoftOverRunCallback ( void );
```

- [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

Remark *n* is the channel number.

[Argument(s)]

- [Fx3]

None.

- [Kx3] [Kx3-A] [Kx3-L] [Lx3]

I/O	Argument	Description
O	`UCHAR  rx_data;`	Receive data (greater than the parameter *rxnum* specified for UARTn_ReceiveData)

[Return value]

None.

UART*n*_ErrorCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the UART communication error interrupt INTSRE*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTSRE*n* corresponding to the UART communication error interrupt INTSRE*n*.

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	UCHAR *err_type*;	Trigger for error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Parity error 000001*xx*B: Framing 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_INTSRE0 ( void ) {          /* Interrupt processing for INTSRE0 */
    UCHAR   err_type;
    ......
    UART0_ErrorCallback ( err_type );           /* Call callback routine */
}

void UART0_ErrorCallback ( UCHAR err_type ) {   /* Callback routine for INTSRE0 */
    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 parity error */
    } else if ( err_type & 0x4 ) {              /* Determine trigger */
        ......  /* Callback processing in response to framing error */
```

```
    }
}
```

CSI*mn*_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization of the serial interface (CSI) channel.

Remark This API function is used as an internal function of SAUm_Init. For this reason, there is normally no need to call it from a user program.

[Classification]

CG_serial.c

[Syntax]

```
void    CSImn_Init ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_Start

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Sets CSI communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    CSImn_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_Stop

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends CSI communication.

[Classification]

CG_serial.c

[Syntax]

```
void    CSImn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_SendData

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts CSI data transmission.

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

2. When performing a CSI transmission, CSImn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending a CSI transmission of four bytes of fixed-length data from channel 00 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 */
    ......
    CSI00_Start ();                     /* Start CSI communication */
    CSI00_SendData ( &txbuf, txnum );   /* Start CSI transmission */
    while ( gFlag );                    /* Wait for txnum transmissions */
    ......
```

```
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                    /* Transmission complete flag */
__interrupt void MD_INTCSI00 ( void ) {   /* Interrupt processing for INTCSI00 */
    if ( gCsi00TxCnt > 0 ) {
        ......
    } else {
        CSI00_SendEndCallback ();         /* Call callback routine */
    }
}

void CSI00_SendEndCallback ( void ) {     /* Callback routine for INTCSI00 */
    gFlag = 0;                            /* Set transmission complete flag */
}
```

CSI*mn*_ReceiveData

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts CSI data reception.

Remarks 1. This API function performs byte-level CSI 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 CSI reception, CSImn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *m* is the unit number, and *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 CSI transmission of four bytes of fixed-length data from channel 00 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 */
    ......
    CSI00_Start ();                         /* Start CSI communication */
    CSI00_ReceiveData ( &rxbuf, rxnum );    /* Start CSI reception */
    while ( gFlag );                        /* Wait for rxnum receptions */
    ......
```

```
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Reception complete flag */
__interrupt void MD_INTCSI00 ( void ) {     /* Interrupt processing for INTCSI00 */
    if ( gCsi00RxCnt < gCsi00RxLen ) {
        ......
        if ( gCsi00RxCnt == gCsi00RxLen ) {
            CSI00_ReceiveEndCallback ();    /* Call callback routine */
        } else {
            ......
        }
    }
}

void CSI00_ReceiveEndCallback ( void ) {    /* Callback routine for INTCSI00 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

CSI*mn*_SendReceiveData

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts CSI data transmission/reception.

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

2. This API function performs byte-level CSI 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 CSI reception, CSImn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *m* is the unit number, and *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 CSI transmission of four bytes of fixed-length data from channel 00 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 */
```

```
    ......
    CSI00_Start ();                                     /* Start CSI communication */
    CSI00_SendReceiveData ( &txbuf, txnum, &rxbuf );    /* Start CSI 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_INTCSI00 ( void ) {     /* Interrupt processing for INTCSI00 */
   if ( gCsi00TxCnt > 0 ) {
        ......
    } else {
        ......
        CSI00_SendEndCallback ();           /* Call callback routine */
    }
    ......
}

void CSI00_SendEndCallback ( void ) {       /* Callback routine for INTCSI00 */
    gSflag = 0;                             /* Set transmission complete flag */
}
```

CSI*mn*_SendEndCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.

Remark This API function is called as the callback routine of interrupt process MD_INTCSI*mn* corresponding to the CSI communication complete interrupt INTCSI*mn* (performed when number of transmission data specified by CSImn_SendData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSImn_SendEndCallback ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_ReceiveEndCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.

Remark This API function is called as the callback routine of interrupt process MD_INTCSI*mn* corresponding to the CSI communication complete interrupt INTCSI*mn* (performed when number of received data specified by CSImn_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSImn_ReceiveEndCallback ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_ErrorCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the CSI communication error interrupt INTSRE*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTSRE*n* corresponding to the UART communication error interrupt INTSRE*n*.

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	UCHAR *err_type*;	Trigger for error interrupt 00000*xx*1B: Overrun error

[Return value]

None.

[Example]

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

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE0 ( void ) {          /* Interrupt processing for INTSRE0 */
    UCHAR   err_type;
    ......
    CSI00_ErrorCallback ( err_type );           /* Call callback routine */
}

void CSI00_ErrorCallback ( UCHAR err_type ) {   /* Callback routine for INTSRE0 */
    if ( err_type & 0x1 ) {                     /* Determine trigger */
        ......  /* Callback processing in response to overrun error */
    }
}
```

IIC*mn*_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization of the serial interface (simple IIC) channel.

Remark This API function is used as an internal function of SAUm_Init. For this reason, there is normally no need to call it from a user program.

[Classification]

CG_serial.c

[Syntax]

```
void    IICmn_Init ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_Stop

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends simple IIC communication.

[Classification]

CG_serial.c

[Syntax]

```
void    IICmn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterSendStart

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts simple IIC master transmission.

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

[Classification]

CG_serial.c

[Syntax]

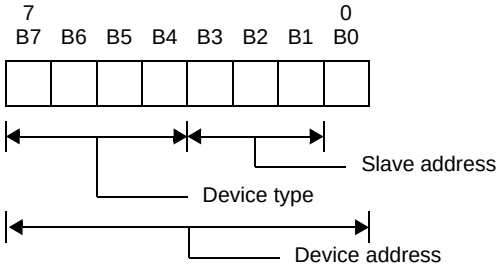
```
#include    "CG_macrodriver.h"
void    IICmn_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

IIC*mn*_MasterReceiveStart

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts simple IIC master reception.

Remark This API function performs byte-level simple IIC 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]

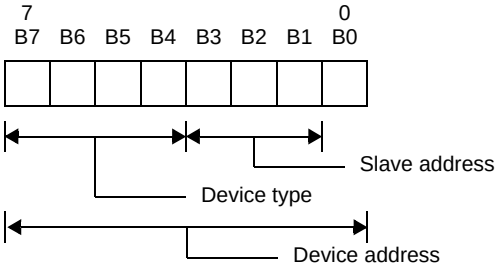
```
#include    "CG_macrodriver.h"
void    IIC mn _MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

IIC*mn*_StartCondition

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Generates start conditions.

Remark This API function is used as an internal function of IICmn_MasterSendStart and IICmn_MasterReceiveStart. For this reason, there is normally no need to call it from a user program.

[Classification]

CG_serial.c

[Syntax]

```
void    IICmn_StartCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_StopCondition

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Generates stop conditions.

[Classification]

CG_serial.c

[Syntax]

```
void    IIC*mn*_StopCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterSendEndCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the simple IIC*mn* communication complete interrupt INTIIC*mn*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*mn* corresponding to the simple IIC*mn* communication complete interrupt INTIIC*mn* (performed when number of transmission data specified by IICmn_MasterSendStart parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    IIC mn _MasterSendEndCallback ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterReceiveEndCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to the simple IIC*mn* communication complete interrupt INTIIC*mn*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*mn* corresponding to the simple IIC*mn* communication complete interrupt INTIIC*mn* (performed when number of received data specified by IICmn_MasterReceiveStart parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICmn_MasterReceiveEndCallback ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterErrorCallback

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs processing in response to detection of parity error (ACK error) in simple IIC communication.

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`MD_STATUS flag;`	Cause of communication error MD_NACK: Acknowledge not detected

[Return value]

None.

UARTF*n*_Init

[Fx3]

Performs initialization of the serial interface (UARTF*n*).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_PowerOff

[Fx3]

Halts the clock supplied to the serial interface (UARTFn).

Remark Calling this API function changes the serial interface (UARTF*n*) to reset status. For this reason, writes to the control registers (e.g. LIN-UARTn status register: UF*n*STR) after this API function is called are ignored.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_Start

[Fx3]

Sets UARTF communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_Stop

[Fx3]

Ends UARTF communication.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_SendData

[Fx3]

Starts UARTF data transmission.

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

2. When performing a UARTF transmission, UARTFn_Start must be called before this API function is called.

3. If the serial interface (UARTF*n*) is used in expansion bit mode, then store the data to send in the buffer specified by parameter *txbuf*, in the following format.

"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification
MD_DATAEXISTS	Executing transmission process

UARTF*n*_ReceiveData

[Fx3]

Starts UARTF data reception.

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

2. Actual UARTF reception starts after this API function is called, and UARTFn_Start is then called.

3. If the serial interface (UARTF*n*) is used in expansion bit mode, then the received data is stored in the buffer specified by parameter *rxbuf*, in the following format.

"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Classification]

CG_serial.c

[Syntax]

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

UARTF*n*_SetComparisonData

[Fx3]

Sets the data to compare to the received data.

Remark The value specified in parameter *comdata* is set to LIN-UART*n* ID setting register (UF*n*ID).

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTFn_SetComparisonData ( UCHAR comdata );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *comdata*;	Data to compare

[Return value]

None.

UARTF*n*_DataComparisonEnable

[Fx3]

Starts the data comparison.

Remark Calling this API function switches the serial interface (UARTF*n*) to expansion bit mode (with data comparison).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_DataComparisonEnable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_DataComparisonDisable

[Fx3]

Ends the data comparison.

Remark Calling this API function switches the serial interface (UARTF*n*) to expansion bit mode (no data comparison).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_DataComparisonDisable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_SendEndCallback

[Fx3]

Performs processing in response to the transmission interrupt INTLT*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLT*n* corresponding to the transmission interrupt INTLT*n* (performed when number of transmission data specified by UARTFn_SendData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_ReceiveEndCallback

[Fx3]

Performs processing in response to the reception complete interrupt INTLR*n*.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_SoftOverRunCallback

[Fx3]

Performs processing in response to the reception complete interrupt INTLR*n*.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_SoftOverRunCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_ExpBitCetectCallback

[Fx3]

Performs processing in response to the status interrupt INTLS*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLS*n* corresponding to the status interrupt INTLS*n* (processing when the expansion bit is received).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_ExpBitCetectCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_IDMatchCallback

[Fx3]

Performs processing in response to the status interrupt INTLS*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLS*n* corresponding to the status interrupt INTLS*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_IDMatchCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_ErrorCallback

[Fx3]

Performs processing in response to the status interrupt INTLS*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLS*n* corresponding to the status interrupt INTLS*n*.

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

IIC*n*_Init

[Kx3]

Performs initialization of the serial interface (IIC*n*).

[Classification]

CG_serial.c

[Syntax]

```
void    IICn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_UserInit

[Kx3]

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_Stop

[Kx3]

Ends IIC*n* communication.

[Classification]

CG_serial.c

[Syntax]

```
void    IICn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_MasterSendStart

[Kx3]

Starts IIC*n* master transmission.

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

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[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_ERROR	Exit with error (abend)

IIC*n*_MasterReceiveStart

[Kx3]

Starts IIC*n* master reception.

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

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[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_ERROR	Exit with error (abend)

IIC*n*_MasterSendEndCallback

[Kx3]

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_MasterSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_MasterReceiveEndCallback

[Kx3]

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_MasterReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_MasterErrorCallback

[Kx3]

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

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[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.

IIC*n*_SlaveSendStart

[Kx3]

Starts IIC*n* slave transmission.

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

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

IIC*n*_SlaveReceiveStart

[Kx3]

Starts IIC*n* slave reception.

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

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

IIC*n*_SlaveSendEndCallback

[Kx3]

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_SlaveSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_SlaveReceiveEndCallback

[Kx3]

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_SlaveReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_SlaveErrorCallback

[Kx3]

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

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[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.

IIC*n*_GetStopConditionCallback

[Kx3]

Performs processing in response to detection of stop condition in IIC*n* slave communication.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_GetStopConditionCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IICA_Init

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

Performs initialization of the serial interface (IICA).

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_UserInit

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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_PowerOff

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

Halts the clock supplied to the serial interface (IICA).

Remark Calling this API function changes the serial interface (IICA) to reset status. For this reason, writes to the control registers (e.g. IICA control register *n*: IICCTL*n*) after this API function is called are ignored.

[Classification]

CG_serial.c

[Syntax]

```
void    IICA_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_Stop

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

Ends IICA communication.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendStart

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

Generates stop conditions.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendEndCallback

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

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

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterReceiveEndCallback

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

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

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterErrorCallback

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

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

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveReceiveEndCallback

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

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

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveErrorCallback

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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

[Kx3-A] [Kx3-L] [Lx3 (excluding LF3)]

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.

3.3.6 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_PowerOff	Halts the clock supplied 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.
AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

AD_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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_PowerOff

[Fx3] [Kx3] [Kx3-L]

Halts the clock supplied to the A/D converter.

Remark Calling this API function changes the A/D converter to reset status. For this reason, writes to the control registers (e.g. A/D converter mode register: ADCM) after this API function is called are ignored.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_ComparatorOn

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Enables operation of voltage converter.

Remarks 1. 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.

2. On the [A/D], in the [Comparator operation setting] area, if "Operation" is selected, then the voltage converter will be switched to "always on". There is thus no need to call this API function in this case.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_ComparatorOff

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Disables operation of voltage converter.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_Start

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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.

[Example 1]

Below is an example of starting analog voltage via the conversion start pin selected in the [A/D], then converting the analog voltage from the ANI1 input pin to digital. In the example, "Stop" is selected in the [Comparator operation setting] area of the [A/D] (when performing the call to AD_ComparatorOn).

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D conversion complete flag */
void main ( void ) {
    USHORT  buffer = 0;
    int     wait = 100;
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    ......
    AD_ComparatorOn ();                 /* Move to operation enabled status */
    while ( wait );             /* Ensure stabilization time (at least 1 micro second) */
    AD_Start ();                        /* Start A/D conversion */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_SelectADChannel ( ADCHANNEL1 );  /* Switch input pins */
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_SelectADChannel ( ADCHANNEL1 );  /* Switch input pins */
```

```
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_Stop ();                         /* End A/D conversion */
    AD_ComparatorOff ();                /* Move to operation disabled status */
    ......
}
```

[CG_ad_user.c]

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

[Example 2]

Below is an example of starting analog voltage via the conversion start pin selected in the [A/D], then converting the analog voltage from the ANI1 input pin to digital. In the example, "Operation" is selected in the [Comparator operation setting] area of the [A/D] (when not performing the call to AD_ComparatorOn).

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D conversion complete flag */
void main ( void ) {
    USHORT  buffer = 0;
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    ......
    AD_Start ();                        /* Start A/D conversion */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_SelectADChannel ( ADCHANNEL1 );  /* Switch input pins */
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_Stop ();                         /* End A/D conversion */
    ......
}
```

[CG_ad_user.c]

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

AD_Stop

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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]

- [Fx3] [Kx3-A] [Kx3-L] [Lx3]

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

- [Kx3]

```
#include    "CG_ad.h"
void    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]

- [Fx3] [Kx3-A] [Kx3-L] [Lx3]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

- [Kx3]

None.

AD_Read

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Reads the results of A/D conversion.

Remark If the target device is 78K0R/Fx3, 78K0R/Kx3, 78K0R/Kx3-L, then the results of A/D conversion will be 10 bits. If the target device is 78K0R/Kx3-A, 78K0R/Lx3, then the results of A/D conversion will be 12 bits.

[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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Classification]

CG_ad.c

[Syntax]

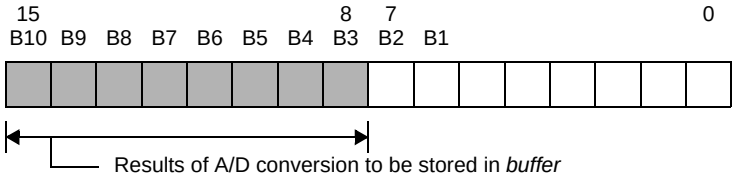
```
#include    "CG_macrodriver.h"
void    AD_ReadByte ( UCHAR *buffer );
```

[Argument(s)]

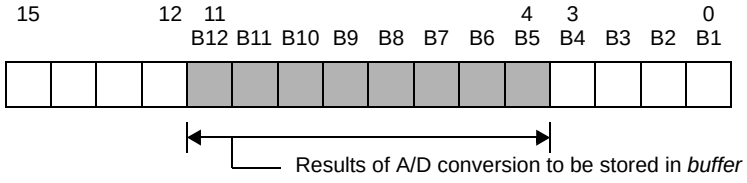
I/O	Argument	Description
O	UCHAR *buffer;	Pointer to area in which to store the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution)

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

- [Fx3] [Kx3] [Kx3-L]



- [Kx3-A] [Lx3]



[Return value]

None.

3.3.7 D/A

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

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

API Function Name	Function
DA_Init	Performs initialization necessary to control D/A converter functions.
DA_UserInit	Performs user-defined initialization relating to the D/A converter.
DA_PowerOff	Halts the clock supplied to the D/A converter.
DAn_Start	Starts D/A conversion.
DAn_Stop	Ends D/A conversion.
DAn_SetValue	Sets the initial analog voltage output to the ANO*n* pin.
DAn_Set8BitsValue	Sets the initial analog voltage (8 bits) output to the ANO*n* pin.
DAn_Set12BitsValue	Sets the initial analog voltage (12 bits) output to the ANO*n* pin.

DA_Init

[Kx3] [Kx3-A] [Lx3]

Performs initialization necessary to control D/A converter functions.

[Classification]

CG_da.c

[Syntax]

```
void    DA_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

DA_UserInit

[Kx3] [Kx3-A] [Lx3]

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

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

[Classification]

CG_da_user.c

[Syntax]

```
void    DA_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

DA_PowerOff

[Kx3] [Kx3-A] [Lx3]

Halts the clock supplied to the D/A converter.

Remark Calling this API function changes the D/A converter to reset status. For this reason, writes to the control registers (e.g. D/A converter mode register: DAM) after this API function is called are ignored.

[Classification]

CG_da.c

[Syntax]

```
void    DA_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

DA*n*_Start

[Kx3] [Kx3-A] [Lx3]

Starts D/A conversion.

[Classification]

CG_da.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_Stop

[Kx3] [Kx3-A] [Lx3]

Ends D/A conversion.

[Classification]

CG_da.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_SetValue

[Kx3]

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

[Classification]

CG_da.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

[Example]

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

[CG_main.c]

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

[CG_tau_user.c]

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

DA*n*_Set8BitsValue

[Kx3-A] [Lx3]

Sets the analog voltage (8 bits) output to the ANO*n* pin.

[Classification]

CG_da.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    DAn_Set8BitsValue ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *value*;	Analog voltage (0x0 to 0x1f)

[Return value]

None.

[Example]

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

[CG_main.c]

```
void main ( void ) {
    ......
    DA0_Start ();                       /* Start D/A conversion */
    DA1_Start ();                       /* Start D/A conversion */
    ......
    DA0_Set8BitsValue ( 0x7f );         /* Set analog voltage */
    ......
}
```

[CG_tau_user.c]

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* Interrupt processing for INTTM05 */
    DA1_Set8BitsValue ( gValue++ );     /* Set analog voltage */
}
```

DA*n*_Set12BitsValue

[Kx3-A] [Lx3]

Sets the analog voltage (12 bits) output to the ANO*n* pin.

[Classification]

CG_da.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    DAn_Set12BitsValue ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

[Example]

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

[CG_main.c]

```
void main ( void ) {
    ......
    DA0_Start ();                       /* Start D/A conversion */
    DA1_Start ();                       /* Start D/A conversion */
    ......
    DA0_Set12BitsValue ( 0x1ff );       /* Set analog voltage */
    ......
}
```

[CG_tau_user.c]

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* Interrupt processing for INTTM05 */
    DA1_Set12BitsValue ( gValue++ );    /* Set analog voltage */
}
```

3.3.8 LCD Controller/Driver

Below is a list of API functions output by Applilet3 for LCD controller/driver use.

Table 3-9. API Functions: [LCD Controller/Driver]

API Function Name	Function
LCD_Init	Performs initialization necessary to control LCD controller/driver functions.
LCD_UserInit	Performs user-defined initialization relating to the LCD controller/driver.
LCD_DisplayOn	Sets the LCD controller/driver to "display on" status.
LCD_DisplayOff	Sets the LCD controller/driver to "display off" status.
LCD_VoltageOn	Eenables operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the deselect signal from the segment pin.
LCD_VoltageOff	Halts operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the groundlevel signal from the segment/common pin.

LCD_Init

[Lx3]

Performs initialization necessary to control LCD controller/driver functions.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_UserInit

[Lx3]

Performs user-defined initialization relating to the LCD controller/driver.

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

[Classification]

CG_lcd_user.c

[Syntax]

```
void    LCD_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_DisplayOn

[Lx3]

Sets the LCD controller/driver to "display on" status.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_DisplayOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_DisplayOff

[Lx3]

Sets the LCD controller/driver to "display off" status.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_DisplayOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_VoltageOn

[Lx3]

Eenables operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the deselect signal from the segment pin.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_VoltageOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_VoltageOff

[Lx3]

Halts operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the ground-level signal from the segment/common pin.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_VoltageOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.9 Timer

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

Table 3-10. API Functions: [Timer]

API Function Name	Function
TAUm_Init	Performs initialization necessary to control timer array unit functions.
TAUm_UserInit	Performs user-defined initialization relating to the timer array unit.
TAUm_PowerOff	Halts the clock supplied to the timer array unit.
TAUm_Channeln_Start	Starts the count for channel *n*.
TAUm_Channeln_Stop	Ends the count for channel *n*.
TAUm_Channeln_ChangeCondition	Changes the counter value.
TAUm_Channeln_ChangeTimerCondition	Changes the counter value.
TAUm_Channeln_GetPulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.
TAUm_Channeln_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO*mn* pin.
TAUm_Channeln_SoftWareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.

TAU*m*_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization necessary to control timer array unit functions.

[Classification]

CG_tau.c

[Syntax]

```
void    TAUm_Init ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

TAU*m*_UserInit

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs user-defined initialization relating to the timer array unit.

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

[Classification]

CG_tau_user.c

[Syntax]

```
void    TAUm_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

TAU*m*_PowerOff

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Halts the clock supplied to the timer array unit.

Remark Calling this API function changes the timer array unit to reset status. For this reason, writes to the control registers (e.g. timer clock select register 0: TPS0) after this API function is called are ignored.

[Classification]

CG_tau.c

[Syntax]

```
void    TAUm_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

TAU*m*_Channel*n*_Start

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts the count for channel *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 external event counter).

[Classification]

CG_tau.c

[Syntax]

```
void    TAUm_Channeln_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAU*m*_Channel*n*_Stop

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends the count for channel *n*.

[Classification]

CG_tau.c

[Syntax]

```
void    TAUm_Channeln_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAU*m*_Channel*n*_ChangeCondition

[Kx3]

Changes the counter value.

Remarks 1. The value specified in parameter *regvalue* is set to timer data register *mn* (TDR*mn*).

2. The timing for calling these API functions differs as follows, depending on the type of function (e.g. interval timer, square wave output, or external event counter).

Function Type	Timing for Calling
Interval timer	Can call at user discretion.
Square wave output	Can call at user discretion.
Divider function	Can call at user discretion.
External event counter	Can call at user discretion.
Input pulse interval measurement	Cannot be called.
Input pulse high-/low-level width measurement	Cannot be called.
PWM output	Cannot be called.
One-shot pulse output	Cannot be called during operation.
Multiple PWM output	Cannot be called.

[Classification]

CG_tau.c

[Syntax]

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

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

[Example]

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

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
```

```
    USHORT  value = TAU_TDR00_VALUE >> 1;      /* TAU_TDR00_VALUE: Current interval time */
    ......
    TAU0_Channel0_Start ();                      /* Start count */
    ......
    TAU0_Channel0_ChangeCondition ( value );     /* Change counter value */
   ......
}
```

TAU*m*_Channel*n*_ChangeTimerCondition

[Fx3] [Kx3-A] [Kx3-L] [Lx3]

Changes the counter value.

Remarks 1. The value specified in parameter *regvalue* is set to timer data register *mn* (TDR*mn*).

2. The timing for calling these API functions differs as follows, depending on the type of function (e.g. interval timer, square wave output, or external event counter).

Function Type	Timing for Calling
Interval timer	Can call at user discretion.
Square wave output	Can call at user discretion.
Divider function	Can call at user discretion.
External event counter	Can call at user discretion.
Input pulse interval measurement	Cannot be called.
Input pulse high-/low-level width measurement	Cannot be called.
PWM output	Cannot be called.
One-shot pulse output	Cannot be called during operation.
Multiple PWM output	Cannot be called.

[Classification]

CG_tau.c

[Syntax]

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

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

[Example]

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

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
```

```
    USHORT  value = TAU_TDR00_VALUE >> 1;      /* TAU_TDR00_VALUE: Current interval time */
    ......
    TAU0_Channel0_Start ();                        /* Start count */
    ......
    TAU0_Channel0_ChangeTimerCondition ( value );  /* Change counter value */
   ......
}
```

TAU*m*_Channel*n*_GetPulseWidth

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.

[Classification]

CG_tau.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_GetPulseWidth ( ULONG *width );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`ULONG *width;`	Pointer to an area to store the measurement width (0x0 to 0x1ffff)

[Return value]

None.

TAU*m*_Channel*n*_ChangeDuty

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Changes the duty ratio of the PWM signal output to the TO*mn* pin.

Remark The timing for calling these API functions differs as follows, depending on the type of function (e.g. interval timer, square wave output, or external event counter).

Function Type	Timing for Calling
Interval timer	Cannot be called.
Square wave output	Cannot be called.
Divider function	Cannot be called.
External event counter	Cannot be called.
Input pulse interval measurement	Cannot be called.
Input pulse high-/low-level width measurement	Cannot be called.
PWM output	After INTTM*mn* master channel interrupt.
One-shot pulse output	Cannot be called.
Multiple PWM output	After INTTM*mn* master channel interrupt.

[Classification]

CG_tau.c

[Syntax]

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

Remark *m* is the unit number, and *n* is the slave-side 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%.
In this example, channels 0 and 1 have been selected for PWM output or multiplex PWM output.

[CG_main.c]

```
#include    "CG_macrodriver.h"
```

```
void main ( void ) {
    UCHAR   ratio = 25;
    ......
    TAU0_Channel0_Start ();             /* Start count */
    ......
    TAU0_Channel1_ChangeDuty ( ratio ); /* Change duty ratio */
    ......
}
```

TAU*m*_Channel*n*_SoftWareTriggerOn

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Classification]

CG_tau.c

[Syntax]

```
void    TAUm_Channeln_SoftWareTriggerOn ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

3.3.10 Watchdog Timer

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

Table 3-11. API Functions: [Watchdog Timer]

API Function Name	Function
WDT_Init	Performs initialization necessary to control watchdog timer functions.
WDT_UserInit	Performs user-defined initialization relating to the watchdog timer.
WDT_Restart	Clears the watchdog timer counter and resumes counting.

WDT_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization necessary to control watchdog timer functions.

[Classification]

CG_wdt.c

[Syntax]

```
void    WDT_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

WDT_UserInit

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs user-defined initialization relating to the watchdog timer.

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

[Classification]

CG_wdt_user.c

[Syntax]

```
void    WDT_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

WDT_Restart

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Clears the watchdog timer counter and resumes counting.

[Classification]

CG_wdt.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

3.3.11 RTC

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

Table 3-12. 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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_ConstPeriodInterruptCallback

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmDisable

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmSet

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

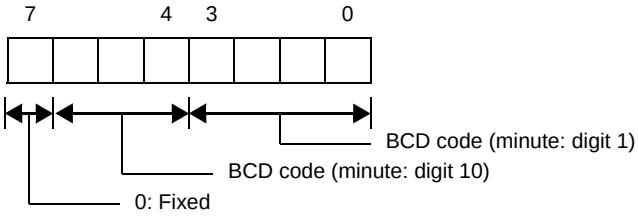
I/O	Argument	Description
I	`struct RTCAlarmValue alarmval;`	Alarm conditions (weekday, hour, minute)

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

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

- Alarmwm (Minute)

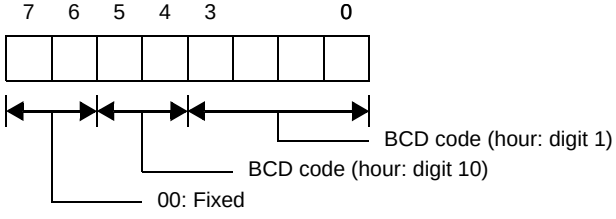
Below are shown the meanings of each bit of the structure member Alarmwm.



- Alarmwh (Hour)

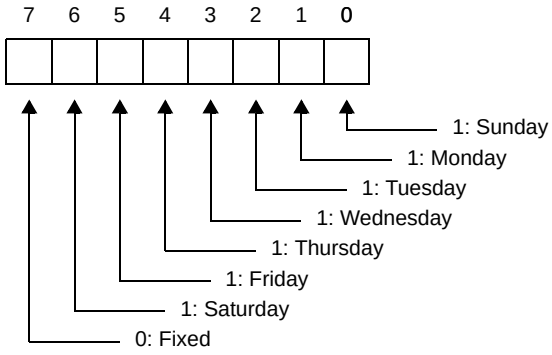
Below are shown the meanings of each bit of the structure member Alarmwh. If the real-time counter is set to the 12-hour clock, then bit 5 has the following meaning.

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalInterruptEnable

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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]

- [Kx3]

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

- [Kx3-A] [Kx3-L] [Lx3]

```
#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/fXT INTERVAL1: 2^7/fXT INTERVAL2: 2^8/fXT INTERVAL3: 2^9/fXT INTERVAL4: 2^10/fXT INTERVAL5: 2^11/fXT INTERVAL6: 2^12/fXT

Remark fXT is the frequency of the subsystem clock.

[Return value]

- [Kx3]

  None.

- [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTC1HZ_OutputEnable

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

3.3.12 Clock Output

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

Table 3-13. 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

[Fx3]

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

[Fx3]

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

[Fx3]

Starts clock output.

[Classification]

CG_pcl.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PCL_Stop

[Fx3]

Ends clock output.

[Classification]

CG_pcl.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PCL_ChangeFreq

[Fx3]

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"
void    PCL_ChangeFreq ( enum PCLclock clock );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PCLclock clock;`	Output clock type MAINCLOCK: fMAIN MAIN2: fMAIN/2 MAIN4: fMAIN/4 MAIN8: fMAIN/8 MAIN16: fMAIN/16 MAIN2048: fMAIN/2048 MAIN4096: fMAIN/4096 MAIN8192: fMAIN/8192 PLLCLOCK: fPLL PLL2: fPLL/2 PLL4: fPLL/4 PLL8: fPLL/8 PLL16: fPLL/16 PLL2048: fPLL/2048 PLL4096: fPLL/4096 PLL8192: fPLL/8192 ILCLOCK: fIL SUBCLOCK: fSUB

Remark fMAIN is the main system clock frequency; fPLL is the PLL clock frequency; fIL is the internal low-speed oscillation clock frequency; fSUB is the subsystem clock frequency.

[Return value]

None.

3.3.13 Clock Output/Buzzer Output

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

Table 3-14. API Functions: [Clock Output/Buzzer Output]

API Function Name	Function
PCLBUZn_Init	Performs initialization necessary to control clock/buzzer output control circuit functions.
PCLBUZn_UserInit	Performs user-defined initialization relating to the clock/buzzer output control circuits.
PCLBUZn_Start	Starts clock/buzzer output.
PCLBUZn_Stop	Ends clock/buzzer output.
PCLBUZn_ChangeFreq	Changes the output clock to the PCLBUZ*n* pin.

PCLBUZ*n*_Init

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization necessary to control clock/buzzer output control circuit functions.

[Classification]

CG_pclbuz.c

[Syntax]

```
void    PCLBUZn_Init ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_UserInit

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

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

[Classification]

CG_pclbuz_user.c

[Syntax]

```
void    PCLBUZn_UserInit ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_Start

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts clock/buzzer output.

[Classification]

CG_pclbuz.c

[Syntax]

```
void    PCLBUZn_Start ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_Stop

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Ends clock/buzzer output.

[Classification]

CG_pclbuz.c

[Syntax]

```
void    PCLBUZn_Stop ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_ChangeFreq

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Changes the output clock to the PCLBUZ*n* pin.

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

[Classification]

CG_pclbuz.c

[Syntax]

```
#include    "CG_pclbuz.h"
MD_STATUS   PCLBUZn_ChangeFreq ( enum PCLBUZclock clock );
```

Remark *n* is the output pin.

[Argument(s)]

I/O	Argument	Description
I	`enum PCLBUZclock clock;`	Output clock type MAINCLOCK: fMAIN MAIN2: fMAIN/2 MAIN4: fMAIN/4 MAIN8: fMAIN/8 MAIN16: fMAIN/16 MAIN2048: fMAIN/2048 MAIN4096: fMAIN/4096 MAIN8192: fMAIN/8192 SUBCLOCK: fSUB SUB2: fSUB/2 SUB4: fSUB/4 SUB8: fSUB/8 SUB16: fSUB/16 SUB32: fSUB/32 SUB64: fSUB/64 SUB128: fSUB/128

Remark fMAIN 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

[Example]

Below is an example of changing the output clock of pin PCLBUZ0 each time an externally connected button is pressed, in a system that generates the interrupt INTP0 each time the button is pressed.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_pclbuz.h"
BOOL    gFlag;                                  /* Button pressed flag */
#define PCLBUZ_FREQUENCY    8                   /* Total number of output clock types */
const   enum    PCLBUZclock gClock[PCLBUZ_FREQUENCY] = {    /* Output clock type */
    PCLBUZ_OUTCLK_fSUB0, PCLBUZ_OUTCLK_fSUB1, PCLBUZ_OUTCLK_fSUB2,
    PCLBUZ_OUTCLK_fSUB3, PCLBUZ_OUTCLK_fSUB4, PCLBUZ_OUTCLK_fSUB5,
    PCLBUZ_OUTCLK_fSUB6, PCLBUZ_OUTCLK_fSUB7
};
void main ( void ) {
    int     index = 0;
    gFlag = 0;                                  /* Initialize button pressed flag */
    ......
    PCLBUZ0_Start ();                           /* Start clock/buzzer output */
    while ( 1 ) {
        if ( gFlag ) {                          /* Wait for INTP0 */
            PCLBUZ_ChangeFreq ( gClock[index++] );  /* Change output clock */
            if ( index >= PCLBUZ_FREQUENCY ) {
                index = 0;
            }
            gFlag = 0;                          /* Clear button pressed flag */
        }
    }
}
```

[CG_int_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Button pressed flag */
__interrupt void MD_INTP0 ( void ) {        /* Interrupt processing for INTP0 */
    gFlag = 1;                              /* Set button pressed flag */
}
```

3.3.14 DMA

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

Table 3-15. API Functions: [DMA]

API Function Name	Function
DMAn_Init	Performs initialization necessary to control DMA controller functions.
DMAn_UserInit	Performs user-defined initialization relating to the DMA controller.
DMAn_Enable	Enables operation of channel *n*.
DMAn_Disable	Disables operation of channel *n*.
DMAn_Hold	Holds a DMA start request.
DMAn_Restart	Releases hold on a DMA start request.
DMAn_CheckStatus	Reads the transfer status (transfer complete/transfer ongoing).
DMAn_SetData	Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.
DMAn_SoftwareTriggerOn	Starts DMA transfer when DMA operation is enabled.

DMA*n*_Init

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs initialization necessary to control DMA controller functions.

[Classification]

CG_dma.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_UserInit

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Performs user-defined initialization relating to the DMA controller.

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

[Classification]

CG_dma_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Enable

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Enables operation of channel *n*.

[Classification]

CG_dma.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Disable

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Disables operation of channel *n*.

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

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

[Classification]

CG_dma.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

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

[CG_main.c]

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

DMA*n*_Hold

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Holds a DMA start request.

Remark Call DMAn_Restart to release the hold on DMA start requests.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Hold ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Restart

[Kx3] [Kx3-A] [Kx3-L] [Lx3]

Releases hold on a DMA start request.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Restart ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_CheckStatus

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Classification]

CG_dma.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

Macro	Description
MD_UNDEREXEC	Transfer ongoing
MD_COMPLETED	Transfer complete

DMA*n*_SetData

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

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

[Classification]

CG_dma.c

[Syntax]

- [Fx3] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( USHORT sfraddr, USHORT ramaddr, USHORT count );
```

- [Kx3]

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( USHORT ramaddr, USHORT count );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	USHORT *sfrcaddr*;	SFR address of source/destination
I	USHORT *ramcaddr*;	RAM address of source/destination
I	USHORT *count*;	Number of data transmissions (1 to 1024)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

DMA*n*_SoftwareTriggerOn

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Starts DMA transfer when DMA operation is enabled.

[Classification]

CG_dma.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

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

[CG_main.c]

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

3.3.15 OPAMP

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

Table 3-16. 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.
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

[Kx3-A] [Lx3]

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

[Kx3-A] [Lx3]

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.

AMP*n*_Start

[Kx3-A] [Lx3]

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

[Kx3-A] [Lx3]

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.

3.3.16 Comparator/PGA

Below is a list of API functions output by Applilet3 for comparator/programmable gain amplifiers use.

Table 3-17. API Functions: [Comparator/PGA]

API Function Name	Function
CMPPGA_Init	Performs initialization necessary to control comparator/programmable gain amplifiers functions.
CMPPGA_UserInit	Performs user-defined initialization relating to the comparator/programmable gain amplifiers.
CMPPGA_PowerOff	Halts the clock supplied to the comparator/programmable gain amplifiers.
CMPPGA_Start	Starts the operation of comparator/programmable gain amplifier.
CMPPGA_Stop	Ends the operation of comparator/programmable gain amplifier.
CMPPGA_ChangeCMPnRefVoltage	Sets comparator *n* internal reference voltage.
CMPPGA_ChangePGAFactor	Sets the input voltage amplification factor of a programmable gain amplifier.

CMPPGA_Init

[Kx3-L]

Performs initialization necessary to control comparator/programmable gain amplifiers functions.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_UserInit

[Kx3-L]

Performs user-defined initialization relating to the comparator/programmable gain amplifiers.

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

[Classification]

CG_cmppga_user.c

[Syntax]

```
void    CMPPGA_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_PowerOff

[Kx3-L]

Halts the clock supplied to the comparator/programmable gain amplifiers.

Remark Calling this API function changes the comparator/programmable gain amplifiers to reset status. For this reason, writes to the control registers (e.g. programmable gain amplifier control register: OAM) after this API function is called are ignored.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_Start

[Kx3-L]

Starts the operation of comparator/programmable gain amplifier.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_Stop

[Kx3-L]

Ends the operation of comparator/programmable gain amplifier.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_ChangeCMP*n*RefVoltage

[Kx3-L]

Sets comparator *n* internal reference voltage.

Remark The value specified in parameter *voltage* is set to comparator *n* internal reference voltage setting register (C*n*RVM).

[Classification]

CG_cmppga.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_ChangeCMPnRefVoltage ( enum CMPRefVoltage voltage );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`enum CMPRefVoltage voltage;`	Comparator *n* internal reference voltage [*n* = 0: In the case of the channel 0] CMPREFVOL0: 2AVREF/16 CMPREFVOL1: 4AVREF/16 CMPREFVOL2: 6AVREF/16 CMPREFVOL3: 8AVREF/16 CMPREFVOL4: 10AVREF/16 CMPREFVOL5: 12AVREF/16 [*n* = 1: In the case of the channel 1] CMPREFVOL0: 3AVREF/16 CMPREFVOL1: 5AVREF/16 CMPREFVOL2: 7AVREF/16 CMPREFVOL3: 9AVREF/16 CMPREFVOL4: 11AVREF/16 CMPREFVOL5: 13AVREF/16

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CMPPGA_ChangePGAFactor

[Kx3-L]

Sets the input voltage amplification factor of a programmable gain amplifier.

Remark The value specified in parameter *voltage* is set to programmable gain amplifier control register (OAM).

[Classification]

CG_cmppga.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_ChangePGAFactor ( enum CMPRefVoltage voltage );
```

[Argument(s)]

I/O	Argument	Description
I	`enum CMPRefVoltage voltage;`	Input voltage amplification factor PGAFACTOR0: x4 PGAFACTOR1: x6 PGAFACTOR2: x8 PGAFACTOR3: x10 PGAFACTOR4: x12

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

3.3.17 LVI

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

Table 3-18. 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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

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

LVI_Stop

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

Stops low-voltage detection.

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

LVI_SetLVILevel

[Fx3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

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

Remark If the target device is 78K0R/Fx3, then keyword that can be specified for *level* is limited to "LVILEVEL0" to "LVILEVEL9".

[Return value]

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

APPENDIX A INDEX

T

U

W

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/