RL78/ I1D

I2C Slave Control using Software (for Multiple Addresses) CC-RL

R01AN3289EJ0100
Rev. 1.00
2016.11.15

Introduction

This application note describes how to implement the multiple slave addresses by using the I2C bus slave function using software.

Operation Checked Device

RL78/I1D

When applied to other microcontrollers, this application note should be modified according to the specifications of the microcontroller used and a thorough evaluation should be made.

Contents

1. Basic Specifications of I2C Bus as Slave

1.1 I2C Bus Specifications

The following shows the basic specifications of the I2C bus.

- I2C bus connected: Fast mode (200 kbps max.) or standard mode[Note]
- Slave address 1: 0x60 (A/D conversion and LED display functions)
- Slave address 2: 0x70 (RAM function)
- Slave address 3: 0x80 (not used)
- Slave address 4: 0x90 (not used)
- Extension code: Not supported (ignores code and withdraws from communication)
- Addressing: 8 bits following the slave address used to specify the RAM address

Note: The communication speed when 24 MHz is selected for the CPU/peripheral hardware clock.

1.2 Slave Function Specifications

The following three slave functions are provided. One of the three functions is selected depending on the slave address and the state of transmission/reception.

- LED display function: 8-bit data is displayed on LED. Two display data units are switched over using SW.

- A/D conversion function: 4-channel analog input is converted to digital data. The moving average of the 16 samples is sent to the master.

- RAM function: 128-byte RAM function is provided. The master can read from and write to the arbitrary address by specifying it.

2. Conditions for Confirming Operations

The sample code operations described in this application note are confirmed under the following conditions.

Table 2.1 Conditions for Confirming Operations

Item	Description
Microcontroller used	RL78/I1D(R5F117GC)
Operating frequency	● High-speed on-chip oscillator (HOCO) clock: 24 MHz ● CPU/peripheral hardware clock: 24 MHz
Operating voltage	3.3V (operation possible within 2.9 V to 3.6 V) LVD operating mode: reset mode; voltage: 2.75 V
Integrated development environment	Renesas Electronics CS+ V3.03.00
Assembler	Renesas Electronics CC-RL V1.02.00[Note]
Board used	RL78/I1D target board (equipped with R5F117GC, LED (8 out of 10 of a module used), SW, and the like.)

[Note] Used in CA78K0R-compatible mode.

3. Related Application Notes

Refer to the following application notes, which are related to this application note.

RL78/G13 Initial Setting Application Note (R01AN2575J)
RL78/I1D I2C Master Communication Control using Serial Array Unit (Simple I2C) Application Note (R01AN3288J)

4. Hardware Descriptions

4.1 Hardware Configuration Example

Figure 4.1 shows an example of the hardware configuration described in this application note.

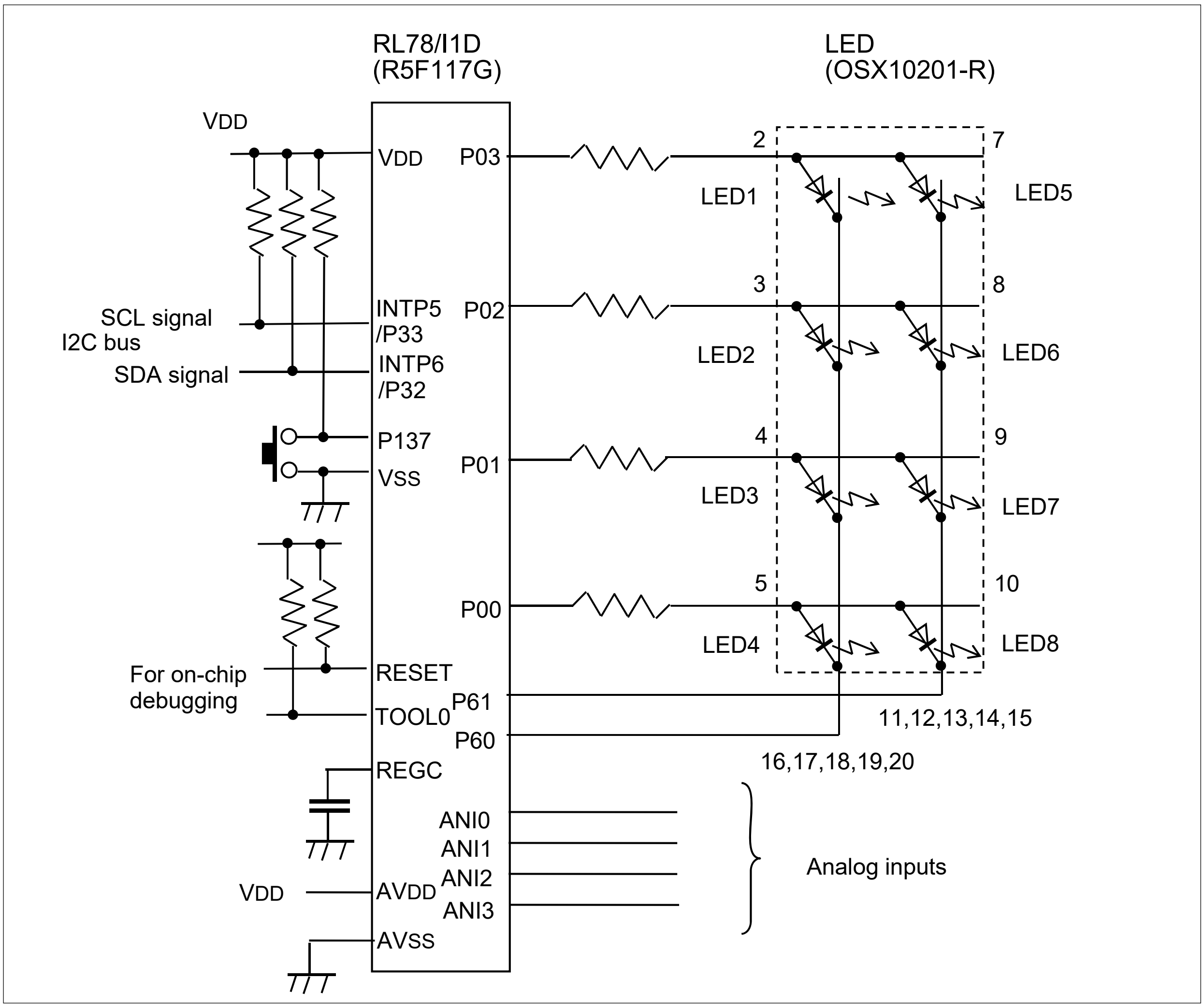


Figure 4.1 Hardware Configuration

Notes: 1. The above figure is a simplified circuit image for showing the outline of the connections. The actual circuit should be designed so that the pins are connected appropriately and that electrical characteristics are satisfied (input-only ports should be each connected to VDD or VSS via a resistor).

2. Set VDD to the reset-release voltage (V_{LVD}) specified by LVD or greater.

4.2 List of Pins Used

Table 4.1 lists the pins used and their functions.

Table 4.1 Pins Used and Their Functions

Pin Name	I/O	Function
INTP6/P32	I/O	I2C communication data signal
INTP5/P33	I/O	I2C communication clock signal
P03 to P00	Output	Data output to LED
P61, P60	Output	LED turning-on timing output
ANI3 to ANI0	Input	Analog signal input
P137	Input	SW input

5. Software Descriptions

5.1 Operation Summary

a) Initial Settings
In this application note, the CS+ code creation function is used only for the initial settings of the on-chip peripheral functions. After making the initial settings of the on-chip peripheral functions, data is initialized and the timers for A/D conversion and turning on LED are started.
- A/D conversion and turning-on of LED are processed on the background using the timer interrupts.
- I2C bus communication is processed on the background using the INTP5 and INTP6 interrupts.

b) Main Process
The main process waits for completion of 4-channel A/D conversion. When conversion is completed, the moving average is transferred to the I2C bus transmission buffer. The data transferred to the transmission buffer is transmitted to the I2C bus in response to the instruction from the master. During 4-channel A/D conversion, if the stop condition is detected on the I2C bus, data is transferred from the data reception buffer for turning on LED to the buffer for controlling turning on of LED.

c) A/D conversion end interrupt process
In the A/D conversion end interrupt process, the conversion result of each channel is added. When the count of data to be added reaches 16, the oldest data is replaced with the latest data. When A/D conversion of channels 0 to 3 is competed in scan mode, the main process is informed of completion of A/D conversion.

d) 5-ms timer interrupt process
The 5-ms timer interrupt is used to turn on LED and check SW. The upper 4-bit and lower 4-bit data for turning on LED are used in this order to turn on LED in the time division manner. The state of SW is checked every 50 ms to determine which data to be used.

e) I2C communication interrupt process
The changes of the SDA and SCL signals cause INTP5 and INTP6 interrupts. These interrupts are used as I2C communication interrupts. When an I2C communication interrupt is generated, the communication contents are analyzed and sent to the upper software. After completion of 1-byte communication, if the communication is intended for the slave itself, the communication status and received data are set in the variables, and the transmission/reception end flag (variable _g_IIC_IF) is set. For details, refer to 6.3 Specifications of Library Interface. If the stop condition is detected on the I2C bus, the variable for interfacing (_g_stop_det) is used to inform the main process that the stop condition has been detected, which indicates completion of the I2C bus communication.

As described above, almost all processes are performed based on the interrupts and flags. The main process sets data in the appropriate buffer so that the data prepared by an interrupt process can be used by another interrupt process.

5.2 List of Settings Reflected to Option Bytes

Table 5.1 shows the sample settings reflected to the option bytes.

Table 5.1 Settings Reflected to Option Bytes

Address	Setting	Content
0x000C0	0b11101110	Watchdog timer is stopped. (Counting stopped after a reset release)
0x000C1	0b01111111	LVD reset mode; 2.75V (2.70 V to 2.87 V)
0x000C2	0b11100000	HS mode; HOCO: 24 MHz
0x000C3	0b10000100	On-chip debugging is enabled.

5.3 List of Constants

Table 5.2 lists the constants used in sample codes.

Table 5.2 Constants Used in Sample Codes

Constant	Setting	Content
TRUTH	1	True
FALSE	0	False
POWER	4	Specification of A/D conversion sampling count (specify a factorial of 2)
SAMPLE	2 << (POWER-1)	A/D conversion sampling count
DATA_NUMBER	2	Number of data to be displayed on LED
INT_MASK	1	Interrupt disabled (masked)
INT_ENABLE	0	Interrupt enabled (mask canceled)
DETECT_START	0b11110010	Mask bit for detecting to be selected as slave
DETECT_TRC	0b00001000	TRC (transmission enable) bit
DETECT_ACK	0b00000100	ACK detection bit
DETECT_STD	0b00000010	Start condition detection bit
DETECT_STOP	0b00000001	Stop condition detection bit
DISP_OFF_DATA	0b00000011	P6 data for turning LED off
TIMING1	0b00000010	P6 data for lighting upper 4 bits
TIMING2	0b00000001	P6 data for lighting lower 4 bits
KEY_TIMING	10	Value for SW state check timing
DATA_MAX	4	Number of I2C transmission data
LED_MAX	2	Number of reception data for lighting LED
TX_LIMIT	DATA_MAX -1	Mask data for transmission pointer
RX_LIMIT	LED_MAX - 1	Mask data for LED lighting data reception pointer
SADR_TBL		Table of slave addresses used
ACK_TBL		ACK response table for each slave address ID
P_IIC	P3	Port used by the I2C bus
P_SCL	P3.3	Port used by the SCL signal
P_DATA	0b00001100	Data for extracting SCL and SDA
P_DATA_SCL	0b00001000	SCL is high; SDA is low.
P_SDA	P3.2	Port used by SDA signal
PM_SCL	PM3.3	Register for controlling SCL signal
PM_SDA	PM3.2	Register for controlling SDA signal
ENG_SCL	EGN0.5	SCL falling edge detection enabled
EPG_SCL	EPG0.5	SCL rising edge detection enabled
DIS_INTSCL	PMK5	SCL edge detection interrupt mask
DIS_INTSDA	PMK6	DA edge detection interrupt mask
RQ_INTSCL	PIF5	SCL edge detection interrupt request
RQ_INTSDA	PIF6	SDA edge detection interrupt request
D_SDA	0xFFEDE.2	SDA bit in P3 image
D_SCL	0xFFEDE.3	SCL bit in P3 image
F_TRC	0xFFEDF.3	Transmission mode bit in status (g_IICS)
F_ACKD	0xFFEDF.2	ACK detection bit in status (g_IICS)
F_STD	0xFFEDF.1	Start condition detection bit in status (g_IICS)
F_SPD	0xFFEDF.0	Stop condition detection bit in status (g_IICS)

5.4 List of Variables

Table 5.3 and table 5.4 list the variables used in sample codes

Table 5.3 List of Global Variables (for C Language Definitions)

Type	Variable Name	Contents	Function Used

uint16_t	g_conv_data	A/D conversion data buffer	R_ADC_Init()
			r_adc_interrupt()

uint16_t	g_sum_data	A/D conversion sum buffer	R_ADC_Init()
			r_adc_interrupt()

uint8_t	g_adc_end	A/D conversion end flag	R_ADC_Init()
			r_adc_interrupt()
			main()

uint16_t *	gp_set_pt	A/D conversion result storage pointer	R_ADC_Init () r_adc_interrupt()

uint16_t *	gp_sum_pt	A/D conversion result sum pointer	R_ADC_Init()
			r_adc_interrupt()
			main()

uint8_t	g_disp_data_bf	LED lighting data	R_LED_Init()
			R_LED_DispData()
			r_tau0_channel3_interrupt()

uint8_t	g_sel_data	Lighting data specification	R_LED_Init()
			r_tau0_channel3_interrupt()

uint8_t	g_disp_timing	Lighting timing	R_LED_Init()
			r_tau0_channel3_interrupt()

uint8_t	g_ram_area	Buffer for RAM function	R_IICA0_Init()
			_R_IIC_Rx_data()
			_R_IIC_Tx_data()

uint8_t	g_rx_data	Reception data buffer	R_IICA0_Init()
			R_IICA0_Get()
			_R_IIC_Rx_data()

uint16_t	g_tx_data	Transmission data buffer	R_IICA0_Init()
			R_IICA0_Put()
			_R_IIC_Tx_data()

uint8_t	g_low_data_temp	For storing lower byte of transmission data	_R_IIC_Tx_data()

uint8_t	g_low_data_index	Lower byte transmission flag	R_IICA0_Init()
			_R_IIC_Rx_data()
			_R_IIC_Tx_data()

uint8_t	g_regadr	Flag for address register	_R_IIC_Rx_data()

uint8_t	g_ptrx_data2	I2C transmission/reception pointer	R_IICA0_Init()
			_R_IIC_Rx_data()
			_R_IIC_Tx_data()

unit_t	g_ptrx_data	I2C transmission/reception pointer	R_IICA0_Init()
			_R_IIC_Rx_data()
			_R_IIC_Tx_data()

uint8_t	g_status	I2C communication state flag	R_IICA0_Init()
			R_IICA0_Status()
			_R_IIC_Rx_data()
			_R_IIC_Tx_data()

Table 5.4 List of Global Variables (for Assembly Language Definitions)

Type	Variable Name	Contents	Function Used

uint8_t	__g_stop_det
	(_g_stop_det)
		Stop condition detection flag	R_IICA0_Init()
			main()
			r_iic_SDA_interrupt

uint8_t	__g_IIC_IF (_g_IIC_IF)	Transmission/reception end flag	R_IICA0_Init() r_iic_int_chk ()
uint8_t	__g_IICS (_g_IICS)	Communication status	r_iic_request()
uint8_t	__g_IICA (_g_IICA)	Reception data	r_iic_request ()
uint8_t	g_ACKE_tbl	Internal buffer for storing initial values to control ACK response to the address	__R_IICSS_Init __set_ACKE_table __get_ACKE_table r_iic_SCL_interrupt
uint8_t	g_ACKE	For controlling ACK response	r_iic_SCL_interrupt
uint16_t	next_proc	Next INTP5 processing function address	_R_IICSS_Init() r_iic_SCL_interrupt r_iic_SDA_interrupt
uint8_t[注]	bit_count	Transmission/reception bit count	r_iic_SCL_interrupt
uint8_t[注]	g_IICA	Data being shifted during transmission/reception	r_iic_SCL_interrupt
uint8_t	g_P_image	For storing P3 data	r_iic_SCL_interrupt
uint8_t	g_IICS[注2]	I2C status	r_iic_SCL_interrupt

Notes: 1. bit_count and g_IICA may be accessed simultaneously by a 16-bit access.

2. The variable g_IICS for indicating I2C communication state has a structure conforming to that of the IICA0.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ID3	ID2	ID1	ID0	F_TRC	F_ACKD	F_STD	F_SPD

[Remarks] The variables __g_stop_det (_g_stop_det), __g_IIC_IF (_g_IIC_IF), __g_IICS (_g_IICS), and __g_IICA (_g_IICA) can be accessed from C language description. The other variables can only be used in library functions described in the assembly language.

5.5 List of Functions

Table 5.5 lists the functions used.

Table 5.5 List of Functions

Function Name	Summary
R_IICA0_Init()	Initializes variables relating to I2C communication.
R_IICA0_Status()	Reads out I2C communication state.
R_IICA0_Get()	Reads out I2C reception data.
R_IICA0_Put()	Sets data in I2C transmission buffer.
r_iic_int_chk()	Checks I2C communication completion.
r_iic_request()	I2C communication end interrupt processing
R_ADC_Init()	Initializes variables relating to A/D conversion.
R_ADC_Start()	Starts A/D conversion.
r_adc_interrupt()	Processes A/D conversion end interrupt.
R_LED_Init()	Initializes variables relating to LED display.
R_TM03_Start()	Starts 5-ms interval timer.
R_LED_DispData ()	Sets LED light-emitting data.
r_tau0_channel3_interrupt()	Processes 5-ms interval timer interrupt.
__R_IICSS_Init	Initializes I2C.
__R_IICSS_Status	Returns I2C state (g_IICS value).
__set_ACKE_table	Sets ACK response for the slave address ID
__get_ACKE_table	Reads out ACK response setting for the slave address ID
r_iic_SCL_interrupt	Processes SCL signal edge detection interrupt.
r_iic_SDA_interrupt	Processes SDA signal edge detection interrupt.
__Tx_data_sub	Cancels I2C bus wait and starts next data transmission.
__Tx_end_sub	Exits communication and cancels I2C bus wait.
__Rx_data_sub	Cancels I2C bus wait and starts next data reception.

A triple line indicates a border between different modules.

5.6 Function Specifications

The following gives the specifications of the functions used in the sample code.

[Function name] R_IICA0_Init

Summary	Initializes I2C communication.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	void R_IICA0_Init(void);
Description	Initializes variables used for the I2C communication.
Arguments	None
Return values	None
Remarks	None

[Function name] R_IICA0_Status

Summary	Checks I2C communication state.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	uint8_t R_IICA0_Status(void);
Description	Reads the variable g_IICS indicating the I2C communication state. If the I2C communication has been completed, performs the corresponding processing.
Arguments	None
Return values	Value of variable g_IICS (g_status)
Remarks	None

[Function name] R_IICA0_Get

Summary	Reads reception data from I2C reception data buffer.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	uint8_t R_IICA0_Get(uint8_t ptr);
Description	Reads data specified with the argument (lighting data) from the I2C reception buffer. If the I2C communication has been completed, the corresponding processing is performed.
Arguments	Specifies reception data buffer.
Return values	Received data
Remarks	None

[Function name] R_IICA0_Put

Summary	Sets data in I2C transmission buffer.	
Header	r_cg_macrodriver.h, r_cg_userdefine.h	
Declaration	void R_IICA0_Put(uint8_t ptr,uint16_t data);	
Description	Stores data indicated by the second argument (A/D conversion result) into the address specified by the first argument in the I2C transmission buffer. If the I2C communication has been completed, the corresponding processing is performed.	
Arguments	First argument	Data storage address
	Second argument	Data to be transmitted
Return values	None	
Remarks	None	

[Function name] r_iic_int_chk

Summary	Checks I2C communication completion.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	uint8_t r_iic_int_chk (void);
Description	Checks the I2C communication end interrupt flag and calls r_iic_request if the communication has been completed.
Arguments	None
Return values	I2C status
Remarks	None

[Function name] r_iic_requestr_IIC_interrupt

Summary	Performs I2C communication completion processing.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	void r_iic_request (void);
Description	This processing corresponds to INTIICA0 of IICA0. Performs processing according to the I2C status (_g_IICS) value.
Arguments	None
Return values	None
Remarks	_g_IICS has I2C communication status. _g_IICA has receive data.

[Function name] R_ADC_Init

Summary	Makes A/D conversion initial settings.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	void R_ADC_Init(void);
Description	Initializes variables relating to A/D conversion.
Arguments	None
Return values	None
Remarks	None

[Function name] R_ADC_Start

Summary	Starts A/D conversion.
Header	r_cg_userdefine.h
Declaration	void R_ADC_Start(void);
Description	Starts the A/D converter.
Arguments	None
Return values	None
Remarks	None

[Function name] r_adc_interrupt

Summary	Processes an A/D conversion end interrupt.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	#pragma interrupt r_adc_interrupt(vect=INTAD,bank=RB2,enable=true) __interrupt static void r_adc_interrupt(void);
Description	Started by an A/D conversion end interrupt; stores the obtained conversion results in the buffer, and simultaneously adds the results 16 times for each channel.
Arguments	None
Return values	None
Remarks	None

[Function name] R_LED_Init

Summary	Performs initialization for LED lighting.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	void R_LED_Init (void);
Description	Initializes variables for controlling LED lighting.
Arguments	None
Return values	None
Remarks	None

[Function name] R_TM03_Start

Summary	Starts TM03 (interval timer).
Header	r_cg_userdefine.h
Declaration	void R_TM03_Start(void);
Description	Starts TM03 (5-ms interval timer).
Arguments	None
Return values	None
Remarks	None

[Function name] R_LED_DispData

Summary	Sets LED light-emitting data.	
Header	r_cg_macrodriver.h, r_cg_userdefine.h	
Declaration	void R_LED_DispData(uint8_t CH_No,uint8_t in_data);	
Description	Stores data specified by the second argument in the buffer specified by the first argument.	
Arguments	First argument	Data storage channel
	Second argument	Data to be set
Return values	None	
Remarks	None	

[Function name] r_tau0_channel3_interrupt

Summary	Processes a 5-ms interval timer interrupt.
Header	r_cg_macrodriver.h, r_cg_userdefine.h
Declaration	#pragma interrupt r_tau0_channel3_interrupt(vect=INTTM03, enable=true) __interrupt static void r_tau0_channel3_interrupt(void);
Description	Started by a 5-ms interrupt; controls dynamic LED lighting in 4-bit units. Checks the state of SW connected to P137 every 50 ms and changes data to be lighted.
Arguments	None
Return values	None
Remarks	None

[Function name] __R_IICSS_Init

Summary	Initializes I2C.
Declaration	void _R_IICSS_Init (void);
Description	Initializes I2C control variables and hardware.
Arguments	None
Return values	None
Remarks	None

[Function name] __R_IICSS_Status

Summary	Checks I2C status.
Declaration	uint8_t _R_IICSS_Status (void);
Description	Passes the value of variable g_IICS indicating I2C bus state.
Arguments	None
Return values	I2C bus status
Remarks	None

[Function name] __set_ACKE_table

Summary	Sets ACK response for the slave address ID.
Declaration	void _set_ACKE_table(uint8_t ACKE);
Description	Sets the ACK response for the slave address ID indicated by the argument bits 4 to 1, to the state indicated by bit 0.
Arguments	ACK response
Return values	None
Remarks	None

[Function name] __get_ACKE_table

Summary	Reads out ACK response setting for the slave address ID.
Declaration	uint8_t _get_ACKE_table(uint8_t ID);
Description	Returns ACK response setting for the slave address passed by the argument.
Arguments	Slave address ID
Return values	ACK response setting value
Remarks	None

[Function name] r_iic_SCL_interrupt

Summary	Processes an SCL edge detection interrupt.
Declaration	r_iic_SCL_interrupt .VECTOR 0x00012
Description	Detects the SCL signal edge, reads SCL and SDA, and performs the corresponding processing. The processed contents are indicated by the address stored in the variable next_proc.
Arguments	None
Return values	None
Remarks	None

[Function name] r_iic_SDA_interrupt

Summary	Processes an SDA edge detection interrupt.
Declaration	r_iic_SDA_interrupt .VECTOR 0x00014
Description	Started on detection of the SDA signal edge; detects the start and stop conditions based on the SDA and SCL signal state.
Arguments	None
Return values	None
Remarks	None

[Function name] __Tx_data_sub

Summary	Starts next data transmission.
Declaration	void _Tx_data_sub(uint8_t data);
Description	Outputs MSB of the data passed by the argument to the SDA signal, cancels the I2C bus wait, and starts data transmission.
Arguments	Next transmission data
Return values	None
Remarks	None

[Function name] __Rx_data_sub

Summary	Starts next data reception.
Declaration	void _Rx_data_sub(void);
Description	Cancels the I2C bus wait, and starts data reception.
Arguments	None
Return values	None
Remarks	None

[Function name] __Tx_end_sub

Summary	Performs transmission completion processing.
Declaration	void _Tx_end_sub(void);
Description	Exits the communication and cancels the I2C bus wait.
Arguments	None
Return values	None
Remarks	None

5.7 Flowcharts

Figure 5.1 shows the overall flow of the process described in this application note.

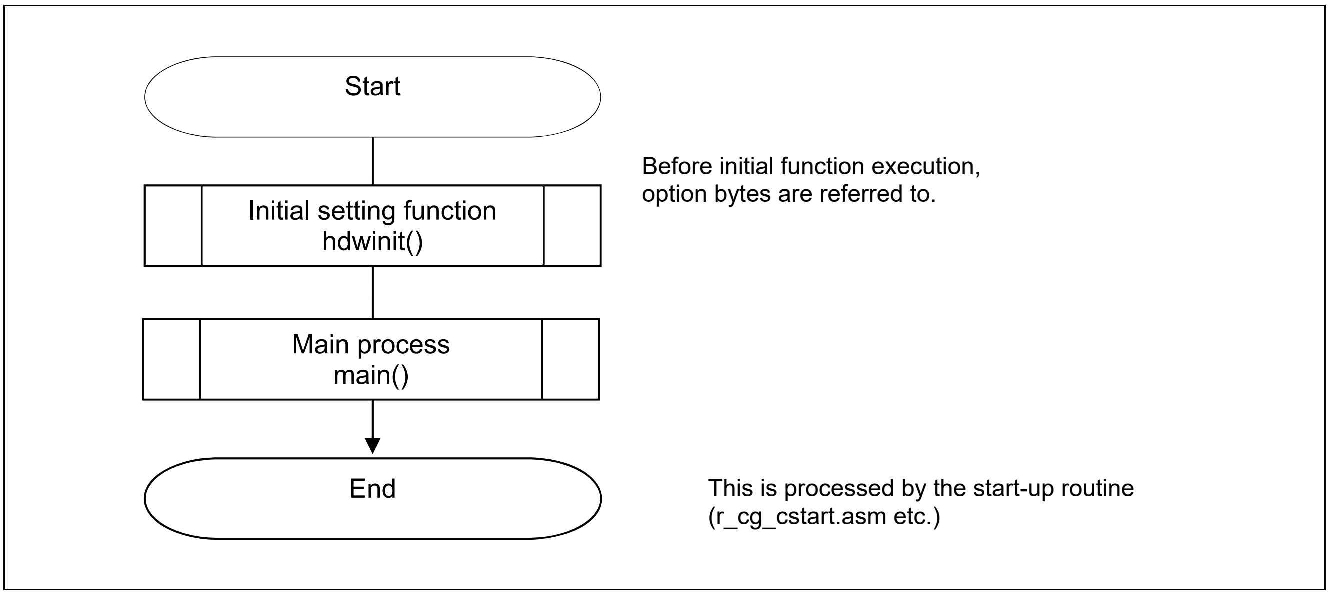

Start
Initial setting function
hdwinit()
Main process
main()
End
Before initial function execution,
option bytes are referred to.
This is processed by the start-up routine
(r_cg_cstart.asm etc.)

Figure 5.1 Overall Flow

Note: This is processed by the start-up routine (r_cg_cstart.asm etc.). The memory-related settings are made between calling the initial setting function and main process function.

5.7.1 Initial Setting Function

Figure 5.2 shows the flowchart of the initial setting function.

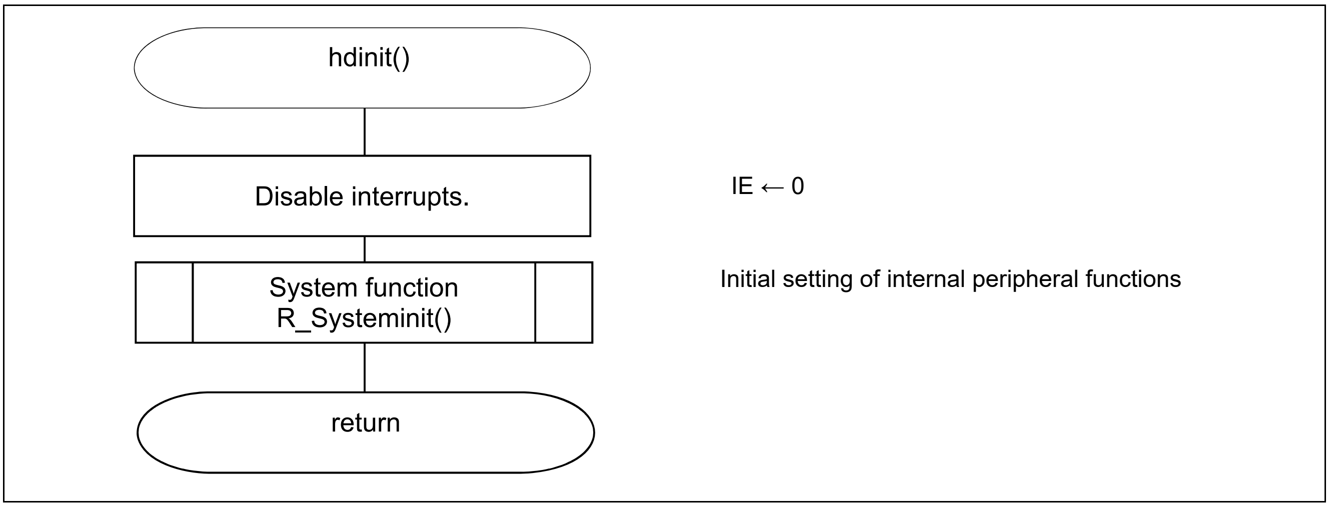


Figure 5.2 Initial Setting Function

5.7.2 System Function

Figure 5.3 shows the flowchart of the system function.

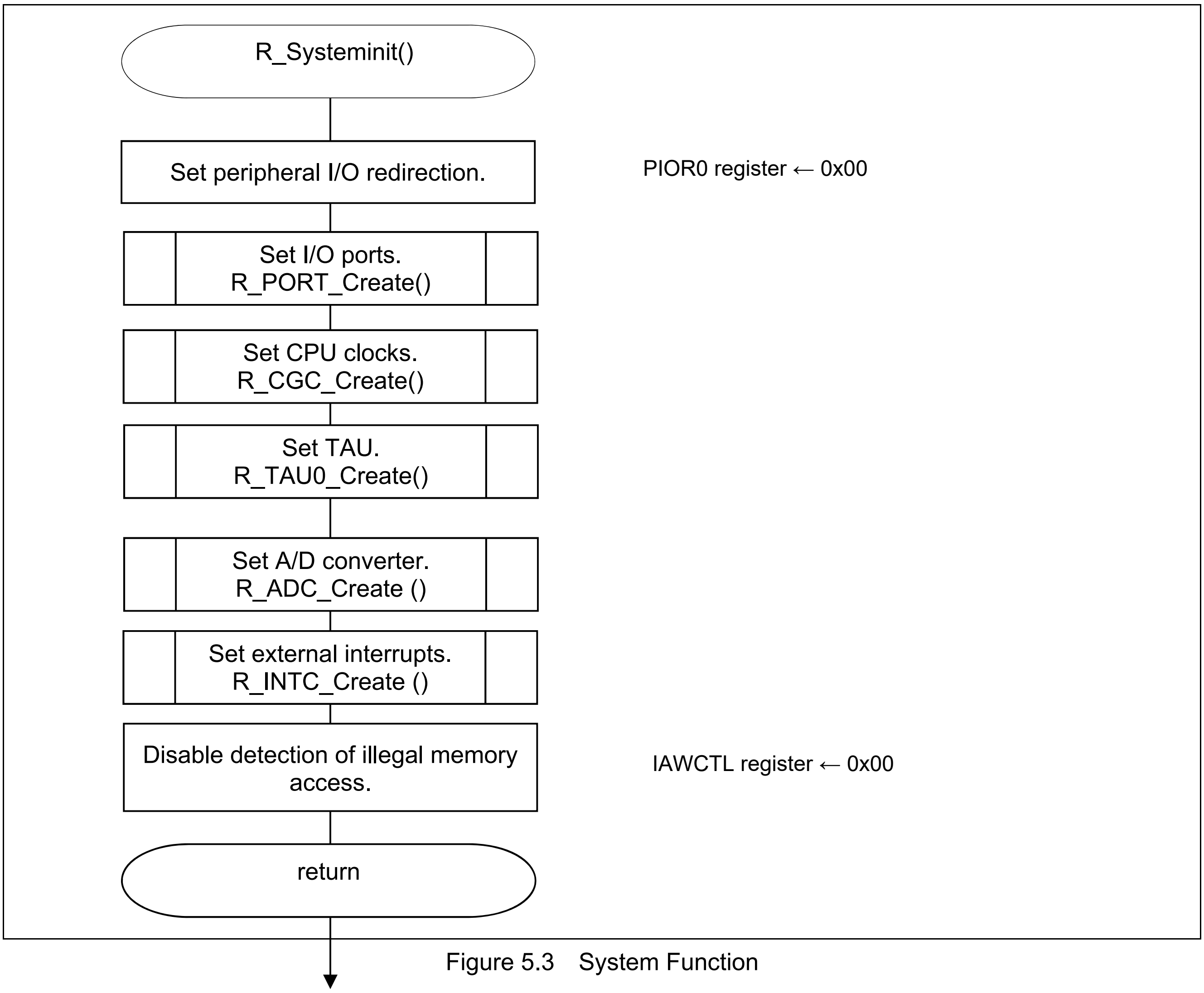


Figure 5.3 System Function

5.7.3 Setting CPU Clocks

Figure 5.4 shows the flowchart for setting the CPU clocks.

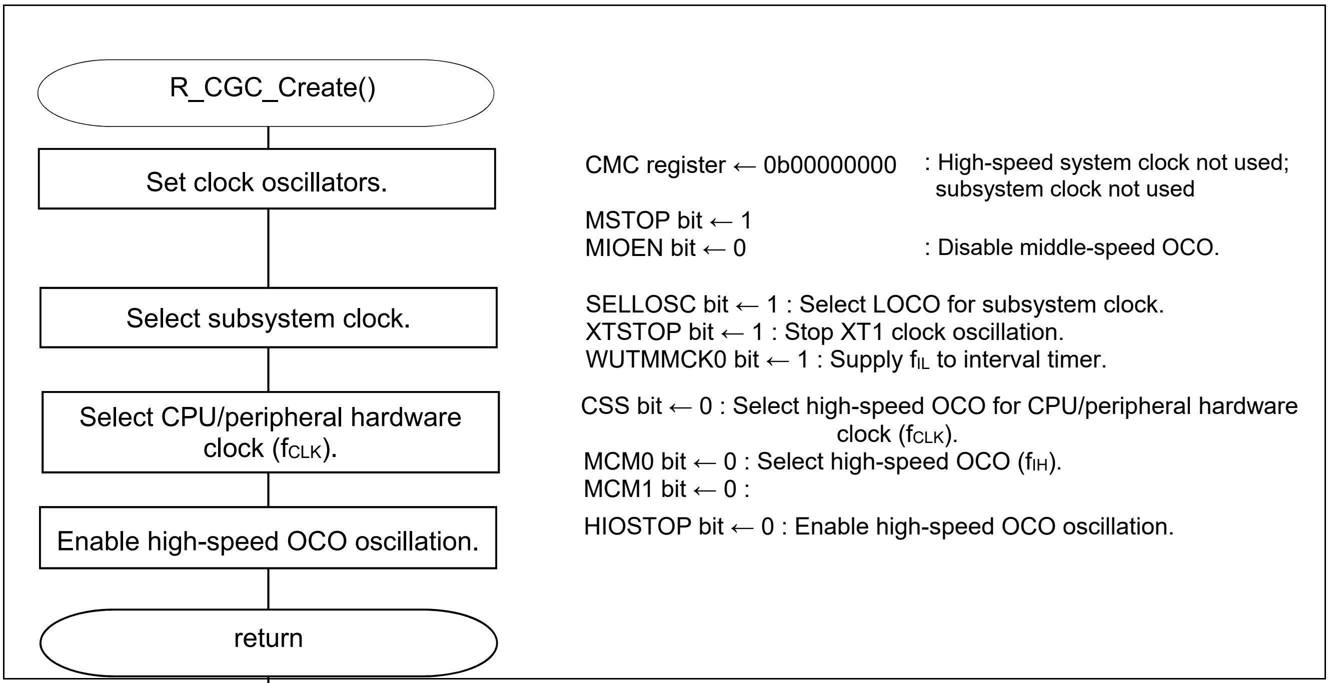

R_CGC_Create()
Set clock oscillators.
Select subsystem clock.
Select CPU/peripheral hardware clock (f_{CLK}).
Enable high-speed OCO oscillation.
return
CMC register ← 0b00000000 : High-speed system clock not used; subsystem clock not used
MSTOP bit ← 1
MIOEN bit ← 0 : Disable middle-speed OCO.
SELLOSC bit ← 1 : Select LOCO for subsystem clock.
XTSTOP bit ← 1 : Stop XT1 clock oscillation.
WUTMMCK0 bit ← 1 : Supply f_{IL} to interval timer.
CSS bit ← 0 : Select high-speed OCO for CPU/peripheral hardware clock (f_{CLK}).
MCM0 bit ← 0 : Select high-speed OCO (f_{IH}).
MCM1 bit ← 0 :
HIOSTOP bit ← 0 : Enable high-speed OCO oscillation.

Figure 5.4 Setting CPU Clocks

5.7.4 Setting I/O Ports

Figure 5.5 shows the flowchart for setting the I/O ports.

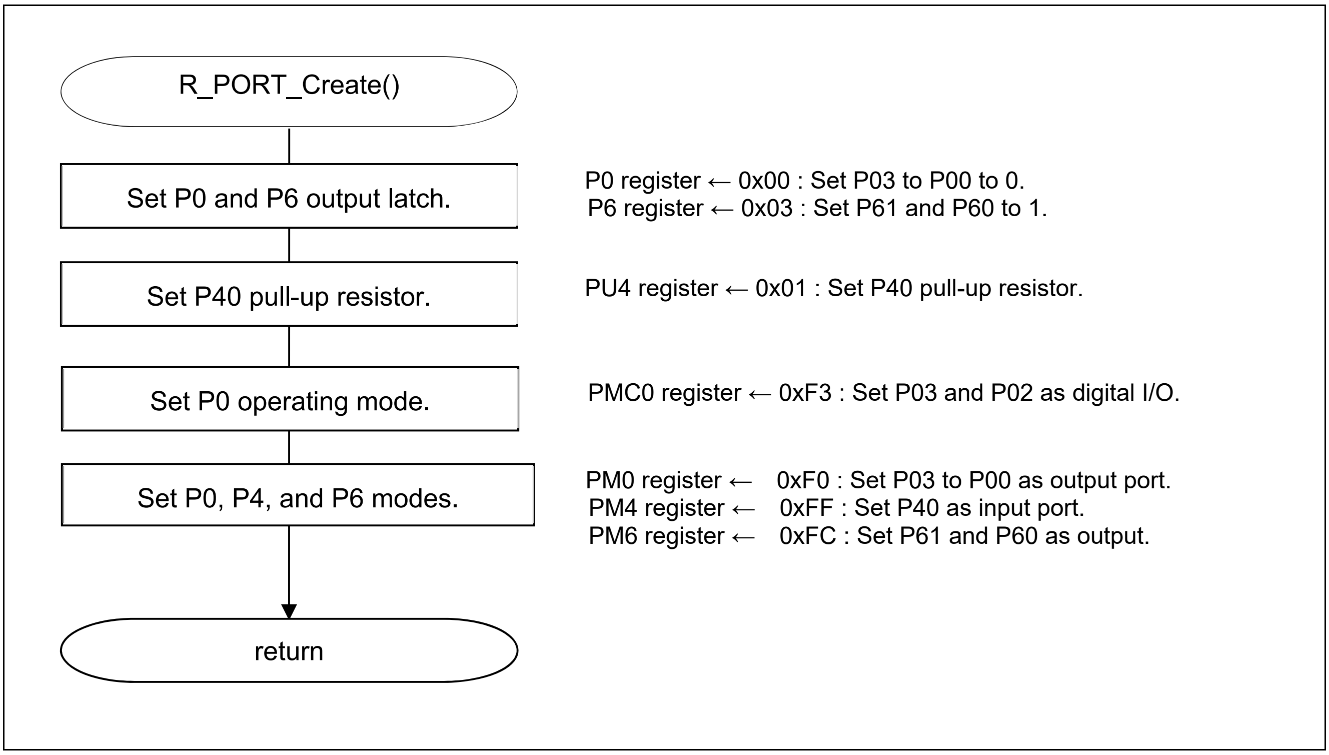

R_PORT_Create()
Set P0 and P6 output latch.
P0 register ← 0x00 : Set P03 to P00 to 0.
P6 register ← 0x03 : Set P61 and P60 to 1.
Set P40 pull-up resistor.
PU4 register ← 0x01 : Set P40 pull-up resistor.
Set P0 operating mode.
PMC0 register ← 0xF3 : Set P03 and P02 as digital I/O.
Set P0, P4, and P6 modes.
PM0 register ← 0xF0 : Set P03 to P00 as output port.
PM4 register ← 0xFF : Set P40 as input port.
PM6 register ← 0xFC : Set P61 and P60 as output.
return

Figure 5.5 Setting I/O Ports

Note: Design unused ports so that the electrical characteristics are satisfied by appropriately treating the pertinent pins. Separately connect unused input-only pins to VDD or VSS via a resistor.

5.7.5 Setting Timer Array Unit

Figure 5.6 shows the flowchart for setting the timer array unit.

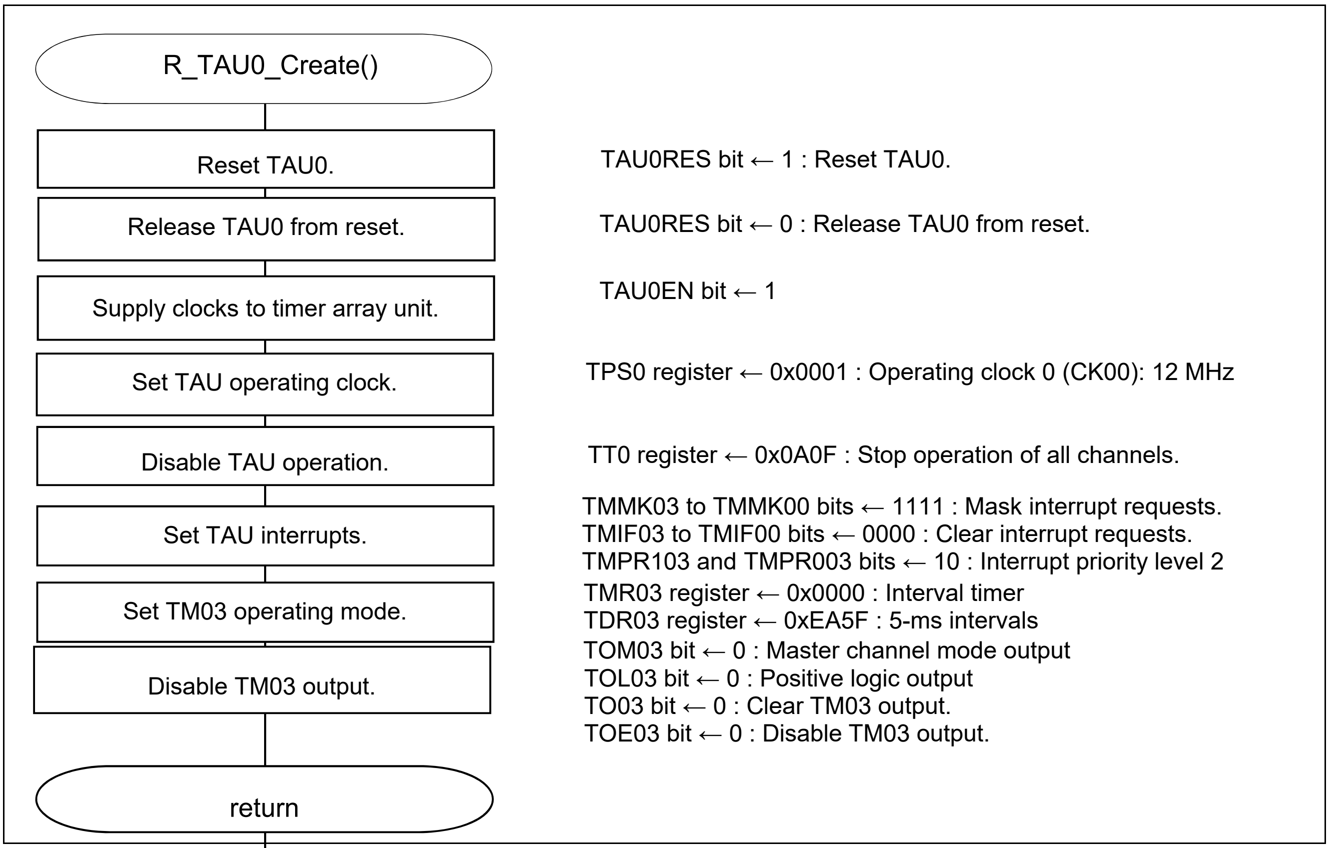

R_TAU0_Create()
Reset TAU0.
TAU0RES bit ← 1 : Reset TAU0.
Release TAU0 from reset.
TAU0RES bit ← 0 : Release TAU0 from reset.
Supply clocks to timer array unit.
TAU0EN bit ← 1
Set TAU operating clock.
TPS0 register ← 0x0001 : Operating clock 0 (CK00): 12 MHz
Disable TAU operation.
TT0 register ← 0x0A0F : Stop operation of all channels.
Set TAU interrupts.
TMMK03 to TMMK00 bits ← 1111 : Mask interrupt requests.
TMIF03 to TMIF00 bits ← 0000 : Clear interrupt requests.
TMPR103 and TMPR003 bits ← 10 : Interrupt priority level 2
Set TM03 operating mode.
TMR03 register ← 0x0000 : Interval timer
TDR03 register ← 0xEA5F : 5-ms intervals
Disable TM03 output.
TOM03 bit ← 0 : Master channel mode output
TOL03 bit ← 0 : Positive logic output
TO03 bit ← 0 : Clear TM03 output.
TOE03 bit ← 0 : Disable TM03 output.
return

Figure 5.6 Setting Timer Array Unit

Resetting TAU0

- Peripheral reset control register 0 (PRR0)
  Reset TAU0.

Symbol: PRR0

7	6	5	4	3	2	1	0
0	0	ADCRES	0	0	SAU0RES	0	TAU0RES
0	0	x	0	0	x	0	**1/0**

Bit 0

SAU0RES	Reset control of timer array unit
0	**Releases the timer array unit from the reset state.**
1	**Resets the timer array unit.**

Starting clock supply to timer array unit 0

- Peripheral enable register 0 (PER0)

  Start clock supply to timer array unit 0.

Symbol: PER0

7	6	5	4	3	2	1	0
RTCWEN	0	ADCEN	0	0	SAU0EN	0	TAU0EN
x	0	x	0	0	x	0	**1**

Bit 0

TAU0EN	Control of input clock to timer array unit 0
0	入力クロック供給停止
1	**Supplies input clock.**

Setting timer clock frequency

- Timer clock select register 0 (TPS0)

  Select the operating clock for timer array unit 0.

Symbol: TPS0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	PRS 031	PRS 030	0	0	PRS 021	PRS 020	PRS 013	PRS 012	PRS 011	PRS 010	PRS 003	PRS 002	PRS 001	PRS 000
0	0	x	x	0	0	x	x	x	x	x	x	**0**	**0**	**0**	**1**

Bits 3 to 0

PRS 003	PRS 002	PRS 001	PRS 000	Operating clock (CK00) selection					
					f_{CLK}= 1MHz	f_{CLK}= 2MHz	f_{CLK}= 4MHz	f_{CLK}= 16MHz	f_{CLK}= 24MHz
0	0	0	0	f_{CLK}	1 MHz	2 MHz	4 MHz	16MHz	24 MHz
0	**0**	**0**	**1**	**$f_{CLK}/2$**	500 kHz	1 MHz	2 MHz	8 MHz	**12 MHz**
0	0	1	0	$f_{CLK}/2^2$	250 kHz	500 kHz	1 MHz	4MHz	6 MHz
0	0	1	1	$f_{CLK}/2^3$	125 kHz	250 kHz	500 kHz	2 MHz	3 MHz
0	1	0	0	$f_{CLK}/2^4$	62.5 kHz	125 kHz	250 kHz	1 MHz	1.5 MHz
0	1	0	1	$f_{CLK}/2^5$	31.3 kHz	62.5 kHz	125 kHz	500 kHz	750 kHz
0	1	1	0	$f_{CLK}/2^6$	15.6 kHz	31.3 kHz	62.5 kHz	250 kHz	375 kHz
0	1	1	1	$f_{CLK}/2^7$	7.81 kHz	15.6 kHz	31.3 kHz	125 kHz	187.5 kHz
1	0	0	0	$f_{CLK}/2^8$	3.91 kHz	7.81 kHz	15.6 kHz	62.5 kHz	93.8 kHz
1	0	0	1	$f_{CLK}/2^9$	1.95 kHz	3.91 kHz	7.81 kHz	31.3 kHz	46.9 kHz
1	0	1	0	$f_{CLK}/2^{10}$	977 Hz	1.95 kHz	3.91 kHz	15.6 kHz	23.4 kHz
1	0	1	1	$f_{CLK}/2^{11}$	488 Hz	977 Hz	1.95 kHz	7.81 kHz	11.7 kHz
1	1	0	0	$f_{CLK}/2^{12}$	244 Hz	488 Hz	977 Hz	3.91 kHz	5.86 kHz
1	1	0	1	$f_{CLK}/2^{13}$	122 Hz	244 Hz	488 Hz	1.95 kHz	2.93 kHz
1	1	1	0	$f_{CLK}/2^{14}$	61 Hz	122 Hz	244 Hz	977 Hz	1.46 kHz
1	1	1	1	$f_{CLK}/2^{15}$	30.5 Hz	61 Hz	122 Hz	488 Hz	732 Hz

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

Stopping timer operation

- Timer channel stop register 0 (TT0)
  Select to stop timer channel operation.

Symbol: TT0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	TT03H	0	TT01H	0	0	0	0	0	TT03	TT02	TT01	TT00
0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	**1**	**1**	**1**

Bit n

TT0n	Trigger for stopping channel n operation
0	トリガ動作しない
1	**Clears TE0n bit to 0 to stop counting operation (stop-trigger generated).**

Setting timer count end interrupt

- TMMK03 bit in interrupt mask flag register (MK1L)
  Set interrupt masks.
- TMIF03 bit in interrupt request flag register (IF1L)
  Clear interrupt request flags.
- MPR003 and TMPR103 bits in priority setting flag registers (PR01L, PR11L)
  Set TM03 interrupts to priority level 2.

Symbol: MK1L

Bit 5

TMMK03		Control of interrupt processing
0		割り込み処理許可
1		**Disables interrupt processing.**

Symbol: IF1L

Bit 5

TMIF03	Interrupt request flag
0	**No interrupt request signals have been generated.**
1	割り込み要求信号が発生し、割り込み要求状態

Symbol: PR01L、PR11L

Bit 5

TMPR103	TMPR003	INTTM03 priority level selection
0	0	レベル 0 を指定(高優先順位)
0	1	レベル 1 を指定
1	**0**	**Level 2**
1	1	レベル 3 を指定(低優先順位)

Note: For details of register settings, refer to the RL78/I1D User's Manual: Hardware.

Setting channel 3 operating mode

- Timer mode register 03 (TMR03)
  Select operating clock (f_{MCK}).
  Select the counting clock.
  Set start and capture triggers.
  Select valid edge of timer input.
  Set the operating mode.

Symbol: TMR03

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CKS 031	CKS 030	0	CCS 03	SPLIT 03	STS 032	STS 031	STS 030	CIS 031	CIS 030	0	0	MD 033	MD 032	MD 031	MD 030
0	**0**	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	**0**	**0**	**0**	**0**

Bits 15 and 14

CKS031	CKS030	Channel 3 operating clock (f_{MCK}) selection
0	**0**	**Operating clock CK00 specified with timer clock selection register 0 (TPS0)**
0	1	タイマ・クロック選択レジスタ 0（TPS0）で設定した動作クロック CK02
1	0	タイマ・クロック選択レジスタ 0（TPS0）で設定した動作クロック CK01
1	1	タイマ・クロック選択レジスタ 0（TPS0）で設定した動作クロック CK03

Bit 12

CCS03	Channel 3 counting clock (f_{TCL}) selection
0	**Operating clock (f_{MCK}) specified with CKS031 and CKS030 bits**
1	TI03 端子からの入力信号の有効エッジ

Bit 11

SPLIT03	8-bit timer/16-bit timer operation selection of channel 3
0	**16-bit timer operation**
1	8 ビット・タイマとして動作

Bits 10 to 8

STS032	STS031	STS030	Setting for channel 3 start and capture triggers
0	**0**	**0**	**Only software trigger start is valid (deselect the other trigger sources.)**
0	0	1	TI00 端子入力の有効エッジを、スタート・トリガ、キャプチャ・トリガの両方に使用
0	1	0	TI00 端子入力の両エッジを、スタート・トリガとキャプチャ・トリガに分けて使用
1	0	0	マスタ・チャネルの割り込み信号を使用（複数チャネル連動動作機能のスレーブ・チャネル時）
上記以外			設定禁止

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

Bits 7 and 6

CIS031	CIS030	Valid edge selection of TI03 pin
0	**0**	**Falling edge**
0	1	立ち上がりエッジ
1	0	両エッジ（ロウ・レベル幅測定時） スタート・トリガ：立ち下がりエッジ、キャプチャ・トリガ：立ち上がりエッジ
1	1	両エッジ（ハイ・レベル幅測定時） スタート・トリガ：立ち上がりエッジ、キャプチャ・トリガ：立ち下がりエッジ

Bits 3 to 0

MD 033	MD 032	MD 031	MD 030	Channel 3 operating mode	Corresponding functions	TCR counting operation
0	**0**	**0**	1/**0**	**Interval timer mode**	**Interval timer/square wave output/divider function/PWM output (master)**	Decrementing
0	1	0	1/0	キャプチャ・モード	入力パルス間隔測定	アップ・カウント
0	1	1	0	イベント・カウンタ・モード	外部イベント・カウンタ	ダウン・カウント
1	0	0	1/0	ワンカウント・モード	ディレイ・カウンタ／ワンショット・パルス出力／PWM 出力（スレーブ）	ダウン・カウント
1	1	0	0	キャプチャ＆ワンカウント・モード	入力信号のハイ／ロウ・レベル幅測定	アップ・カウント
上記以外				設定禁止		

Setting delay time

- Timer data register 03 (TDR03)
  Set the delay time.

Symbol: TDR03

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

Disabling timer output

- Timer output mode register 0 (TOM0L)
  Set master mode output.
- Timer output level register 0 (TOL0L)
  Set positive logic output.
- Timer output register 0 (TO0L)
  Set output to 0.
- Timer output enable register 0 (TOE0L)
  Enable/disable timer output of each channel.

Symbol: TOM0L

7	6	5	4	3	2	1	0
0	0	0	0	TOM03	TOM02	TOM01	0
0	0	0	0	**0**	x	x	0

Bit 3

TOM03	Control of channel 3 timer output mode
0	**Master channel output mode**
1	スレーブ・チャネル出力モード

Symbol: TOL0L

7	6	5	4	3	2	1	0
0	0	0	0	TOL03	TOL02	TOL01	0
0	0	0	0	**0**	x	x	0

Bit 3

TOL03	Control of channel 3 timer output level
0	**Positive logic output (active high)**
1	反転出力(アクティブ・ロウ)

Symbol: TO0L

7	6	5	4	3	2	1	0
0	0	0	0	TO03	TO02	TO01	TO00
0	0	0	0	**0**	x	x	x

Bit 3

TO03	Control of channel 3 timer output level
0	**Low**
1	ハイ・レベル

Symbol: TOE0L

7	6	5	4	3	2	1	0
0	0	0	0	TOE03	TOE02	TOE01	TOE00
0	0	0	0	**0**	x	x	x

Bit 3

TOE03	Control of channel 3 timer output enable/disable
0	**Disables TO03 (timer channel output bit) operation triggered by counting operation.**
1	カウント動作による TO03（タイマ・チャネル出力ビット）の動作許可。

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

5.7.6 Setting A/D Converter

Figure 5.7 shows the flowchart for setting the A/D converter.

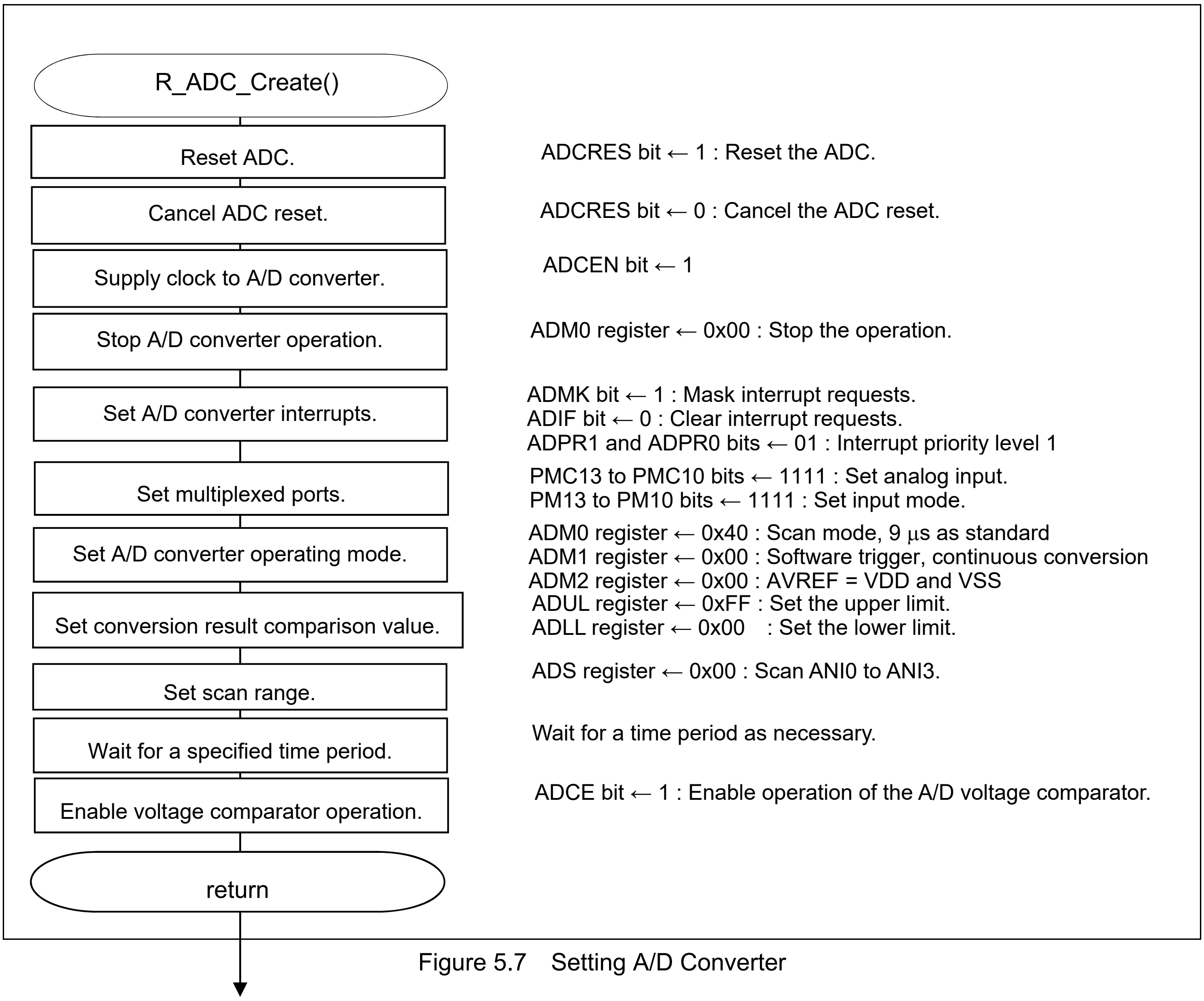

R_ADC_Create()
Reset ADC.
ADCRES bit ← 1 : Reset the ADC.
Cancel ADC reset.
ADCRES bit ← 0 : Cancel the ADC reset.
Supply clock to A/D converter.
ADCEN bit ← 1
Stop A/D converter operation.
ADM0 register ← 0x00 : Stop the operation.
Set A/D converter interrupts.
ADMK bit ← 1 : Mask interrupt requests.
ADIF bit ← 0 : Clear interrupt requests.
ADPR1 and ADPR0 bits ← 01 : Interrupt priority level 1
Set multiplexed ports.
PMC13 to PMC10 bits ← 1111 : Set analog input.
PM13 to PM10 bits ← 1111 : Set input mode.
Set A/D converter operating mode.
ADM0 register ← 0x40 : Scan mode, 9 μs as standard
ADM1 register ← 0x00 : Software trigger, continuous conversion
ADM2 register ← 0x00 : AVREF = VDD and VSS
Set conversion result comparison value.
ADUL register ← 0xFF : Set the upper limit.
ADLL register ← 0x00 : Set the lower limit.
Set scan range.
ADS register ← 0x00 : Scan ANI0 to ANI3.
Wait for a specified time period.
Wait for a time period as necessary.
Enable voltage comparator operation.
ADCE bit ← 1 : Enable operation of the A/D voltage comparator.
return

Figure 5.7 Setting A/D Converter

Resetting ADC

- Peripheral reset control register 0 (PRR0)
  Resets the ADC.

Symbol: PRR0

7	6		5	4	3	2	1	0
0	0		ADCRES	0	0	SAU0RES	0	TAU0RES
0	0		**1/0**	0	0	x	0	x

Bit 5

ADCRES	Control of A/D converter reset
0	**Cancels the reset of A/D converter.**
1	**Resets the A/D converter.**

Starting clock supply to A/D converter

- Peripheral enable register 0 (PER0)
  Starts supplying clock to the A/D converter.

Symbol: PER0

7	6	5	4	3	2	1	0
RTCEN	0	ADCEN	I2CA0EN	SAU1EN	SAU0EN	0	TAU0EN
x	0	**1**	x	x	x	0	x

Bit 5

ADCEN	Control of A/D converter input clock
0	入力クロック供給停止
1	**Supplies input clock.**

Stopping A/D converter operation

- A/D converter mode register 0 (ADM0)
  Stops A/D converter operation.

Symbol: ADM0

7	6	5	4	3	2	1	0
ADCS	ADMD	FR2	FR1	FR0	LV1	LV0	ADCE
0	x	x	x	x	x	x	**0**

Bit 7

ADCS	Control of A/D conversion operation
0	**Stops conversion operation.**
1	変換動作許可

Bit 0

ADCE	Control of A/D voltage comparator operation
0	**Stops A/D voltage comparator operation.**
1	A/D 電圧コンパレータの動作許可

Setting A/D conversion end interrupt

- ADMK bit in the interrupt mask flag register (MK1H)
  Set interrupt masks.
- ADIF bit in the interrupt request flag register (IF1H)
  Clear interrupt request flags.
- ADPR0 and ADPR1 bits in the priority order specification flag register (PR01H, PR11H)
  Set the A/D conversion end interrupt priority to level 1.

Bit 0

ADMK	Control of interrupt processing
0	割り込み処理許可
1	**Disables interrupt processing.**

Bit 0

ADIF	Interrupt request flag
0	**Interrupt request signal has not been generated.**
1	割り込み要求信号が発生し，割り込み要求状態

Bit 0

ADPR1	ADPR0	Selection of priority level.
0	0	レベル０を指定（高優先順位）
0	**1**	**Specifies level 1.**
1	0	レベル２を指定
1	1	レベル３を指定（低優先順位）

Setting multiplexed ports

- Port mode control register 1 (PMC1)
  Set the pins to analog input.
- Port mode register 1 (PM1)
  Turn off the output buffer of the port.

Symbol: PMC1

7	6	5	4	3	2	1	0
PMC17	PMC16	PMC15	PMC14	PMC13	PMC12	PMC11	PMC10
x	x	x	x	**1**	**1**	**1**	**1**

Bits 3 to 0

PMC1n	Digital IO/analog input selection for P1n pin (n = 0-7)
0	デジタル入出力(アナログ入力以外の兼用機能)
1	**Analog input**

Symbol: PM1

7	6	5	4	3	2	1	0
PM17	PM16	PM15	PM14	PM13	PM12	PM11	PM10
x	x	x	x	**1**	**1**	**1**	**1**

Bits 3 to 0

PM1n	P1n pin I/O mode selection
0	出力モード（出力バッファ・オン）
1	**Input mode (output buffer turned off)**

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

Setting A/D converter operating mode

- A/D converter mode register 0 (ADM0)
  Set the conversion operation to scan mode.
  Set the conversion time to 9 µs.
- A/D converter mode register 1 (ADM1)
  Set software trigger mode.
  Set continuous conversion mode.
- A/D converter mode register 2 (ADM2)
  Set the reference voltage.
  Set 12-bit resolution.

Symbol: ADM0

7	6	5	4	3	2	1	0
ADCS	ADMD	FR2	FR1	FR0	LV1	LV0	ADCE
0	**1**	**1**	**0**	**1**	**0**	**0**	**1**

Bit 6

ADMD	A/D conversion channel selection mode setting
0	セレクト・モード
1	**Scan mode**

Bits 5 to 1

FR2	FR1	FR0	LV1	LV0	Conversion time for 12-bit resolution				
					f_{CLK}= 1MHz	f_{CLK}= 4MHz	f_{CLK}= 8MHz	f_{CLK}= 16MHz	**f_{CLK}=** **24MHz**
0	0	0	**0**	**0**	設定禁止	設定禁止	設定禁止	設定禁止	72μs
0	0	1						54μs	38μs
0	1	0					54μs	27μs	18μs
0	1	1					40.5μs	20.25μs	13.5μs
1	0	0					33.75μs	16.875μs	11.25μs
1	**0**	**1**				54μs	27μs	13.5μs	**9 μs**
1	1	0				27μs	13.5μs	6.75μs	4.5μs
1	1	1			54μs	13.5μs	6.75μs	3.375μs	設定禁止
0	0	0	0	1	設定禁止	設定禁止	設定禁止	設定禁止	88μs
0	0	1						66μs	44μs
0	1	0					66μs	33μs	22μs
0	1	1					49.5μs	24.75μs	16.5μs
1	0	0					41.25μs	20.625μs	13.75μs
1	0	1				66μs	33μs	16.5μs	11μs
1	1	0				33μs	16.5μs	8.25μs	5.5μs
1	1	1			66μs	16.5μs	8.25μs	4.125μs	設定禁止

Symbol: ADM1

7	6	5	4	3	2	1	0
ADTMD1	ADTMD0	ADSCM	0	0	0	ADTRS1	ADTRS0
0	**0**	**0**	0	0	0	**0**	**0**

Bits 7 and 6

ADTMD1	ADTMD0	Selection of A/D conversion trigger mode
0	**x**	**Software trigger mode**
1	0	ハードウエア・トリガ・ノーウエイト・モード
1	1	ハードウエア・トリガ・ウエイト・モード

Bit 5

ADSCM	Selection of A/D conversion operating mode
0	**Continuous conversion mode**
1	ワンショット変換モード

Symbol: ADM1

7	6	5	4	3	2	1	0
ADTMD1	ADTMD0	ADSCM	0	0	0	ADTRS1	ADTRS0
0	**0**	**0**	0	0	0	**0**	**0**

Bits 1 and 0

ADTRS1	ADTRS0	Selection of hardware trigger signal
0	0	**Timer channel 1 count end or capture end interrupt signal (INTTM01)**
0	1	ELCで選択されたイベント信号
1	0	リアルタイム・クロック2割り込み信号(INTRTC)
1	1	2ビット・インターバル・タイマ割り込み信号(INTIT)

Symbol: ADM2

7	6	5	4	3	2	1	0
ADREFP1	ADREFP0	ADREFM	0	ADRCK	AWC	0	ADTYP
0	**0**	**0**	0	0	0	**0**	**0**

Bits 7 and 6

ADREFP1	ADREFP0	Selection of A/D converter plus-side reference voltage
0	**0**	**Supplied from AVDD**
0	1	AVREFP/ANI0から供給
1	0	内部基準電圧（1.45 V）から供給
1	1	設定禁止

Bit 5

ADREFM	Selection of A/D converter minus-side reference voltage
0	**Supplied from AVss**.
1	AVREFM/ANI1から供給

Bit 3

ADRCK	Check of conversion result upper-limit/lower-limit value
0	**Interrupt signal (INTAD) is generated when ADLL register ≤ ADCR register ≤ ADUL register (AREA1)**
1	ADCRレジスタ＜ADLLレジスタ（AREA2），ADULレジスタ＜ADCRレジスタ（AREA3）のとき割り込み信号（INTAD）が発生

Bit 2

AWC	SNOOZE mode setting
0	**SNOOZE mode not used**
1	SNOOZEモード機能を使用する

Bit 0

ADTYP	Selection of A/D conversion resolution
0	**12-bit resolution**
1	8ビット分解能

Note: For details of register settings, refer to the RL78/I1D User's Manual: Hardware.

Setting conversion result upper limit and lower limit

- Conversion result comparison upper limit setting register (ADUL)
  Set the upper limit value.
- Conversion result comparison lower limit setting register (ADLL)
  Sets the lower limit value.

Symbol: ADUL

7	6	5	4	3	2	1	0
ADUL7	ADUL6	ADUL5	ADUL4	ADUL3	ADUL2	ADUL1	ADUL0
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Symbol: ADLL

7	6	5	4	3	2	1	0
ADLL7	ADLL6	ADLL5	ADLL4	ADLL3	ADLL2	ADLL1	ADLL0
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Setting A/D conversion channels

- Analog input channel specification register (ADS)
  Set ANI0 to ANI3.

Symbol: ADS

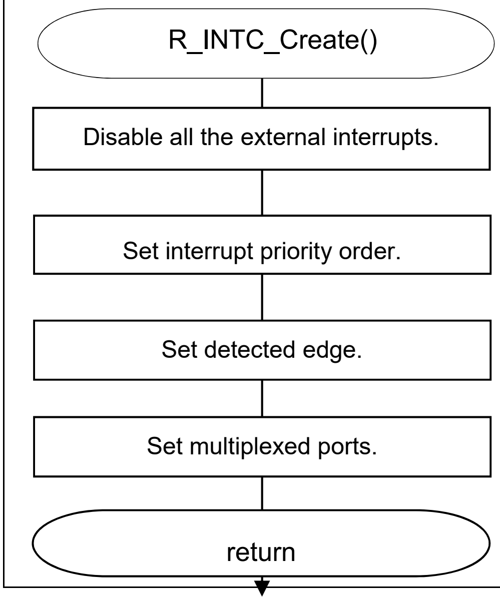
7	6	5	4	3	2	1	0
ADISS	0	0	ADS4	ADS3	ADS2	ADS1	ADS0
0	0	0	**0**	**0**	**0**	**0**	**0**

Bits 4 to 0

ADS4	ADS3	ADS2	ADS1	ADS0	Analog input channel
0	**0**	**0**	**0**	**0**	**ANI0—ANI3**
0	0	0	0	1	ANI1—ANI4
0	0	0	1	0	ANI2—ANI5
0	0	0	1	1	ANI3—ANI6
The rest of the combinations are omitted.					

5.7.7 Setting External Interrupts

Figure 5.8 shows the flowchart for setting the external interrupts.


R_INTC_Create()
Disable all the external interrupts.
Set interrupt priority order.
Set detected edge.
Set multiplexed ports.
return

PMK6 to PMK0 bits ← 1 : Mask all the interrupt requests.
PIF6 to PIF0 bits ← 0 : Clear the interrupt requests.

PPR15, PPR05 bits ← 0, 0 : Set INTP5 to the highest priority.
PPR16, PPR06 bits ← 0, 0 : Set INTP6 to the highest priority.

EGN0 register ← 0x40 : Detect INTP6 falling edge.
EGP0 register ← 0x60 : Detect INTP5 and INTP6 rising edges.

PM3.3 and PM3.2 bits ← 1, 1 : Turns the output buffer off.

Figure 5.8 Setting External Interrupts

Disabling all the external interrupts

- Interrupt mask flag register (MK0L, MK0H)
  Mask interrupt requests.
- Interrupt request flag register (IF0L, IF0H)
  Clear interrupt requests.

Symbol: PMK0L

7	6	5	4	3	2	1	0
PMK5	PMK4	PMK3	PMK2	PMK1	PMK0	LVIMK	WDTIMK
1	**1**	**1**	**1**	**1**	**1**	x	x

Symbol: PMK0H

7	6	5	4	3	2	1	0
RTITMK	TMMK00	SREMK0	1	1	SRMK0	STMK0	PMK6
x	x	x	1	1	x	x	**1**

Bit n

PMKn	Control of interrupt processing
0	割り込み処理許可
1	**Disables interrupt processing.**

Symbol: PIF0L

7	6	5	4	3	2	1	0
PIF5	PIF4	PIF3	PIF2	PIF1	PIF0	LVIIF	WDTIIF
0	**0**	**0**	**0**	**0**	**0**	x	x

Symbol: PIF0H

7	6	5	4	3	2	1	0
RTITIF	TMIF00	SREIF0	0	0	SRIF0	STIF0	PIF6
x	x	x	0	0	x	x	**0**

Bit n

PIFn	Interrupt request flag
0	**Interrupt request signal has not been generated.**
1	割り込み要求信号が発生し，割り込み要求状態

Setting interrupt priority level

- PPR15, PPR15, PPR16, and PPR06 bits in the priority order specification flag register
  Set the highest priority level.

Bit 7

PPR15	PPR05	Selection of INTP5 priority level
0	**0**	**Sets level 0 (highest priority)**
0	1	レベル 1 を指定
	0	レベル 2 を指定
1	1	レベル 3 を指定（低優先順位）

Bit 0

PPR16	PPR06	Selection of INTP6 priority level
0	**0**	**Sets level 0 (highest priority)**
0	1	レベル 1 を指定
	0	レベル 2 を指定
1	1	レベル 3 を指定（低優先順位）

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

Setting edge detection

- External interrupt rising edge enable register (EGP0)
  Enable INTP5 and INTP6 edge detection.
- External interrupt falling edge enable register (EGN0)
  Enable INTP6 edge detection.

Symbol: EGP0

7	6	5	4	3	2	1	0
0	EGP6	EGP5	EGP4	EGP3	EGP2	EGP1	EGP0
0	**1**	**1**	**0**	**0**	**0**	**0**	**0**

Symbol: EGN0

7	6	5	4	3	2	1	0
0	EGN6	EGN5	EGN4	EGN3	EGN2	EGN1	EGN0
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**

Bits 6 and 5

EGPn	EGNn	Selection of valid edge of INTPn pin
0	0	エッジ検出禁止
0	1	立ち下がりエッジ
1	**0**	**Rising edge**
1	**1**	**Both of rising and falling edges**

Setting multiplexed ports

- PM3.3 and PM3.2 bits in port mode register3 (PM3)
  Turn the output buffer off.

Symbol: PM3

7	6	5	4	3	2	1	0
1	1	1	1	PM33	PM32	PM31	PM30
1	1	1	1	**1**	**1**	x	x

Bits 3 and 2

PM3n	P3n pin I/O mode selection
0	出力モード(出力ポートとして機能(出力バッファ・オン))
1	**Input mode (functions as an input port (output buffer turned off).)**

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

5.7.8 Main Process

Figure 5.9 shows the flowchart of the main process.

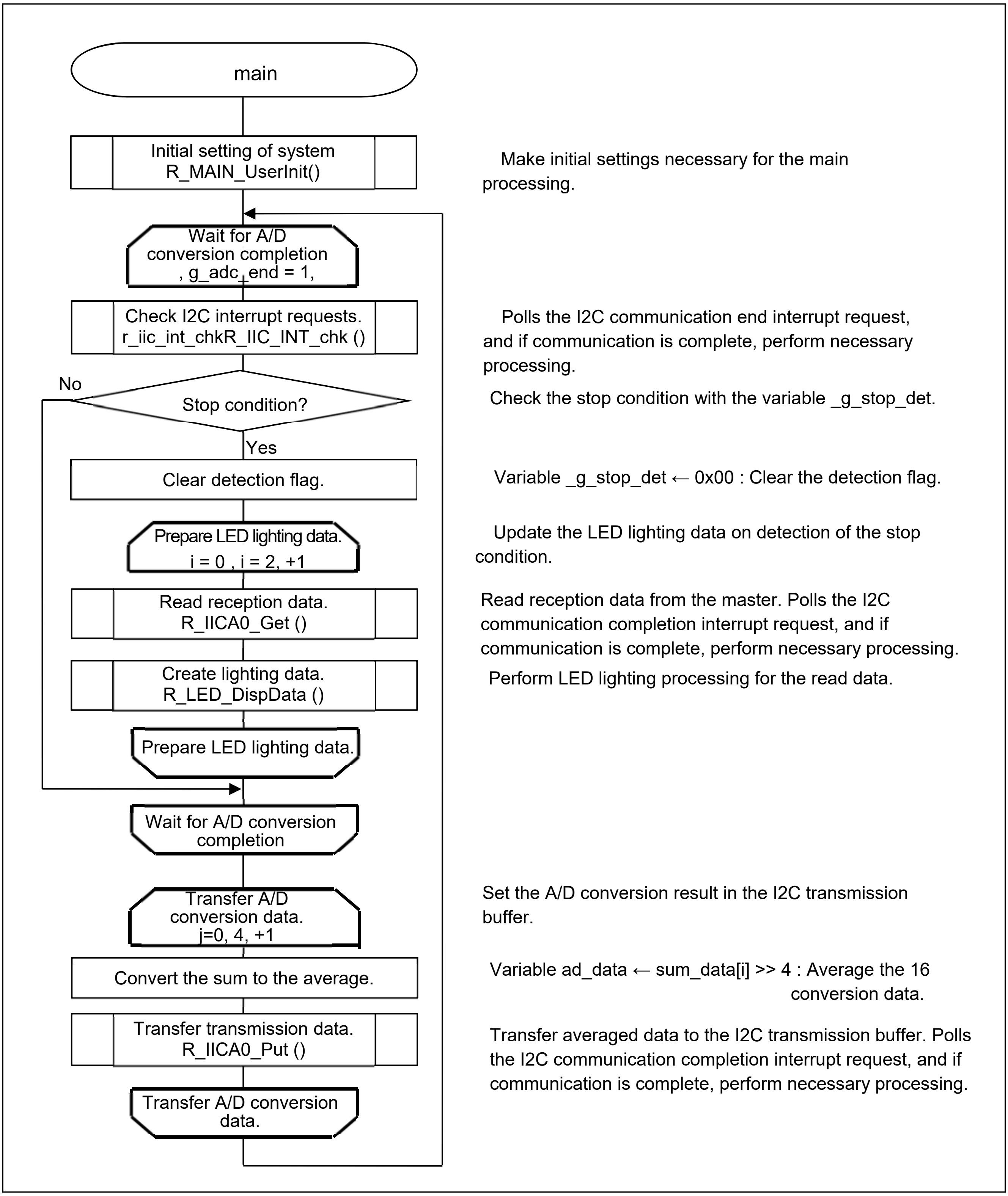

main
Initial setting of system
R_MAIN_UserInit()
Make initial settings necessary for the main processing.
Wait for A/D conversion completion
, g_adc_end = 1,
Check I2C interrupt requests.
r_iic_int_chkR_IIC_INT_chk ()
Polls the I2C communication end interrupt request, and if communication is complete, perform necessary processing.
No
Stop condition?
Check the stop condition with the variable _g_stop_det.
Yes
Clear detection flag.
Variable _g_stop_det ← 0x00 : Clear the detection flag.
Prepare LED lighting data.
i = 0 , i = 2, +1
Update the LED lighting data on detection of the stop condition.
Read reception data.
R_IICA0_Get ()
Read reception data from the master. Polls the I2C communication completion interrupt request, and if communication is complete, perform necessary processing.
Create lighting data.
R_LED_DispData ()
Perform LED lighting processing for the read data.
Prepare LED lighting data.
Wait for A/D conversion completion
Transfer A/D conversion data.
j=0, 4, +1
Set the A/D conversion result in the I2C transmission buffer.
Convert the sum to the average.
Variable ad_data ← sum_data[i] >> 4 : Average the 16 conversion data.
Transfer transmission data.
R_IICA0_Put ()
Transfer averaged data to the I2C transmission buffer. Polls the I2C communication completion interrupt request, and if communication is complete, perform necessary processing.
Transfer A/D conversion data.

Figure 5.9 Main Process

5.7.9 R_MAIN_UserInit Process

Figure 5.10 shows the flowchart of the R_MAIN_UserInit process.

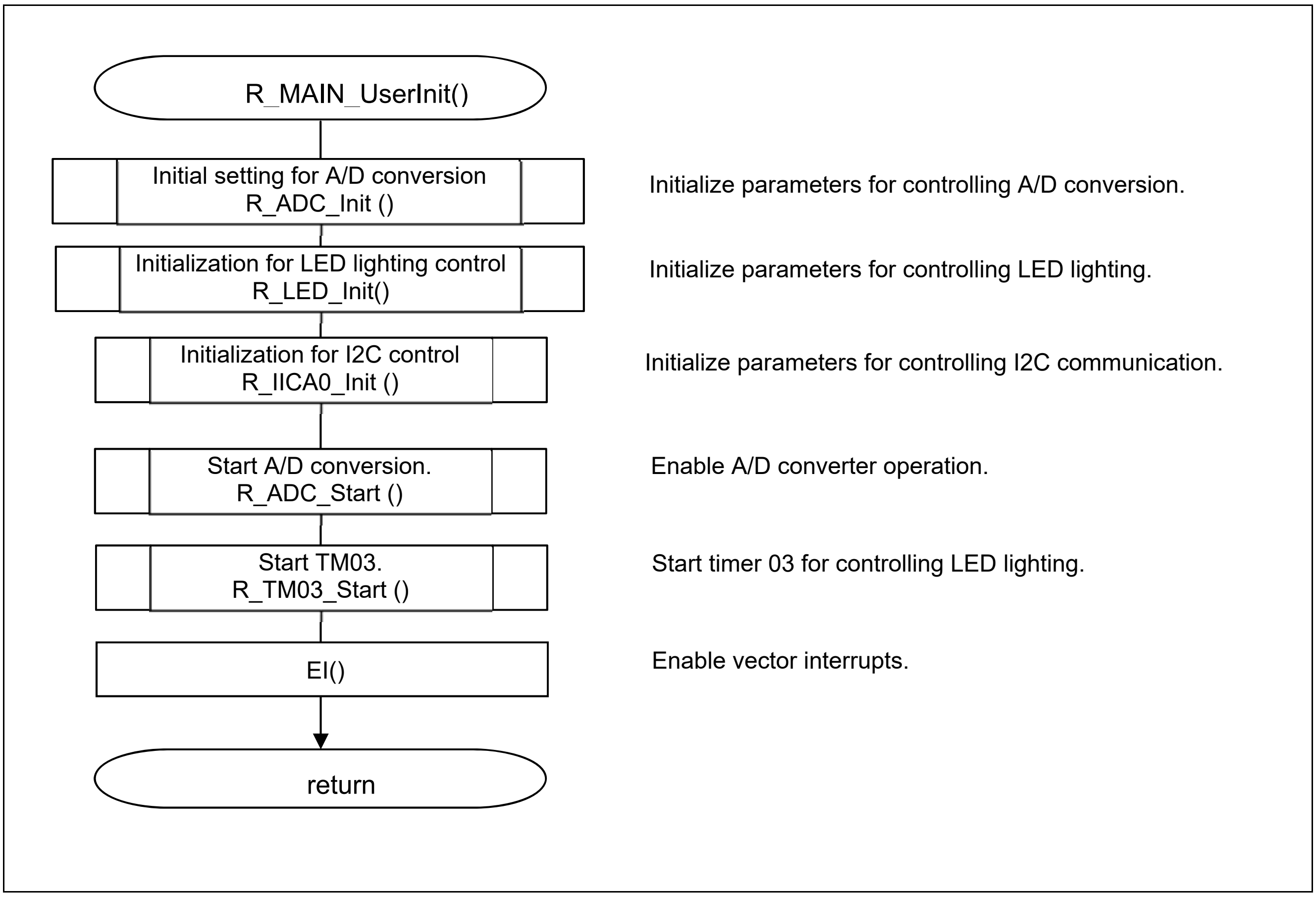

R_MAIN_UserInit()
Initial setting for A/D conversion
R_ADC_Init ()
Initialize parameters for controlling A/D conversion.
Initialization for LED lighting control
R_LED_Init()
Initialize parameters for controlling LED lighting.
Initialization for I2C control
R_IICA0_Init ()
Initialize parameters for controlling I2C communication.
Start A/D conversion.
R_ADC_Start ()
Enable A/D converter operation.
Start TM03.
R_TM03_Start ()
Start timer 03 for controlling LED lighting.
EI()
Enable vector interrupts.
return

Figure 5.10 R_MAIN_UserInit Process

5.7.10 Initial Setting of A/D Conversion

Figure 5.11 shows the flowchart for making A/D conversion initial settings.

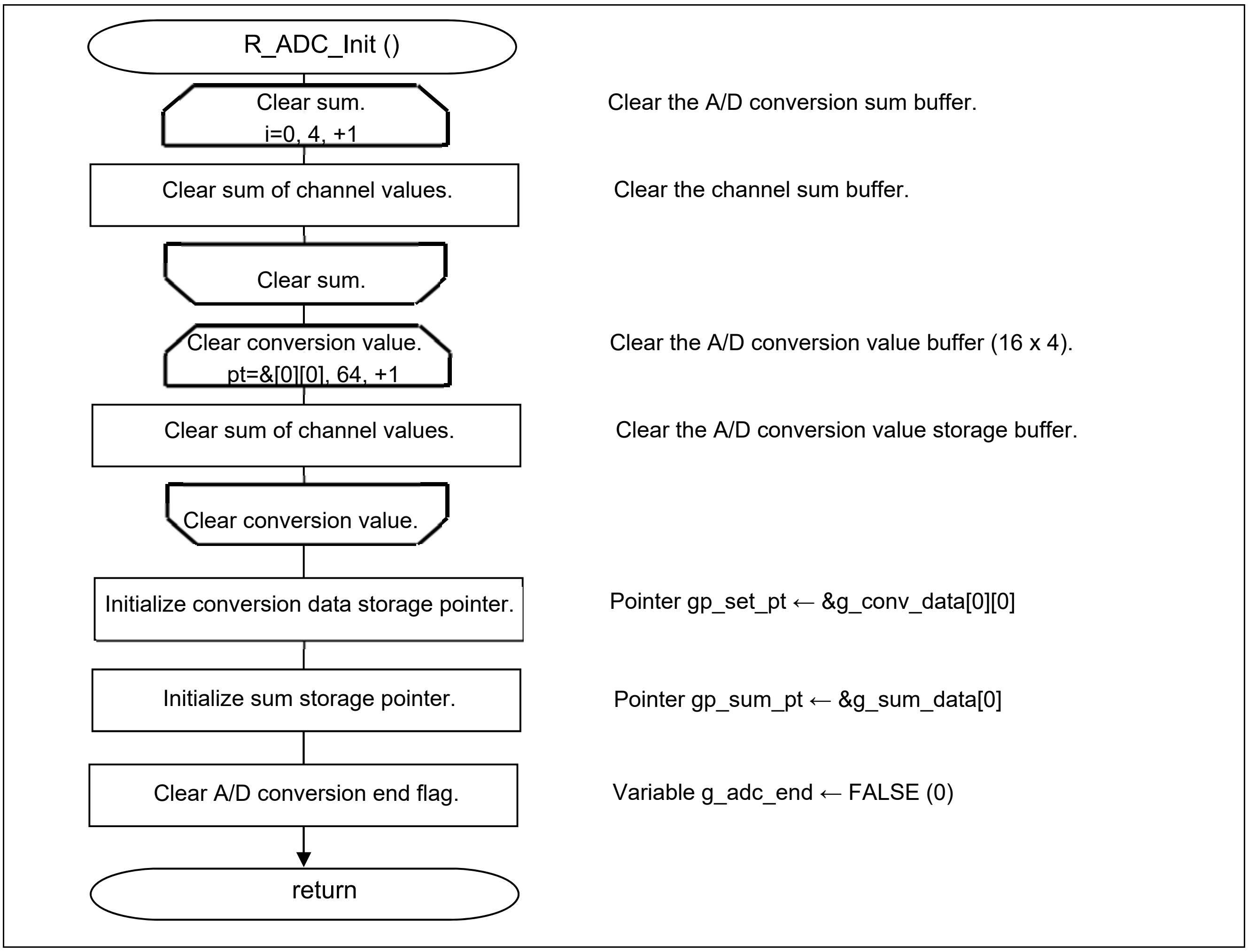

R_ADC_Init ()
Clear sum.
i=0, 4, +1
Clear the A/D conversion sum buffer.
Clear sum of channel values.
Clear the channel sum buffer.
Clear sum.
Clear conversion value.
pt=&[0][0], 64, +1
Clear the A/D conversion value buffer (16 x 4).
Clear sum of channel values.
Clear the A/D conversion value storage buffer.
Clear conversion value.
Initialize conversion data storage pointer.
Pointer gp_set_pt ← &g_conv_data[0][0]
Initialize sum storage pointer.
Pointer gp_sum_pt ← &g_sum_data[0]
Clear A/D conversion end flag.
Variable g_adc_end ← FALSE (0)
return

Figure 5.11 A/D Conversion Initial Settings

5.7.11 Starting A/D Conversion

Figure 5.12 shows the flowchart for starting A/D conversion.

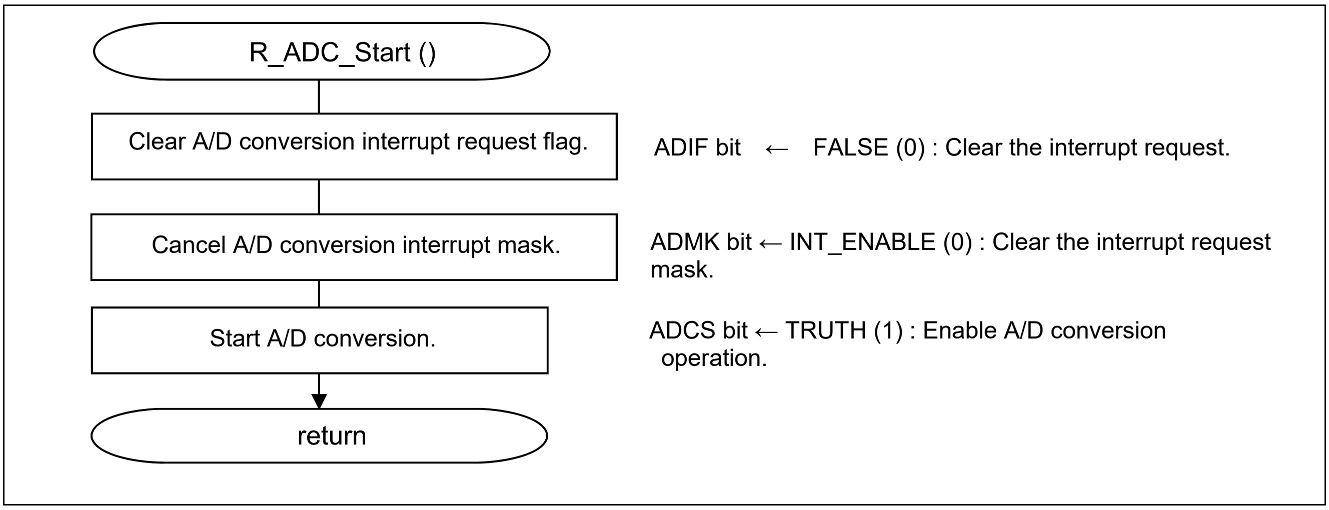

R_ADC_Start ()
Clear A/D conversion interrupt request flag.
ADIF bit ← FALSE (0) : Clear the interrupt request.
Cancel A/D conversion interrupt mask.
ADMK bit ← INT_ENABLE (0) : Clear the interrupt request mask.
Start A/D conversion.
ADCS bit ← TRUTH (1) : Enable A/D conversion operation.
return

Figure 5.12 Starting A/D Conversion

Setting A/D conversion interrupts

- ADIF bit in the interrupt request flag register 1H (IF1H)
  Clear the interrupt request.
- ADMK bit in the interrupt request mask flag register 1H (MK1H)
  Cancel the interrupt request mask.

Bit 0

ADIF	Interrupt request flag
0	**Interrupt request signal has not been generated.**
1	割り込み要求信号が発生し，割り込み要求状態

Bit 0

ADMK	Interrupt request flag
0	**Enables interrupt processing.**
1	割り込み処理禁止

Starting A/D conversion

- A/D converter mode register 0 (ADM0)
  Enable A/D converter operation.

Symbol: ADM0

7	6	5	4	3	2	1	0
ADCS	ADMD	FR2	FR1	FR0	LV1	LV0	ADCE
1	1	1	0	1	0	0	1

Bit 7

ADCS	Control of A/D conversion operation
0	変換動作停止
1	**Enables conversion operation.**

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

5.7.12 A/D Conversion End interrupt Process

Figure 5.13 shows the flowchart of the A/D conversion end interrupt process.

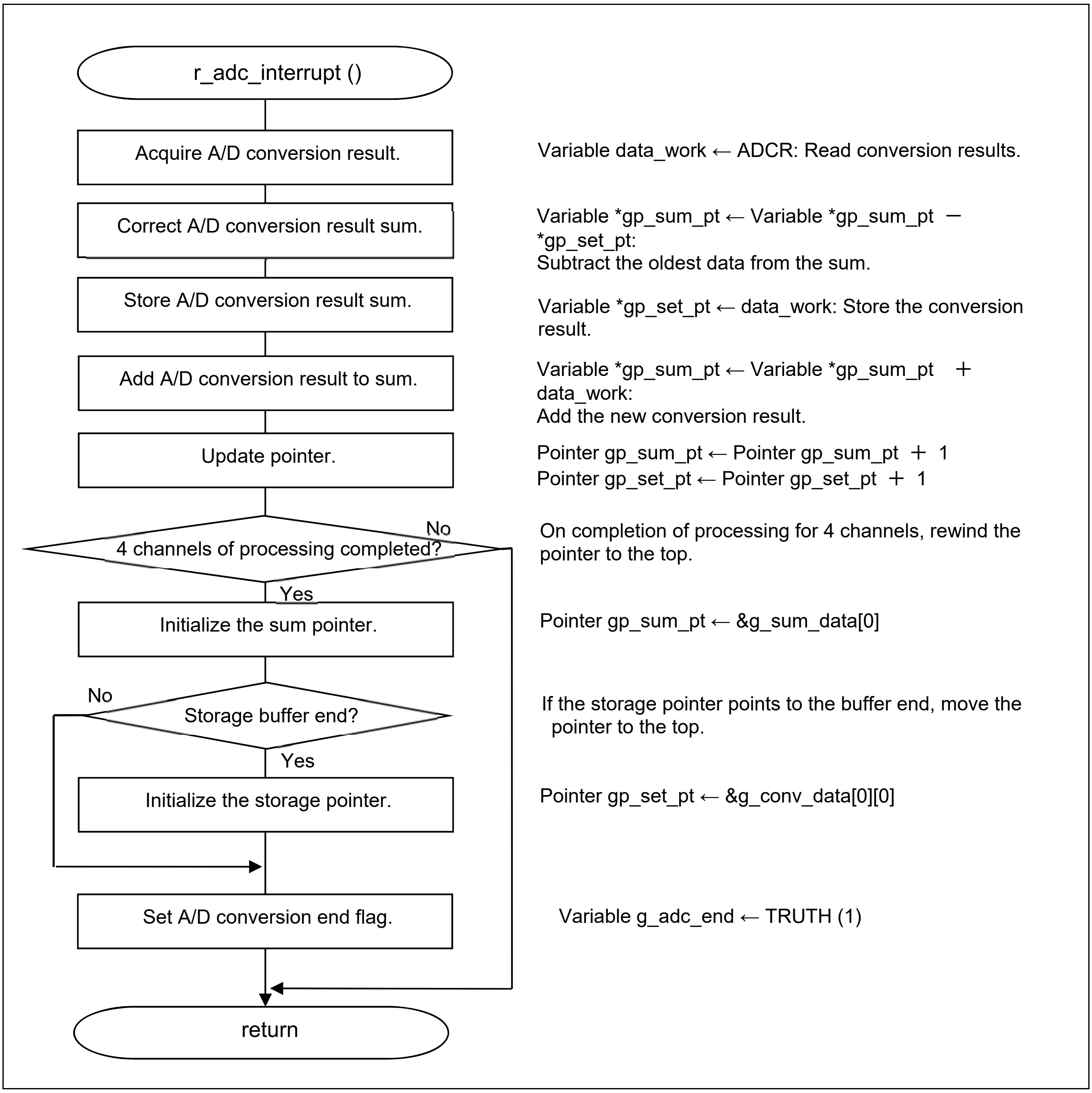

r_adc_interrupt ()
Acquire A/D conversion result.
Correct A/D conversion result sum.
Store A/D conversion result sum.
Add A/D conversion result to sum.
Update pointer.
4 channels of processing completed?
No
Yes
Initialize the sum pointer.
Storage buffer end?
No
Yes
Initialize the storage pointer.
Set A/D conversion end flag.
return
Variable data_work ← ADCR: Read conversion results.
Variable *gp_sum_pt ← Variable *gp_sum_pt − *gp_set_pt:
Subtract the oldest data from the sum.
Variable *gp_set_pt ← data_work: Store the conversion result.
Variable *gp_sum_pt ← Variable *gp_sum_pt + data_work:
Add the new conversion result.
Pointer gp_sum_pt ← Pointer gp_sum_pt + 1
Pointer gp_set_pt ← Pointer gp_set_pt + 1
On completion of processing for 4 channels, rewind the pointer to the top.
Pointer gp_sum_pt ← &g_sum_data[0]
If the storage pointer points to the buffer end, move the pointer to the top.
Pointer gp_set_pt ← &g_conv_data[0][0]
Variable g_adc_end ← TRUTH (1)

Figure 5.13 A/D Conversion End interrupt Processing

Acquiring A/D conversion results

- 12-bit A/D conversion result register (ADCR)
  Read the A/D conversion results.

Symbol: ADCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	bit11	bit10	bit9	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

5.7.13 Initializing LED Lighting

Figure 5.14 shows the flowchart for initializing the LED lighting.

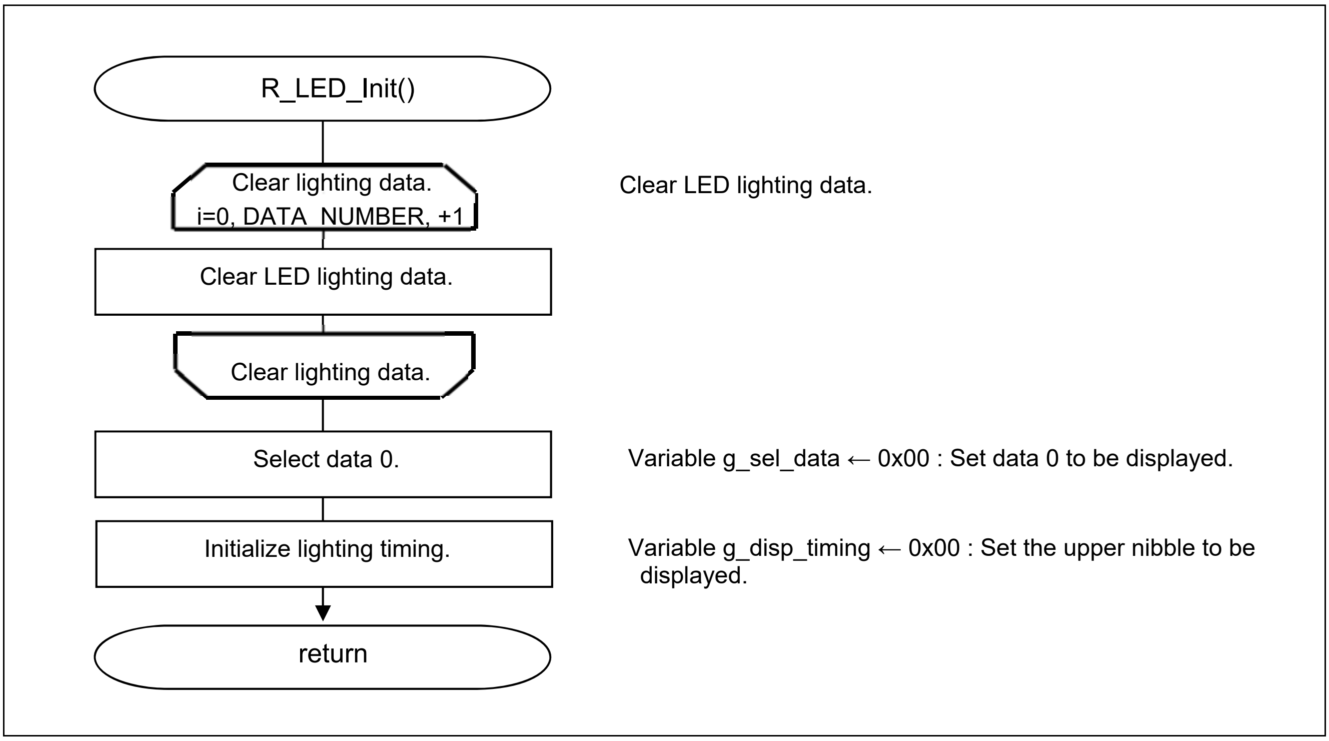

R_LED_Init()
Clear lighting data.
i=0, DATA_NUMBER, +1
Clear LED lighting data.
Clear lighting data.
Select data 0.
Initialize lighting timing.
return
Clear LED lighting data.
Variable g_sel_data ← 0x00 : Set data 0 to be displayed.
Variable g_disp_timing ← 0x00 : Set the upper nibble to be displayed.

Figure 5.14 Initialization for LED Lighting

5.7.14 Starting TM03

Figure 5.15 shows the flowchart for starting the TM03.

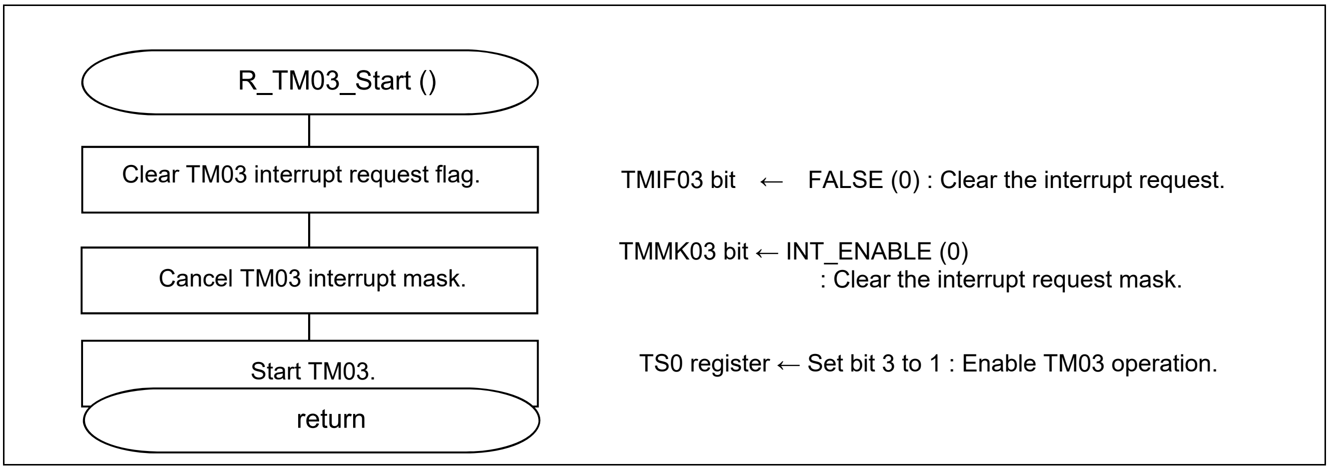

R_TM03_Start ()
Clear TM03 interrupt request flag.
Cancel TM03 interrupt mask.
Start TM03.
return
TMIF03 bit ← FALSE (0) : Clear the interrupt request.
TMMK03 bit ← INT_ENABLE (0)
: Clear the interrupt request mask.
TS0 register ← Set bit 3 to 1 : Enable TM03 operation.

Figure 5.15 Starting TM03

Setting TM03 interrupt

- TMIF03 bit in interrupt request flag register 1L (IF1L)
  Clear the interrupt request.
- TMMK03 bit in interrupt request mask flag register 1L (MK1L)
  Cancel the interrupt request mask.

Bit 5

TMIF03	Interrupt request flag
0	**Interrupt request signal has not been generated.**
1	割り込み要求信号が発生し，割り込み要求状態

Bit 5

TMMK03	Interrupt request flag
0	**Enables interrupt processing.**
1	割り込み処理禁止

Starting TM03

- Timer channel start register 0 (TS0)
  Enable TM03 operation.

Symbol: TS0L

7	6	5	4	3	2	1	0
TS07	TS06	TS05	TS04	TS03	TS02	TS01	TS00
0	0	0	0	**1**	0	0	0

Bit 3

TS03	Channel 3 operation enable (start) trigger
0	トリガ動作しない
1	**Sets TE03 bit to 1 to enable count operation.**

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

5.7.15 Setting LED Light-Emitting Data

Figure 5.16 shows the flowchart for setting the LED light-emitting data.

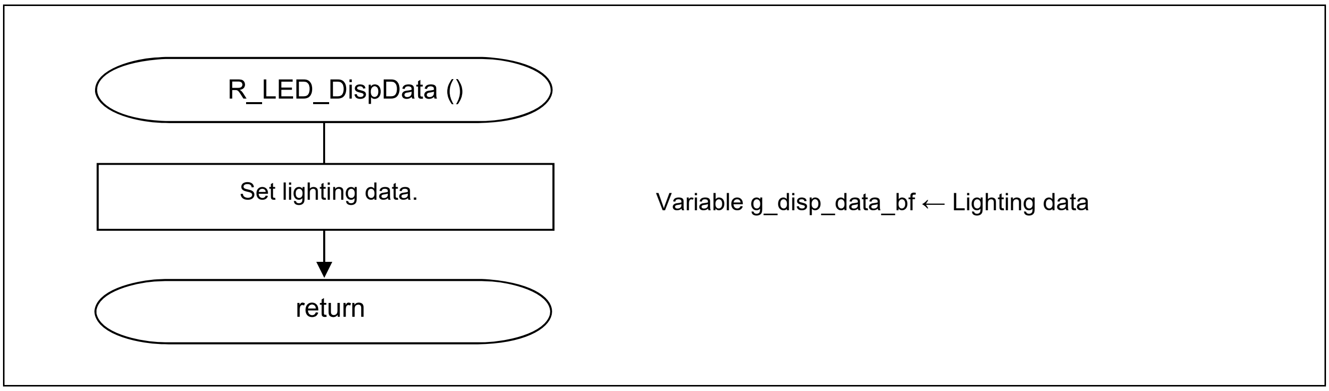


Figure 5.16 Setting LED Light-Emitting Data

5.7.16 5-ms Interval Timer Interrupt Process

Figure 5.17 shows the flowchart of the 5-ms interval timer interrupt process.

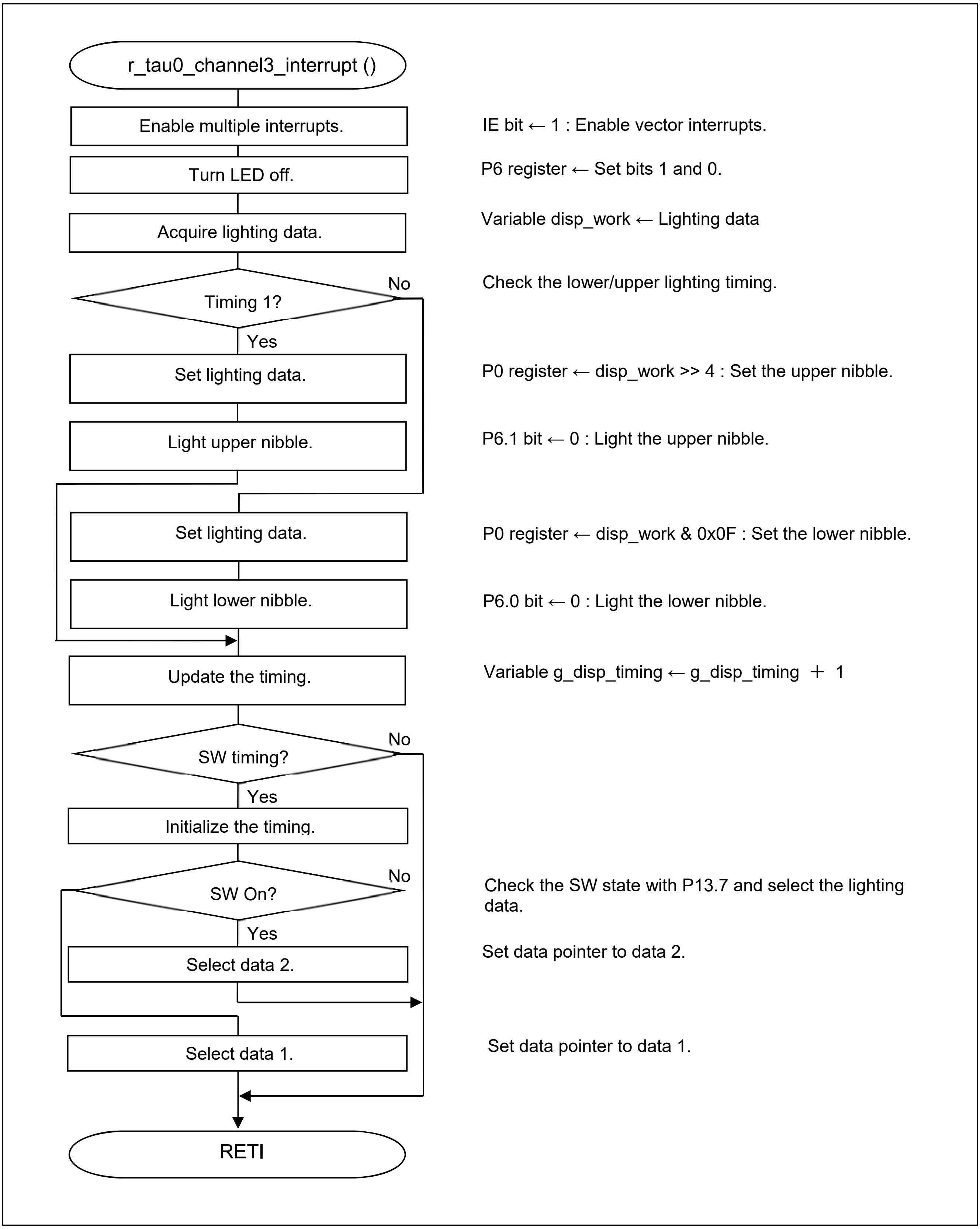

r_tau0_channel3_interrupt ()
Enable multiple interrupts.
IE bit ← 1 : Enable vector interrupts.
Turn LED off.
P6 register ← Set bits 1 and 0.
Acquire lighting data.
Variable disp_work ← Lighting data
Timing 1?
No
Yes
Check the lower/upper lighting timing.
Set lighting data.
P0 register ← disp_work >> 4 : Set the upper nibble.
Light upper nibble.
P6.1 bit ← 0 : Light the upper nibble.
Set lighting data.
P0 register ← disp_work & 0x0F : Set the lower nibble.
Light lower nibble.
P6.0 bit ← 0 : Light the lower nibble.
Update the timing.
Variable g_disp_timing ← g_disp_timing ＋ 1
SW timing?
No
Yes
Initialize the timing.
SW On?
No
Yes
Check the SW state with P13.7 and select the lighting data.
Select data 2.
Set data pointer to data 2.
Select data 1.
Set data pointer to data 1.
RETI

Figure 5.17 5-ms Interval Timer Interrupt Process

5.7.17 Initializing I2C Communication

Figure 5.18 shows the flowchart for initializing the I2C communication.

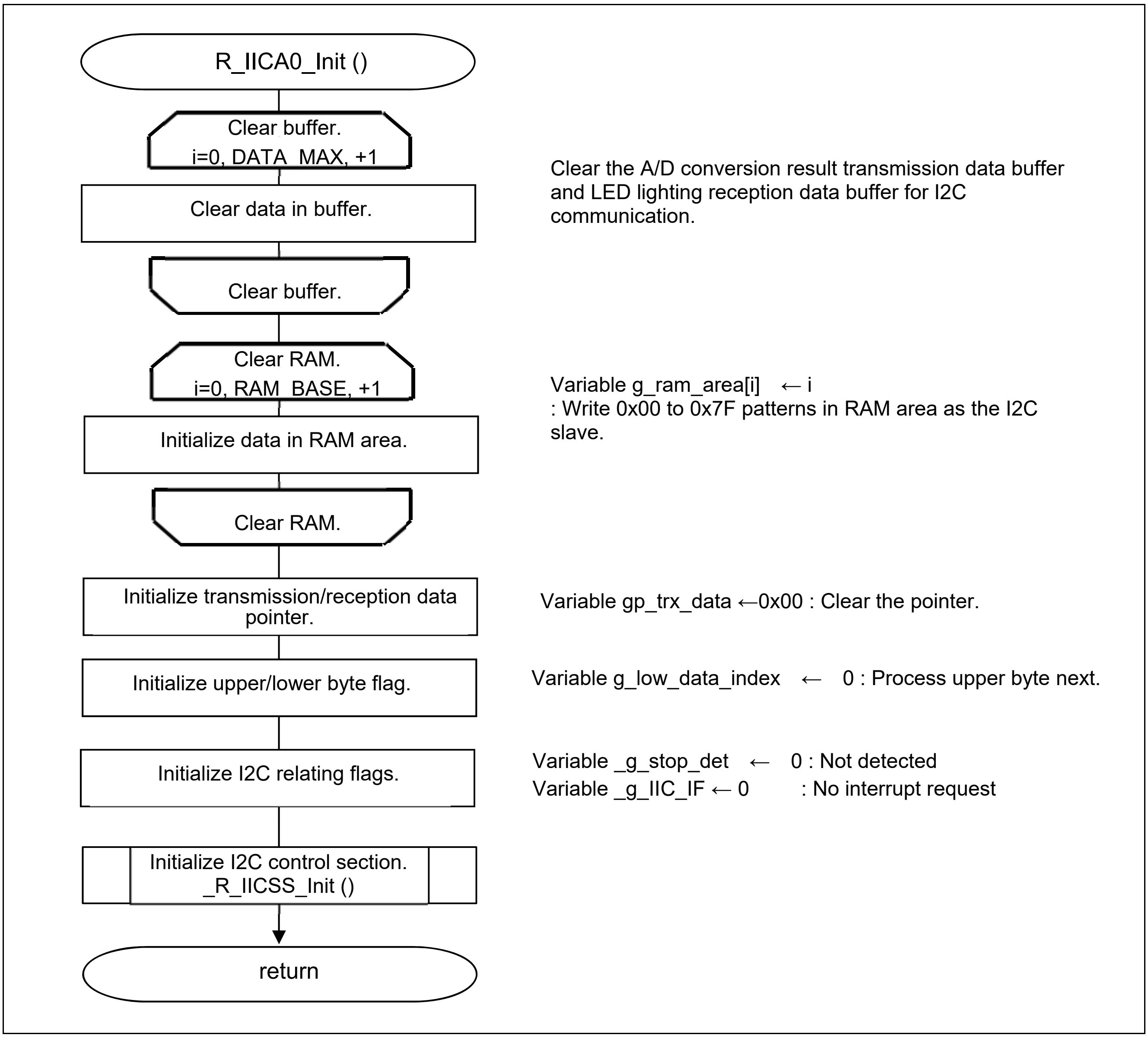

R_IICA0_Init ()
Clear buffer.
i=0, DATA_MAX, +1
Clear data in buffer.
Clear buffer.
Clear RAM.
i=0, RAM_BASE, +1
Initialize data in RAM area.
Clear RAM.
Initialize transmission/reception data pointer.
Initialize upper/lower byte flag.
Initialize I2C relating flags.
Initialize I2C control section.
_R_IICSS_Init ()
return
Clear the A/D conversion result transmission data buffer and LED lighting reception data buffer for I2C communication.
Variable g_ram_area[i] ← i
: Write 0x00 to 0x7F patterns in RAM area as the I2C slave.
Variable gp_trx_data ←0x00 : Clear the pointer.
Variable g_low_data_index ← 0 : Process upper byte next.
Variable _g_stop_det ← 0 : Not detected
Variable _g_IIC_IF ← 0 : No interrupt request

Figure 5.18 Initializing I2C Communication

5.7.18 Checking I2C Communication State

Figure 5.19 shows the flowchart for checking the I2C communication state.

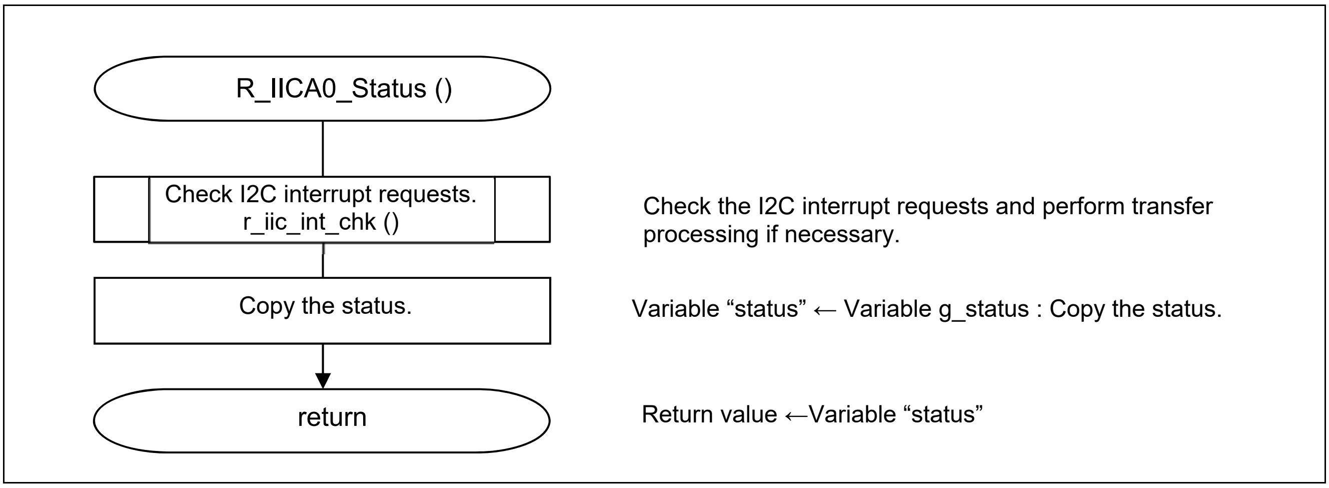

R_IICA0_Status ()
Check I2C interrupt requests.
r_iic_int_chk ()
Check the I2C interrupt requests and perform transfer processing if necessary.
Copy the status.
Variable “status” ← Variable g_status : Copy the status.
return
Return value ←Variable “status”

Figure 5.19 Checking I2C Communication State

5.7.19 Reading I2C Reception Data

Figure 5.20 shows the flowchart for reading I2C reception data.

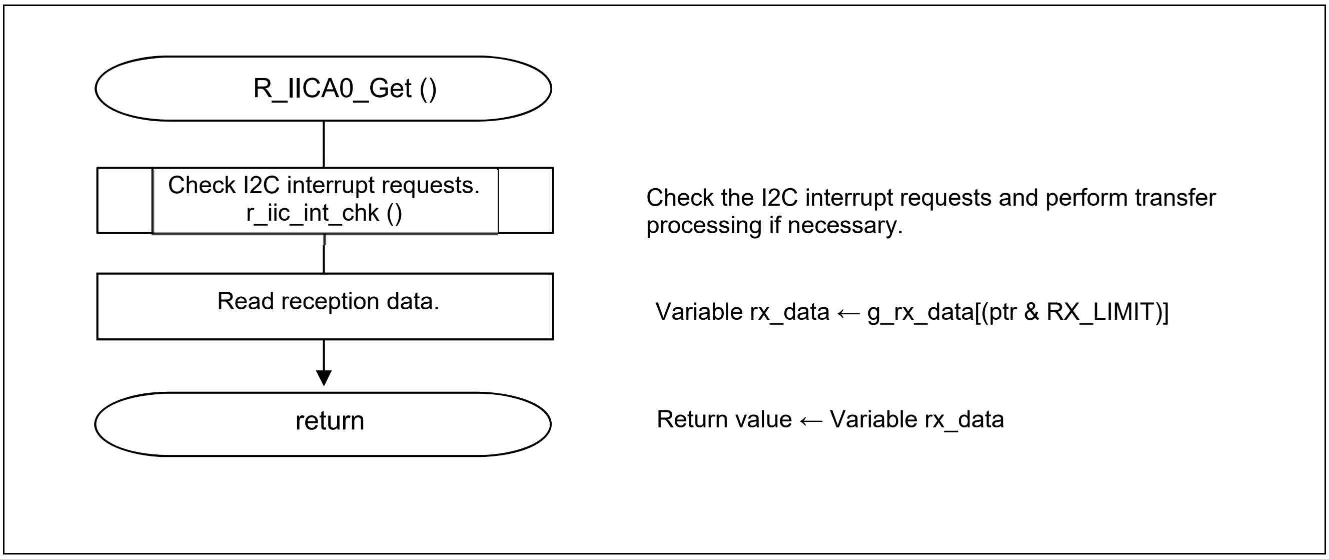

R_IICA0_Get ()
Check I2C interrupt requests.
r_iic_int_chk ()
Check the I2C interrupt requests and perform transfer processing if necessary.
Read reception data.
Variable rx_data ← g_rx_data[(ptr & RX_LIMIT)]
return
Return value ← Variable rx_data

Figure 5.20 Reading I2C Receive Data

5.7.20 Setting Data in I2C Transmission Buffer

Figure 5.21 shows the flowchart for setting data in the I2C transmission buffer.

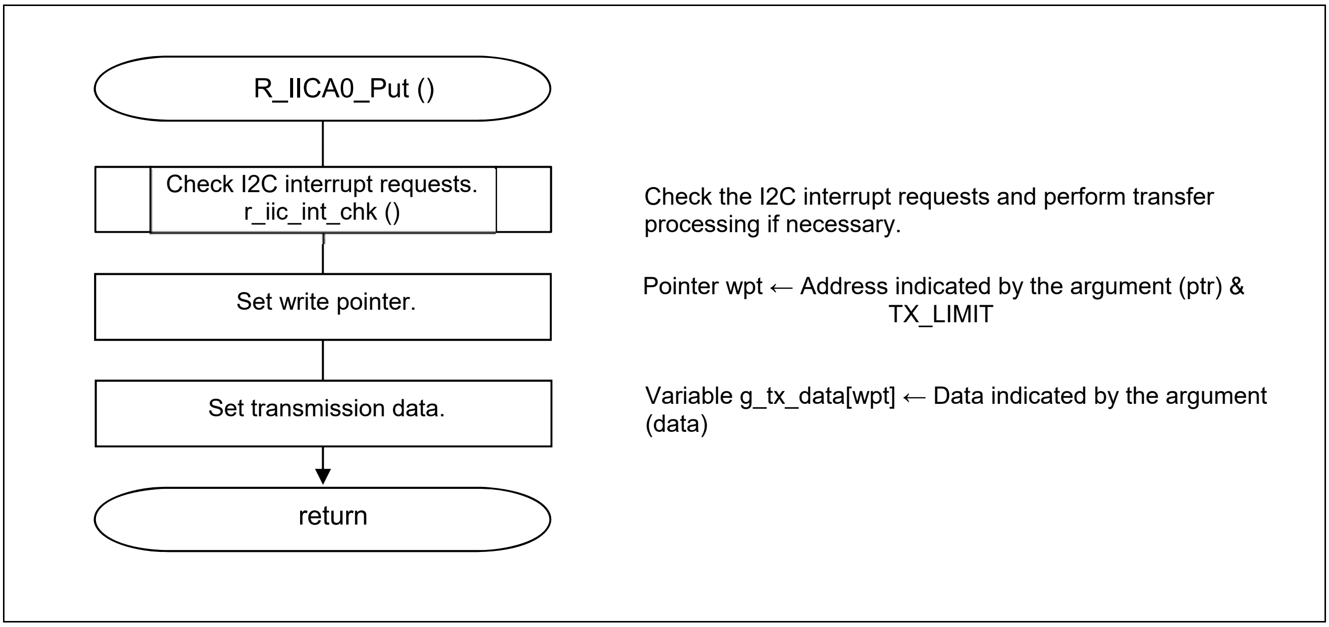

R_IICA0_Put ()
Check I2C interrupt requests.
r_iic_int_chk ()
Check the I2C interrupt requests and perform transfer processing if necessary.
Set write pointer.
Pointer wpt ← Address indicated by the argument (ptr) & TX_LIMIT
Set transmission data.
Variable g_tx_data[wpt] ← Data indicated by the argument (data)
return

Figure 5.21 Setting Data in I2C Transmit Buffer

5.7.21 Checking I2C Communication End interrupt Request

Figure 5.22 shows the flowchart for checking the I2C communication end interrupt request.

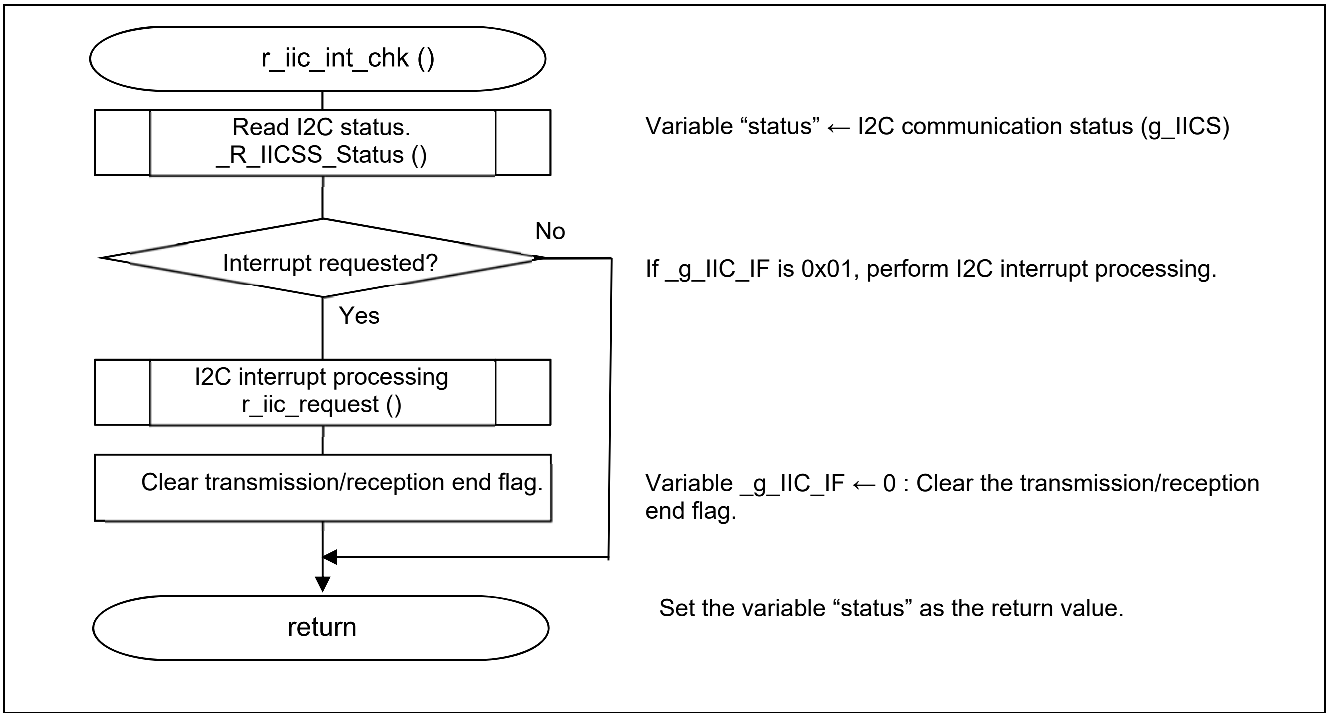

r_iic_int_chk ()
Read I2C status.
_R_IICSS_Status ()
Variable "status" ← I2C communication status (g_IICS)
Interrupt requested?
No
Yes
If _g_IIC_IF is 0x01, perform I2C interrupt processing.
I2C interrupt processing
r_iic_request ()
Clear transmission/reception end flag.
Variable _g_IIC_IF ← 0 : Clear the transmission/reception end flag.
return
Set the variable "status" as the return value.

Figure 5.22 I2CA0 Communication End interrupt Reception Process

5.7.22 I2C Communication End interrupt Process

Figures 5.23 to 5.26 show the flowcharts of the I2C communication end interrupt process.

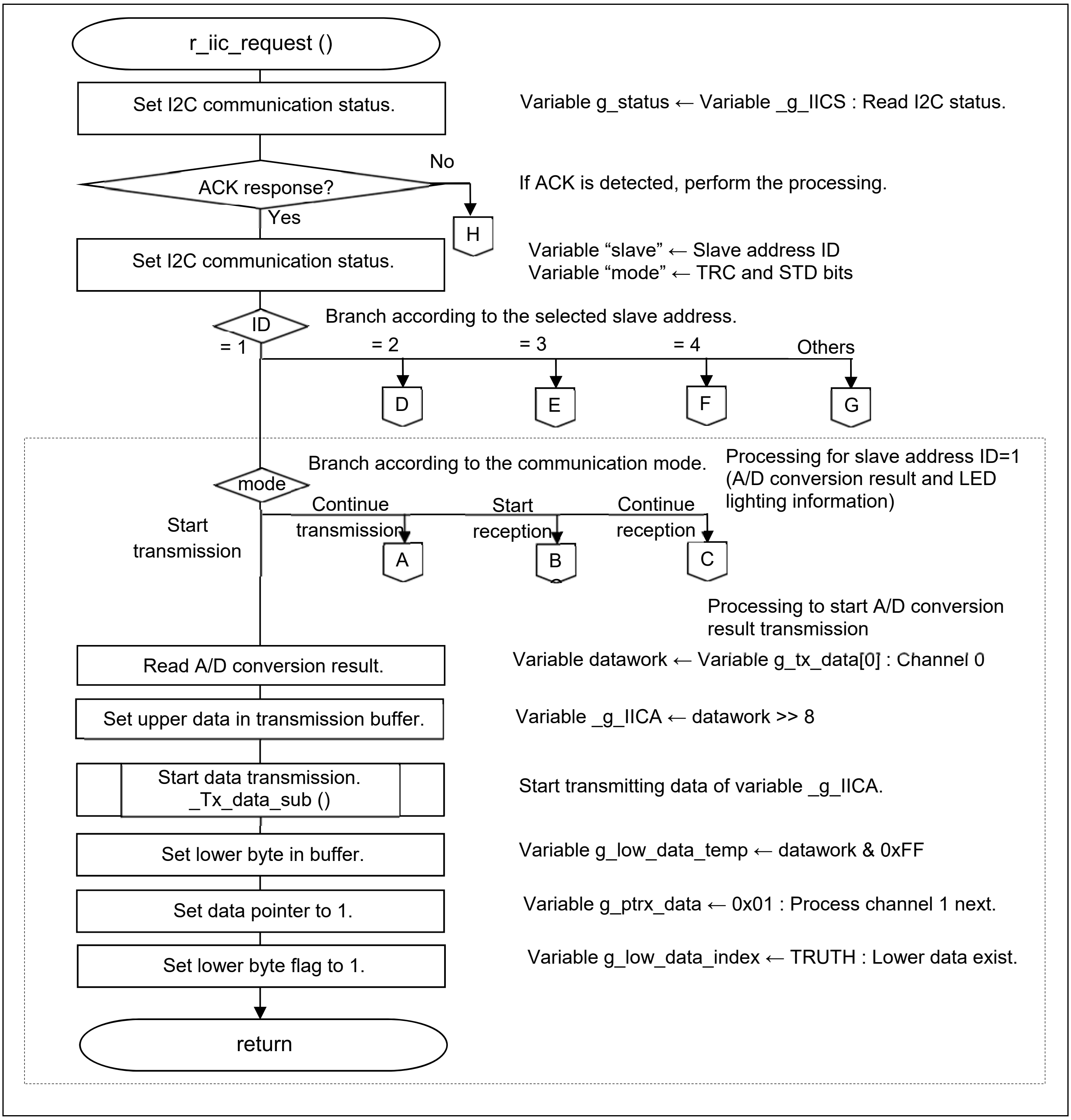

r_iic_request ()
Set I2C communication status.
Variable g_status ← Variable _g_IICS : Read I2C status.
ACK response?
No
Yes
H
If ACK is detected, perform the processing.
Set I2C communication status.
Variable "slave" ← Slave address ID
Variable "mode" ← TRC and STD bits
ID
Branch according to the selected slave address.
= 1
= 2
= 3
= 4
Others
D
E
F
G
mode
Branch according to the communication mode.
Processing for slave address ID=1 (A/D conversion result and LED lighting information)
Start transmission
Continue transmission
Start reception
Continue reception
A
B
C
Processing to start A/D conversion result transmission
Read A/D conversion result.
Variable datawork ← Variable g_tx_data[0] : Channel 0
Set upper data in transmission buffer.
Variable _g_IICA ← datawork >> 8
Start data transmission. _Tx_data_sub ()
Start transmitting data of variable _g_IICA.
Set lower byte in buffer.
Variable g_low_data_temp ← datawork & 0xFF
Set data pointer to 1.
Variable g_ptrx_data ← 0x01 : Process channel 1 next.
Set lower byte flag to 1.
Variable g_low_data_index ← TRUTH : Lower data exist.
return

Figure 5.23 I2C Communication End interrupt Transmission Process (1/4)

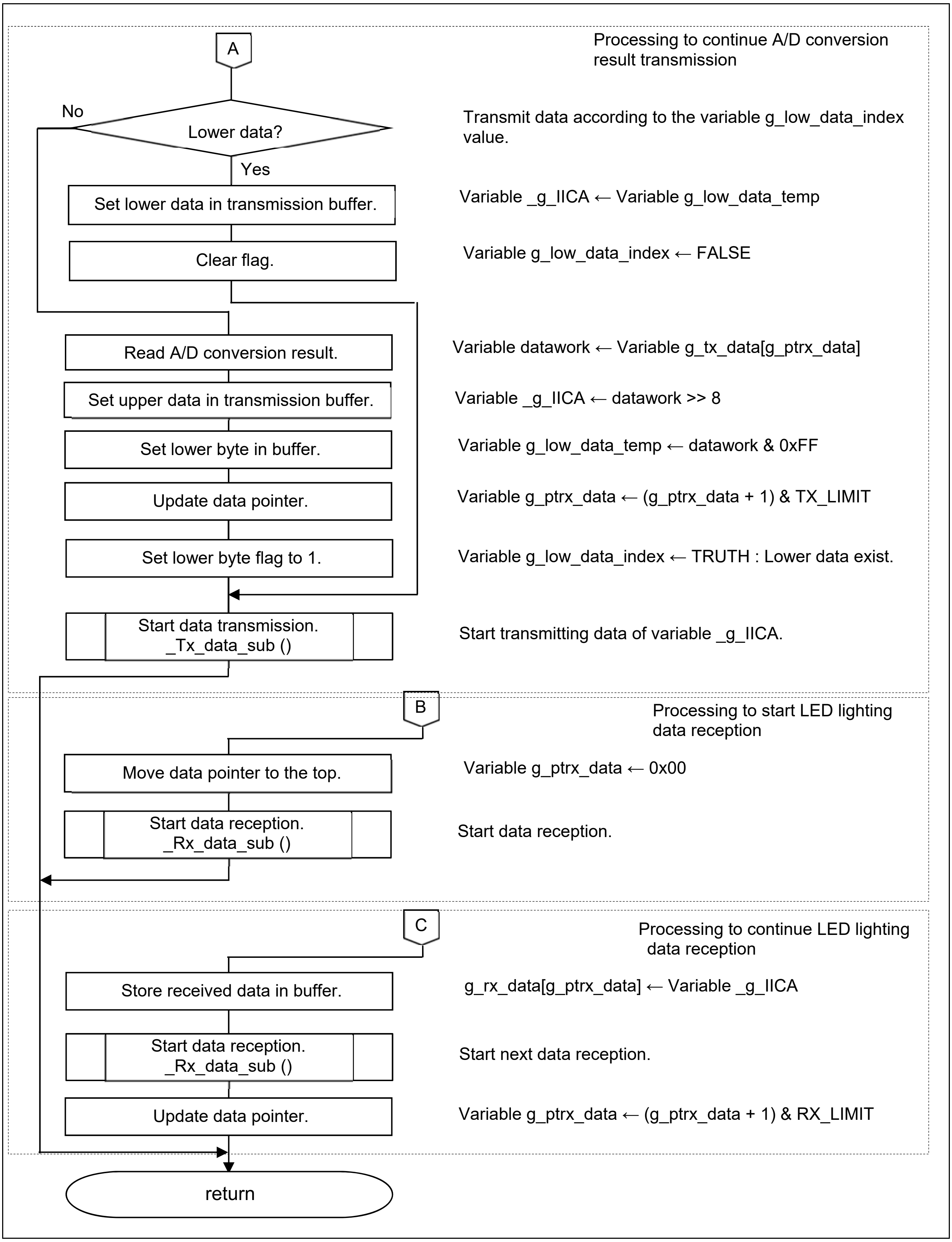

A
Processing to continue A/D conversion result transmission
Lower data?
No
Yes
Transmit data according to the variable g_low_data_index value.
Set lower data in transmission buffer.
Variable _g_IICA ← Variable g_low_data_temp
Clear flag.
Variable g_low_data_index ← FALSE
Read A/D conversion result.
Variable datawork ← Variable g_tx_data[g_ptrx_data]
Set upper data in transmission buffer.
Variable _g_IICA ← datawork >> 8
Set lower byte in buffer.
Variable g_low_data_temp ← datawork & 0xFF
Update data pointer.
Variable g_ptrx_data ← (g_ptrx_data + 1) & TX_LIMIT
Set lower byte flag to 1.
Variable g_low_data_index ← TRUTH : Lower data exist.
Start data transmission.
_Tx_data_sub ()
Start transmitting data of variable _g_IICA.
B
Processing to start LED lighting data reception
Move data pointer to the top.
Variable g_ptrx_data ← 0x00
Start data reception.
_Rx_data_sub ()
Start data reception.
C
Processing to continue LED lighting data reception
Store received data in buffer.
g_rx_data[g_ptrx_data] ← Variable _g_IICA
Start data reception.
_Rx_data_sub ()
Start next data reception.
Update data pointer.
Variable g_ptrx_data ← (g_ptrx_data + 1) & RX_LIMIT
return

Figure 5.24 I2C Communication End interrupt Process (2/4)

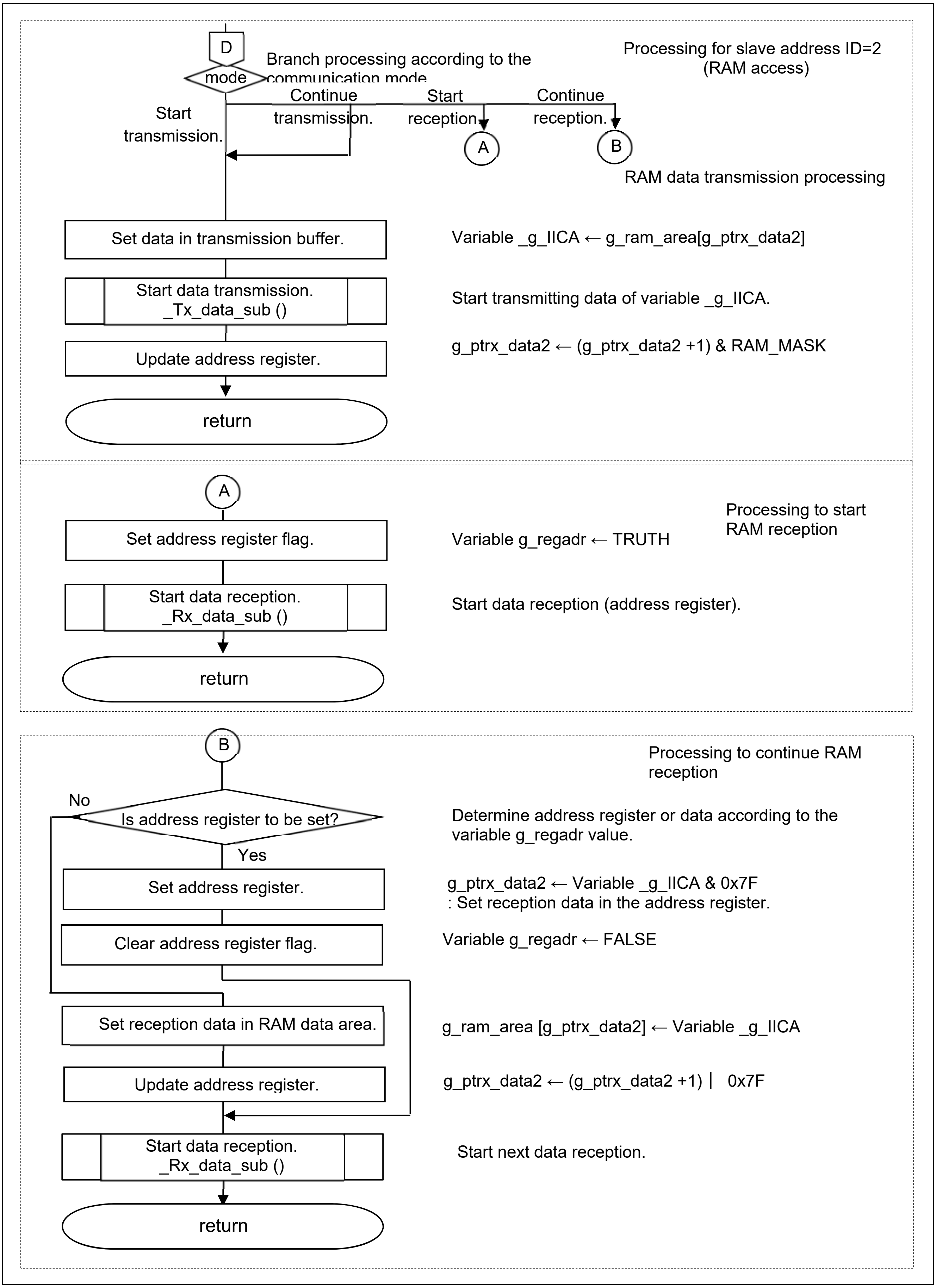
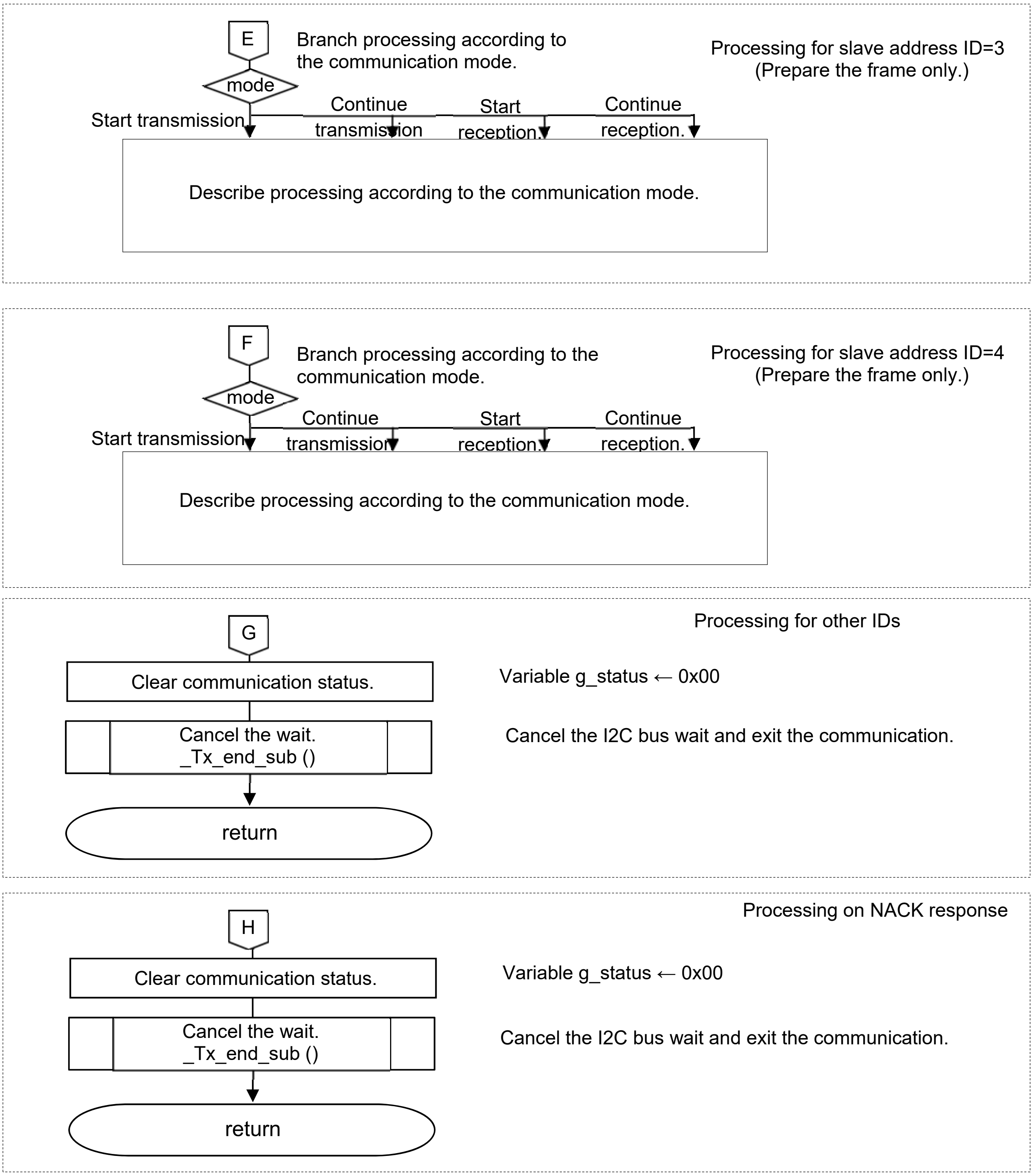

D
mode
Branch processing according to the communication mode
Processing for slave address ID=2 (RAM access)
Start transmission.
Continue transmission.
Start reception.
Continue reception.
A
B
RAM data transmission processing
Set data in transmission buffer.
Variable _g_IICA ← g_ram_area[g_ptrx_data2]
Start data transmission. _Tx_data_sub ()
Start transmitting data of variable _g_IICA.
Update address register.
g_ptrx_data2 ← (g_ptrx_data2 +1) & RAM_MASK
return
A
Processing to start RAM reception
Set address register flag.
Variable g_regadr ← TRUTH
Start data reception. _Rx_data_sub ()
Start data reception (address register).
return
B
Processing to continue RAM reception
No
Is address register to be set?
Determine address register or data according to the variable g_regadr value.
Yes
Set address register.
g_ptrx_data2 ← Variable _g_IICA & 0x7F
: Set reception data in the address register.
Clear address register flag.
Variable g_regadr ← FALSE
Set reception data in RAM data area.
g_ram_area [g_ptrx_data2] ← Variable _g_IICA
Update address register.
g_ptrx_data2 ← (g_ptrx_data2 +1) | 0x7F
Start data reception. _Rx_data_sub ()
Start next data reception.
return

Figure 5.25 I2C Communication End interrupt Process (3/4)


E
Branch processing according to the communication mode.
mode
Start transmission
Continue transmission
Start reception.
Continue reception.
Describe processing according to the communication mode.
Processing for slave address ID=3
(Prepare the frame only.)
F
Branch processing according to the communication mode.
mode
Start transmission
Continue transmission
Start reception.
Continue reception.
Describe processing according to the communication mode.
Processing for slave address ID=4
(Prepare the frame only.)
G
Processing for other IDs
Clear communication status.
Variable g_status ← 0x00
Cancel the wait.
_Tx_end_sub ()
Cancel the I2C bus wait and exit the communication.
return
H
Processing on NACK response
Clear communication status.
Variable g_status ← 0x00
Cancel the wait.
_Tx_end_sub ()
Cancel the I2C bus wait and exit the communication.
return

[Remarks] Only frames are provided for slave addresses ID=3 and ID=4 processes in this sample code. Whether _SADDR3_ and _SADDR4_ are previously defined or not determines whether to include the corresponding processes as targets of build between #ifdef and #endif. To use these parts, remove the comment-out for SADDR3 and SADDR4 definitions.

Figure 5.26 I2C Communication End interrupt Process (4/4)

5.7.23 Initializing I2C (Assembler Section)

Figure 5.27 shows the flowchart for initializing the I2C for the assembler section.

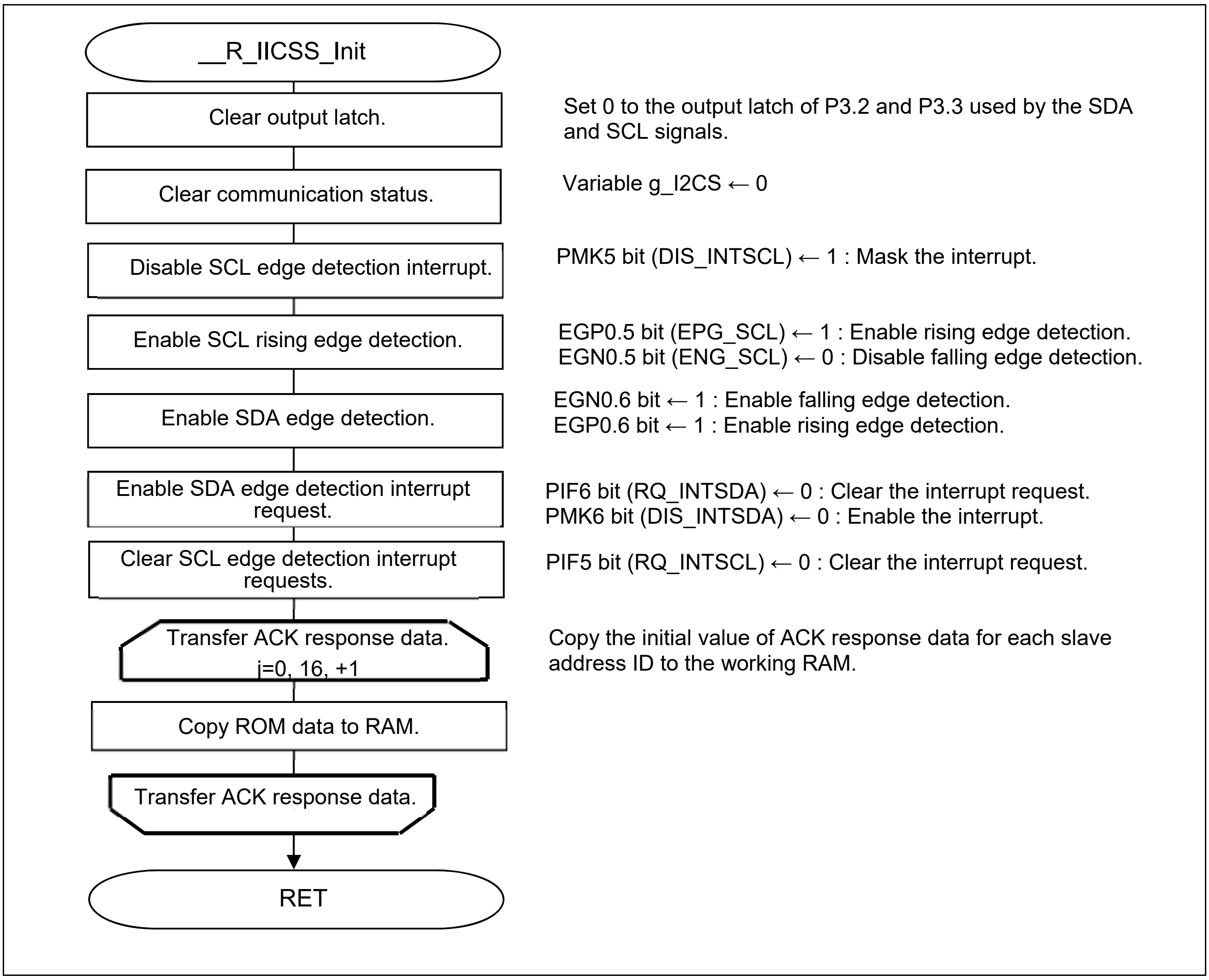

__R_IICSS_Init
Clear output latch.
Clear communication status.
Disable SCL edge detection interrupt.
Enable SCL rising edge detection.
Enable SDA edge detection.
Enable SDA edge detection interrupt request.
Clear SCL edge detection interrupt requests.
Transfer ACK response data.
i=0, 16, +1
Copy ROM data to RAM.
Transfer ACK response data.
RET
Set 0 to the output latch of P3.2 and P3.3 used by the SDA and SCL signals.
Variable g_I2CS ← 0
PMK5 bit (DIS_INTSCL) ← 1 : Mask the interrupt.
EGP0.5 bit (EPG_SCL) ← 1 : Enable rising edge detection.
EGN0.5 bit (ENG_SCL) ← 0 : Disable falling edge detection.
EGN0.6 bit ← 1 : Enable falling edge detection.
EGP0.6 bit ← 1 : Enable rising edge detection.
PIF6 bit (RQ_INTSDA) ← 0 : Clear the interrupt request.
PMK6 bit (DIS_INTSDA) ← 0 : Enable the interrupt.
PIF5 bit (RQ_INTSCL) ← 0 : Clear the interrupt request.
Copy the initial value of ACK response data for each slave address ID to the working RAM.

Figure 5.27 Initializing I2C (Assembler Section)

Clearing output latch

- Port register 3 (P3)
  Clear P3.3 and P3.2.

Symbol: P3

7	6	5	4	3	2	1	0
0	0	0	0	P33	P32	P31	P30
0	0	0	0	**0**	**0**	x	x

Bits 3 and 2

P3n	Data written to output latch
0	**Sets 0.**
1	1 を設定

For details of register settings, refer to the RL78/I1D User's Manual: Hardware.

Disabling SCL edge detection interrupt

- PMK5 bit in the interrupt mask flag register 0L (MK0L)
  Mask the INTP5 interrupt request.

Bit 7

PMK5	Control of interrupt processing
0	割り込み処理許可
1	**Disables interrupt processing.**

Setting SCL and SDA edge detection

- External interrupt rising edge enable register (EGP0)
  Set the EGP6 and EGP5 bits.
- External interrupt falling edge enable register (EGN0)
  Set the EGN6 bit and clear the EGN5 bit.

Symbol: EGP0

7	6	5	4	3	2	1	0
0	EGP6	EGP5	EGP4	EGP3	EGP2	EGP1	EGP0
0	**1**	**1**	x	x	x	x	x

Symbol: EGN0

7	6	5	4	3	2	1	0
0	EGN6	EGN5	EGN4	EGN3	EGN2	EGN1	EGN0
0	**1**	**0**	x	x	x	x	x

Bits 6 and 5

EGPn	EGNn	Selection of valid edge of INTPn pin
0	0	エッジ検出禁止
0	1	立ち下がりエッジ
1	0	**Rising edge (INTP5: SCL edge)**
1	**1**	**Rising and falling edges (INTP6: SDA edge)**

Clearing interrupt requests

- Interrupt request flag register (IF0L, IF0H)
  Clear the INTP5 and INTP6 interrupt requests.

Symbol: IF0L

7	6	5	4	3	2	1	0
PIF5	PIF4	PIF3	PIF2	PIF1	PIF0	LVIIF	WDTIIF
0	x	x	x	x	x	x	x

Symbol: IF0H

7	6	5	4	3	2	1	0
RTITIF	TMIF00	SREIF0	0	0	SRIF0	STIF0	PIF6
x	x	x	0	0	x	x	**0**

Bit n

PIFn	Interrupt request flag
0	**Interrupt request signal has not been generated.**
1	割り込み要求信号が発生し，割り込み要求状態

Enabling SDA edge detection interrupts

- PMK6 bit in the interrupt mask flag register 0H (MK0H)
  Cancel the INTP6 interrupt request mask.

Symbol: MK0H

7	6	5	4	3	2	1	0
RTITMK	TMMK00	SREMK0	0	0	SRMK0	STMK0	PMK6
x	x	x	0	0	x	x	**0**

Bit 0

PMK6	Control of interrupt processing
0	**Enables interrupt processing.**
1	割り込み処理禁止

5.7.24 Reading I2C Communication Status

Figure 5.28 shows the flowchart for reading the I2C communication status.

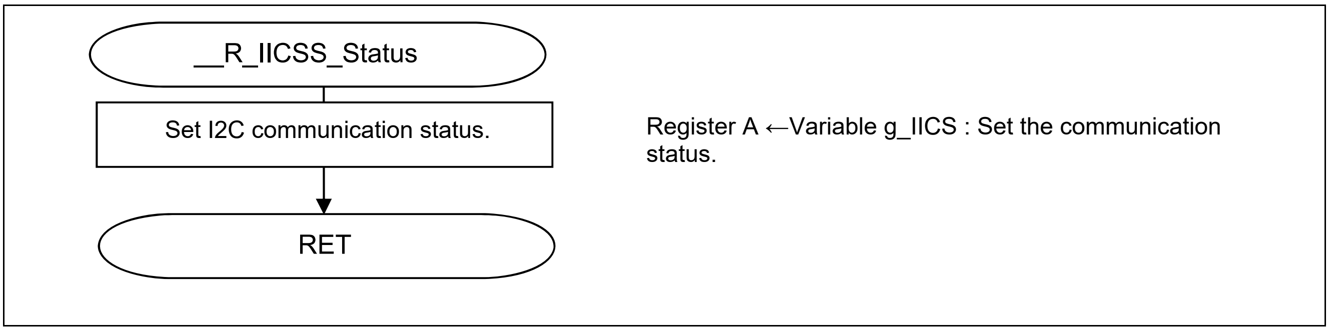

__R_IICSS_Status
Set I2C communication status.
Register A ←Variable g_IICS : Set the communication status.
RET

Figure 5.28 Reading I2C Communication Status

5.7.25 Setting ACK Response

Figure 5.29 shows the flowchart for setting ACK responses.

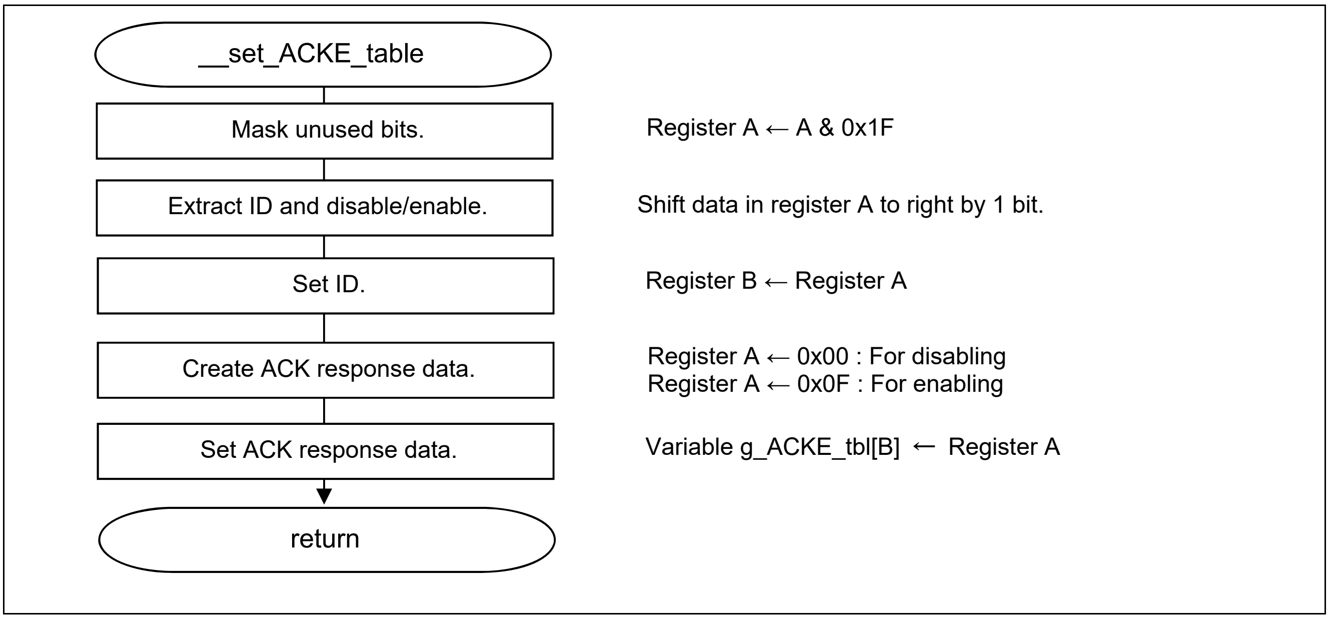

__set_ACKE_table
Mask unused bits.
Register A ← A & 0x1F
Extract ID and disable/enable.
Shift data in register A to right by 1 bit.
Set ID.
Register B ← Register A
Create ACK response data.
Register A ← 0x00 : For disabling
Register A ← 0x0F : For enabling
Set ACK response data.
Variable g_ACKE_tbl[B] ← Register A
return

Figure 5.29 Setting ACK Response

5.7.26 Reading ACK Responses

Figure 5.30 shows the flowchart for reading the ACK responses.

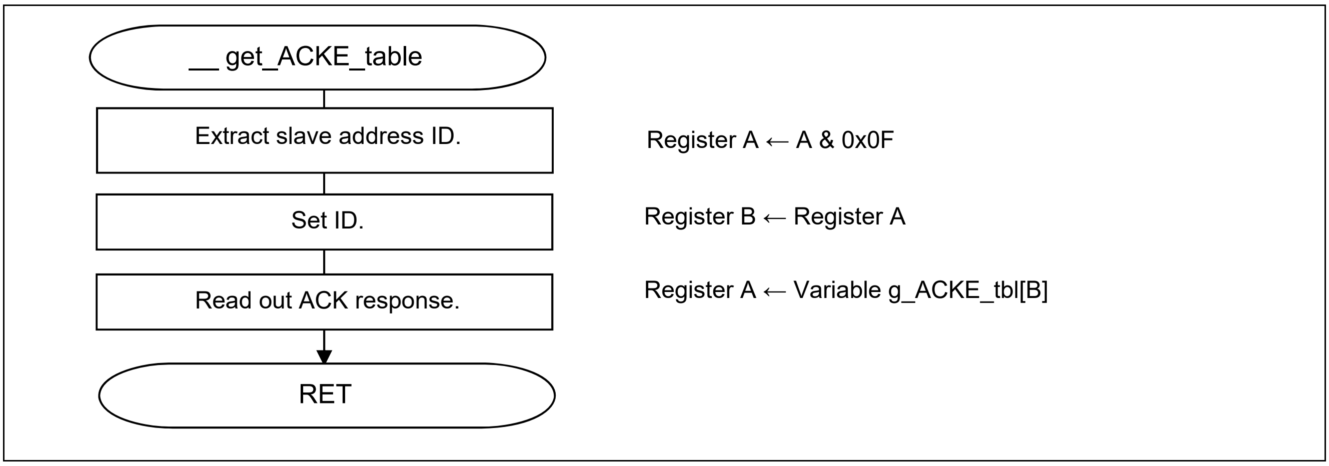


Figure 5.30 Reading ACK Responses

5.7.27 SCL Edge Detection Interrupt Entry Process

Figure 5.31 shows the flowchart of the SCL edge detection interrupt entry process.

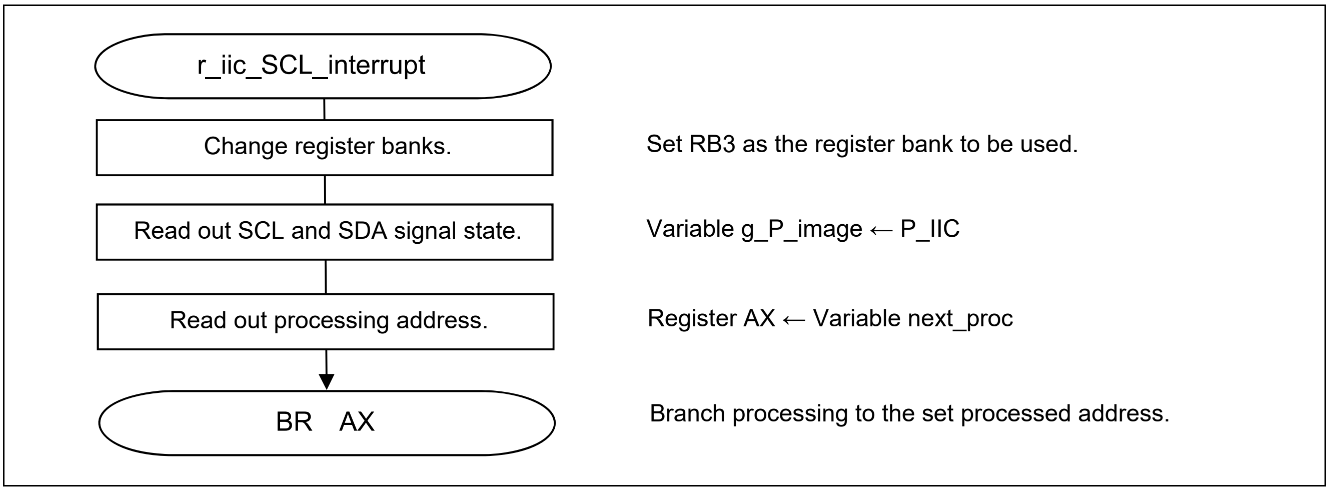

r_iic_SCL_interrupt
Change register banks.
Set RB3 as the register bank to be used.
Read out SCL and SDA signal state.
Variable g_P_image ← P_IIC
Read out processing address.
Register AX ← Variable next_proc
BR AX
Branch processing to the set processed address.

Figure 5.31 SCL Edge Detection Interrupt Entry Process

This section only shows the entry processing of the SCL edge detection interrupt. For the description of actual processing of the interrupt, see 5.8.29, SCL Edge Detection Interrupt Processing.

Reading Out SCL and SDA Signals

- Port register 3 (P3)

Read out SCL and SDA signals.

Symbol: P3

7	6	5	4	3	2	1	0
0	0	0	0	SCL	SDA	P31	P30

5.7.28 SDA Edge Detection Interrupt Process

Figures 5.32 to 5.33 show the flowcharts of the SDA edge detection interrupt process.

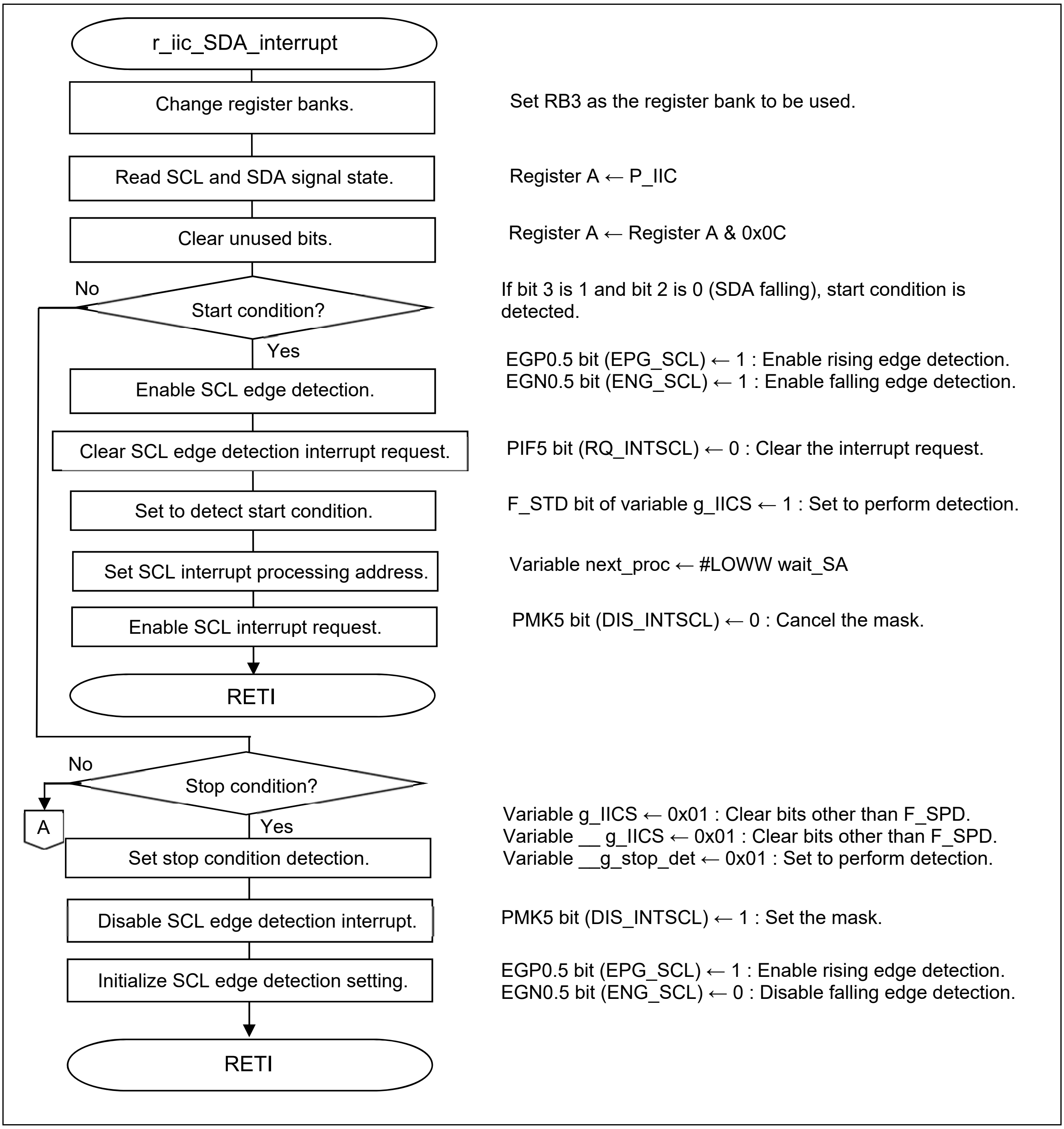

r_iic_SDA_interrupt
Change register banks.
Set RB3 as the register bank to be used.
Read SCL and SDA signal state.
Register A ← P_IIC
Clear unused bits.
Register A ← Register A & 0x0C
No
Start condition?
If bit 3 is 1 and bit 2 is 0 (SDA falling), start condition is detected.
Yes
Enable SCL edge detection.
EGP0.5 bit (EPG_SCL) ← 1 : Enable rising edge detection.
EGN0.5 bit (ENG_SCL) ← 1 : Enable falling edge detection.
Clear SCL edge detection interrupt request.
PIF5 bit (RQ_INTSCL) ← 0 : Clear the interrupt request.
Set to detect start condition.
F_STD bit of variable g_IICS ← 1 : Set to perform detection.
Set SCL interrupt processing address.
Variable next_proc ← #LOWW wait_SA
Enable SCL interrupt request.
PMK5 bit (DIS_INTSCL) ← 0 : Cancel the mask.
RETI
No
Stop condition?
A
Yes
Set stop condition detection.
Variable g_IICS ← 0x01 : Clear bits other than F_SPD.
Variable __g_IICS ← 0x01 : Clear bits other than F_SPD.
Variable __g_stop_det ← 0x01 : Set to perform detection.
Disable SCL edge detection interrupt.
PMK5 bit (DIS_INTSCL) ← 1 : Set the mask.
Initialize SCL edge detection setting.
EGP0.5 bit (EPG_SCL) ← 1 : Enable rising edge detection.
EGN0.5 bit (ENG_SCL) ← 0 : Disable falling edge detection.
RETI

Figure 5.32 SDA Edge Detection Interrupt Process (1/2)

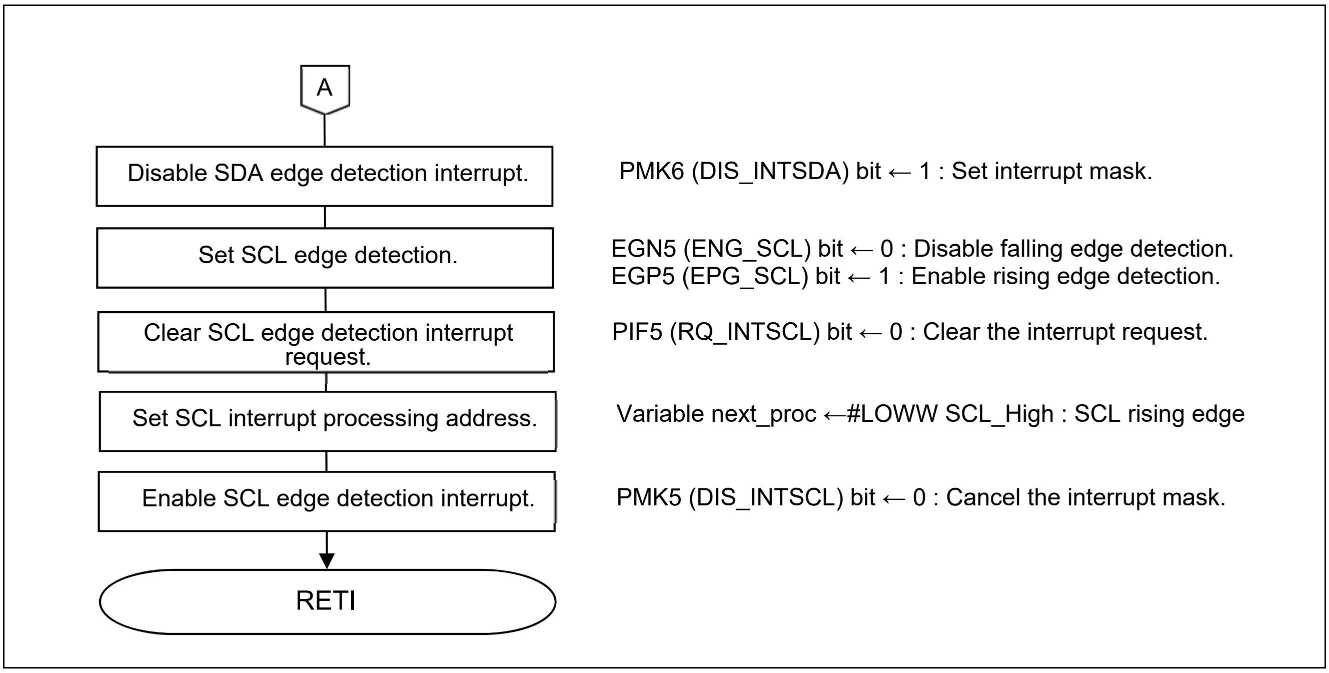

A
Disable SDA edge detection interrupt.
PMK6 (DIS_INTSDA) bit ← 1 : Set interrupt mask.
Set SCL edge detection.
EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.
EGP5 (EPG_SCL) bit ← 1 : Enable rising edge detection.
Clear SCL edge detection interrupt request.
PIF5 (RQ_INTSCL) bit ← 0 : Clear the interrupt request.
Set SCL interrupt processing address.
Variable next_proc ←#LOWW SCL_High : SCL rising edge
Enable SCL edge detection interrupt.
PMK5 (DIS_INTSCL) bit ← 0 : Cancel the interrupt mask.
RETI

Figure 5.33 SDA Edge Detection Interrupt Process (2/2)

The following control registers are used for both the SCL edge detection interrupt processing and SDA edge detection interrupt processing. This section describes these control registers collectively.

Reading SCL and SDA signals

- Port register 3 (P3)

Read SCL and SDA signals.

Symbol: P3

7	6	5	4	3	2	1	0
0	0	0	0	SCL	SDA	P31	P30

Setting edge detection

- External interrupt rising edge enable register (EGP0)

Enable/disable INTP5 edge detection.

- External interrupt falling edge enable register (EGN0)

Enable/disable INTP5 edge detection.

Symbol: EGP0

7	6	5	4	3	2	1	0
0	EGP6	EGP5	EGP4	EGP3	EGP2	EGP1	EGP0
0x	1	**1**	x	x	x	x	x

Symbol: EGN0

7	6	5	4	3	2	1	0
0	EGN6	EGN5	EGN4	EGN3	EGN2	EGN1	EGN0
0	1	**1/0**	x	x	x	x	x

Bit 5

EGP5	EGN5	Selection of valid edge of INTP5 pin
0	0	エッジ検出禁止
0	1	立ち下がりエッジ
1	**0**	**Rising edge**
1	**1**	**Both rising and falling edges**

Controlling SCL and SDA edge detection interrupts

- Interrupt mask flag register (MK0L, MK0H)
  Control interrupt requests.
- Interrupt request flag register (IF0L, IF0H)
  Control interrupt requests.

Symbol: MK0L

7	6	5	4	3	2	1	0
PMK5	PMK4	PMK3	PMK2	PMK1	PMK0	LVIMK	WDTIMK
0/1	x	x	x	x	x	x	x

Symbol: MK0H

7	6	5	4	3	2	1	0
RTITMK	TMMK00	SREMK0	1	1	SRMK0	STMK0	PMK6
x	x	x	1	1	x	x	**0/1**

Bit n

PMKn	Control of interrupt processing
0	**Enables interrupt processing.**
1	**Disables interrupt processing.**

Symbol: IF0L

7	6	5	4	3	2	1	0
PIF5	PIF4	PIF3	PIF2	PIF1	PIF0	LVIIF	WDTIIF
0	**0**	**0**	**0**	**0**	**0**	x	x

Symbol: IF0H

7	6	5	4	3	2	1	0
RTITIF	TMIF00	SREIF0	0	0	SRIF0	STIF0	PIF6
x	x	x	0	0	x	x	**0**

Bit n

PIFn	Interrupt request flag
0	**Interrupt request signal has not been generated.**
1	割り込み要求信号が発生し，割り込み要求状態

5.7.29 SCL Edge Detection Interrupt Process

Figures 5.34 to 5.52 show the flowcharts of SCL edge detection interrupt process.

(0) Waiting for stop/start condition

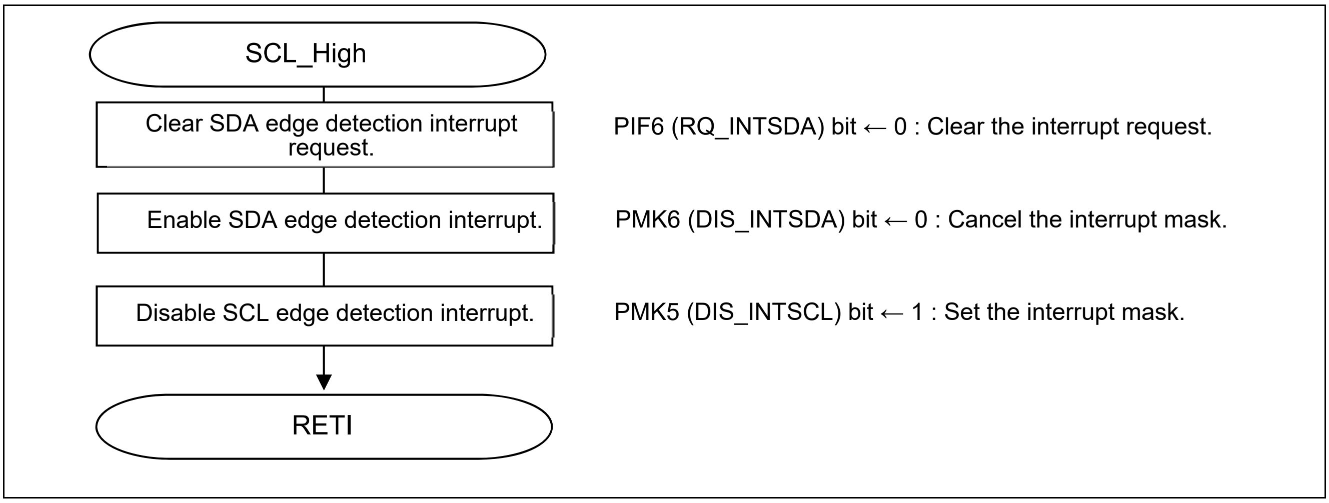

SCL_High
Clear SDA edge detection interrupt request.
Enable SDA edge detection interrupt.
Disable SCL edge detection interrupt.
RETI
PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.
PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.
PMK5 (DIS_INTSCL) bit ← 1 : Set the interrupt mask.

Figure 5.34 Waiting for Stop/Start Condition

(1) Waiting for slave address reception start (sequence (4))

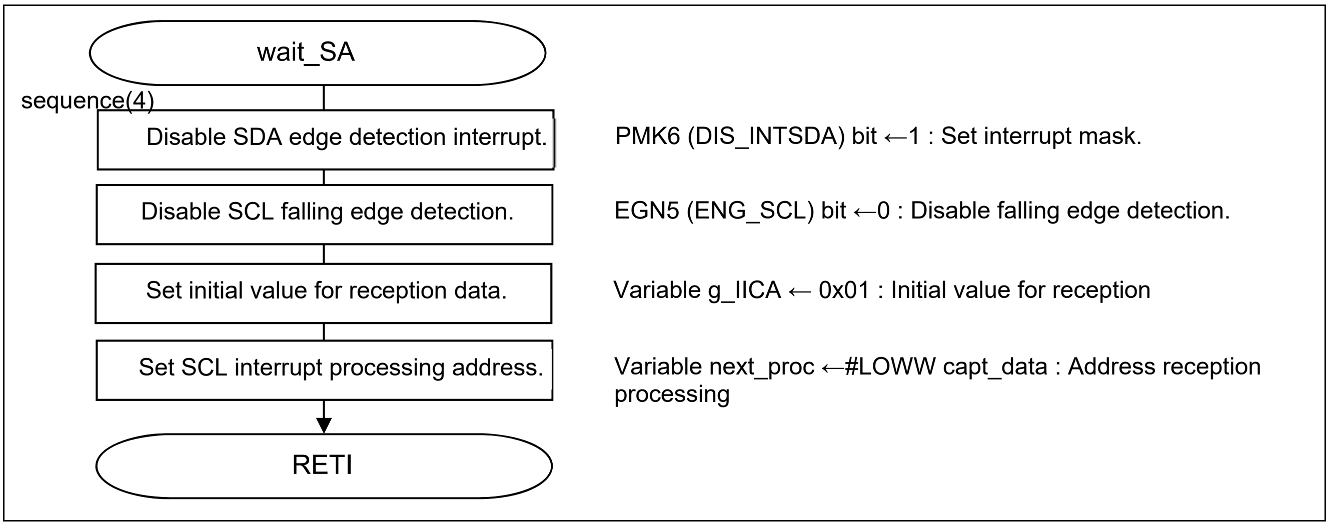
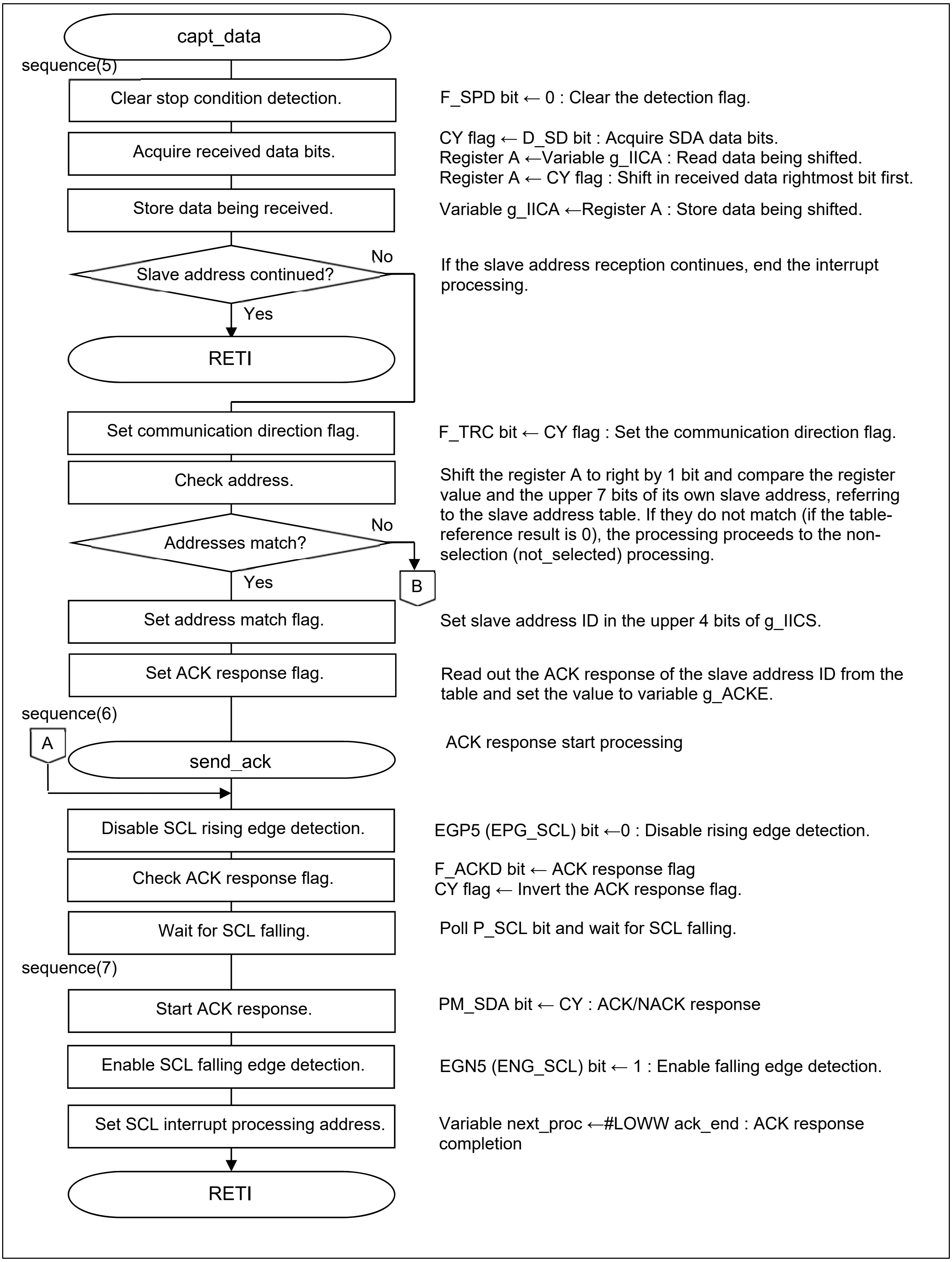

wait_SA
sequence(4)
Disable SDA edge detection interrupt.
Disable SCL falling edge detection.
Set initial value for reception data.
Set SCL interrupt processing address.
RETI
PMK6 (DIS_INTSDA) bit ←1 : Set interrupt mask.
EGN5 (ENG_SCL) bit ←0 : Disable falling edge detection.
Variable g_IICA ← 0x01 : Initial value for reception
Variable next_proc ←#LOWW capt_data : Address reception processing

Figure 5.35 Waiting for Slave Address Reception Start

(2) Receiving slave address (sequence (5))



F_SPD bit ← 0 : Clear the detection flag.

CY flag ← D_SD bit : Acquire SDA data bits.
Register A ←Variable g_IICA : Read data being shifted.
Register A ← CY flag : Shift in received data rightmost bit first.

Variable g_IICA ←Register A : Store data being shifted.

If the slave address reception continues, end the interrupt processing.

F_TRC bit ← CY flag : Set the communication direction flag.

Shift the register A to right by 1 bit and compare the register value and the upper 7 bits of its own slave address, referring to the slave address table. If they do not match (if the table-reference result is 0), the processing proceeds to the non-selection (not_selected) processing.

Set slave address ID in the upper 4 bits of g_IICS.

Read out the ACK response of the slave address ID from the table and set the value to variable g_ACKE.

ACK response start processing

EGP5 (EPG_SCL) bit ←0 : Disable rising edge detection.

F_ACKD bit ← ACK response flag
CY flag ← Invert the ACK response flag.

Poll P_SCL bit and wait for SCL falling.

PM_SDA bit ← CY : ACK/NACK response

EGN5 (ENG_SCL) bit ← 1 : Enable falling edge detection.

Variable next_proc ←#LOWW ack_end : ACK response completion

Figure 5.36 Receiving Slave Address

(3) Non-Selection Processing (sequence(6)’)

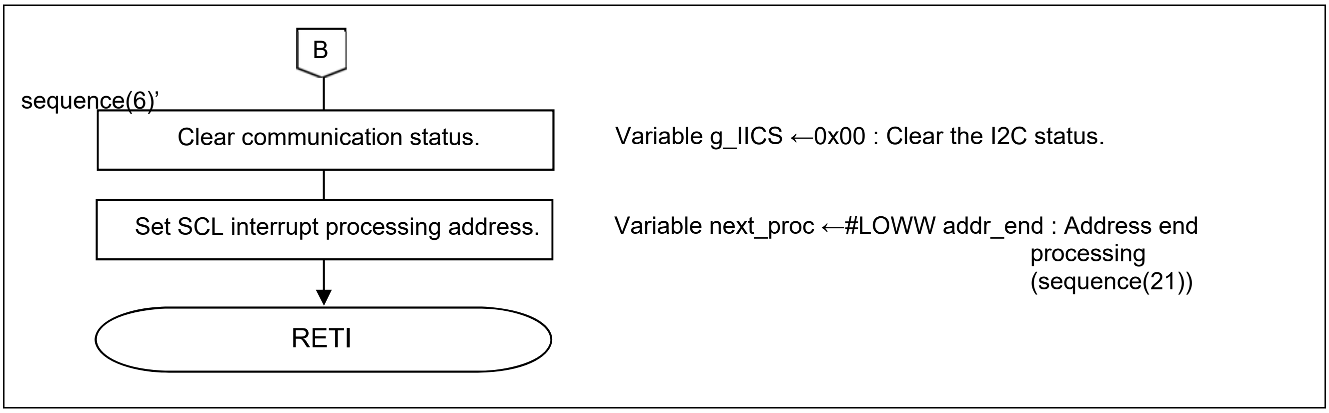

B
sequence(6)’
Clear communication status.
Set SCL interrupt processing address.
RETI
Variable g_IICS ←0x00 : Clear the I2C status.
Variable next_proc ←#LOWW addr_end : Address end processing (sequence(21))

Figure 5.37 Non-Selection Processing

(4) ACK Response End Processing (sequence(8))

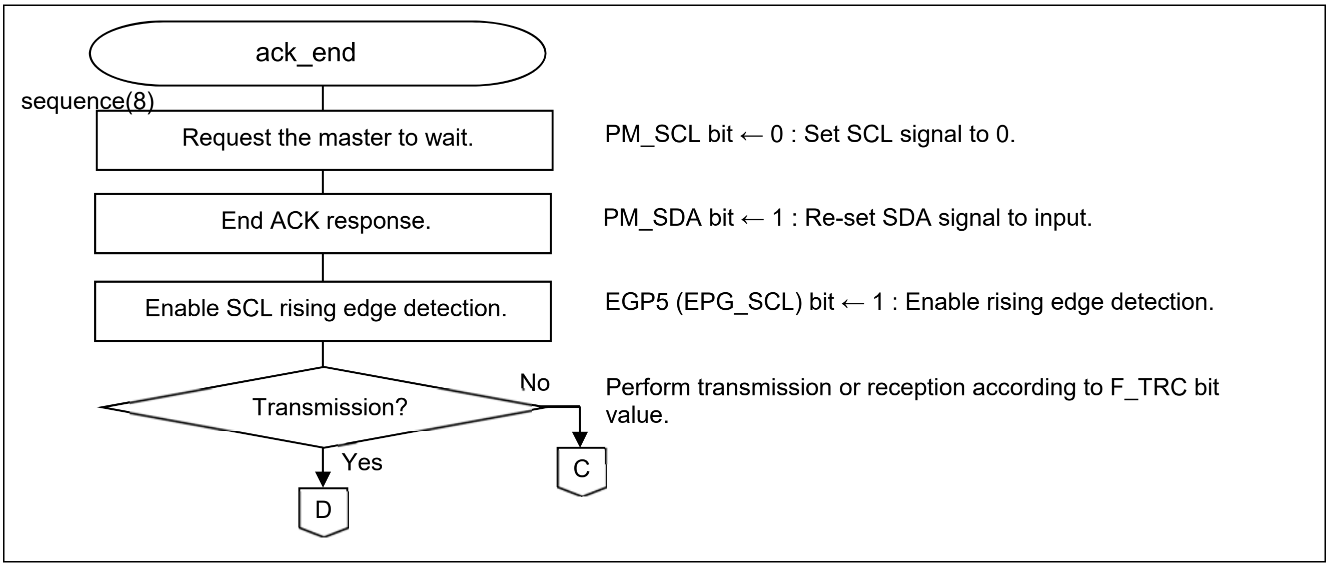
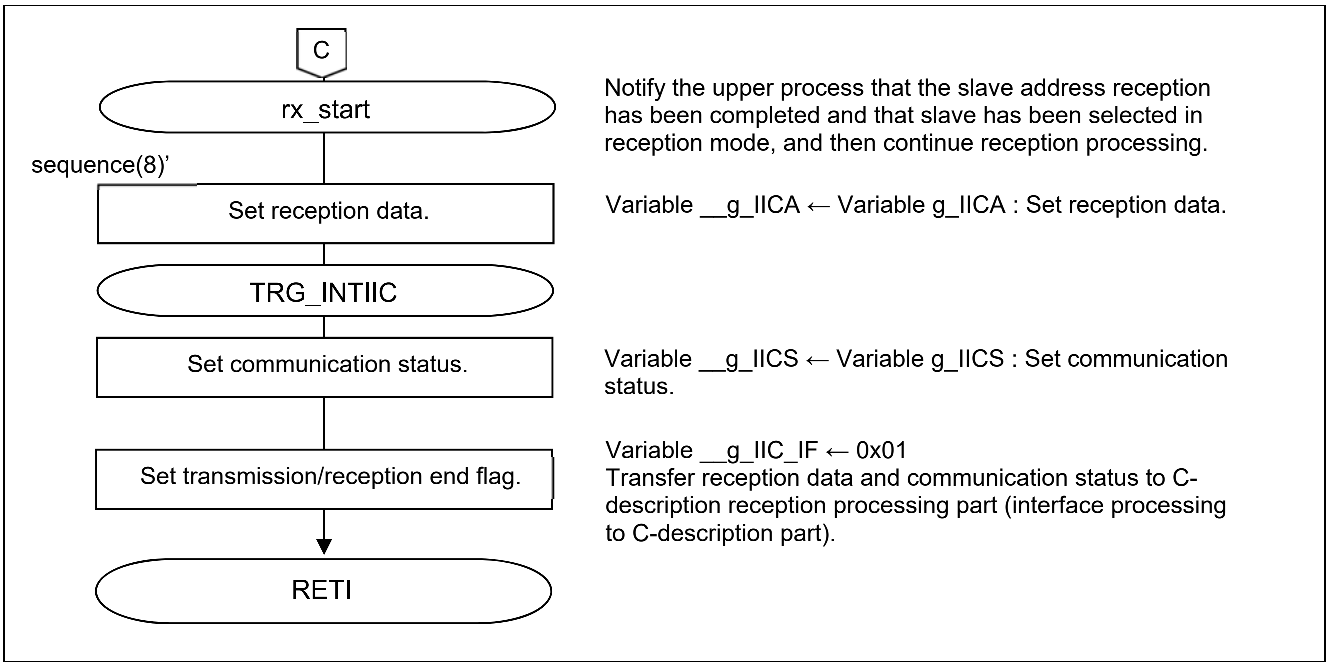

ack_end
sequence(8)
Request the master to wait.
End ACK response.
Enable SCL rising edge detection.
Transmission?
No
Yes
C
D
PM_SCL bit ← 0 : Set SCL signal to 0.
PM_SDA bit ← 1 : Re-set SDA signal to input.
EGP5 (EPG_SCL) bit ← 1 : Enable rising edge detection.
Perform transmission or reception according to F_TRC bit value.

Figure 5.38 ACK Response End Processing

(5) Reception Completion (Operation Start Request) Processing (sequence(8)')

After completion of one-byte data reception, set the communication end flag to the upper software and end processing.


C
rx_start
sequence(8)'
Set reception data.
TRG_INTIIC
Set communication status.
Set transmission/reception end flag.
RETI

Notify the upper process that the slave address reception has been completed and that slave has been selected in reception mode, and then continue reception processing.

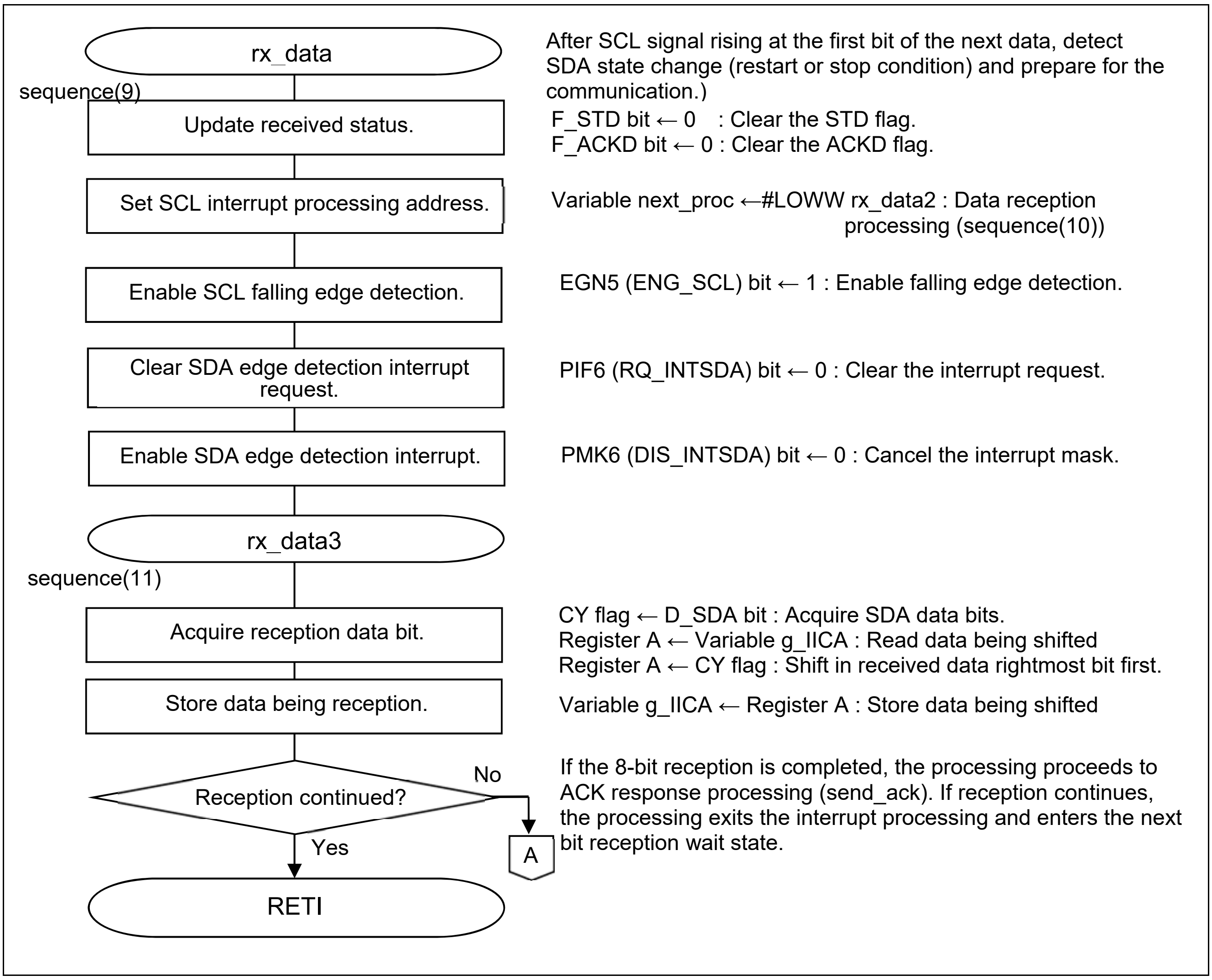
Variable __g_IICA ← Variable g_IICA : Set reception data.

Variable __g_IICS ← Variable g_IICS : Set communication status.

Variable __g_IIC_IF ← 0x01
Transfer reception data and communication status to C-description reception processing part (interface processing to C-description part).

Figure 5.39 Reception Completion Processing

(6) Receiving Data (sequence(9))


rx_data
sequence(9)
Update received status.
Set SCL interrupt processing address.
Enable SCL falling edge detection.
Clear SDA edge detection interrupt request.
Enable SDA edge detection interrupt.
rx_data3
sequence(11)
Acquire reception data bit.
Store data being reception.
Reception continued?
No
Yes
A
RETI

After SCL signal rising at the first bit of the next data, detect SDA state change (restart or stop condition) and prepare for the communication.)

F_STD bit ← 0 : Clear the STD flag.
F_ACKD bit ← 0 : Clear the ACKD flag.

Variable next_proc ←#LOWW rx_data2 : Data reception processing (sequence(10))

EGN5 (ENG_SCL) bit ← 1 : Enable falling edge detection.

PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.

PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.

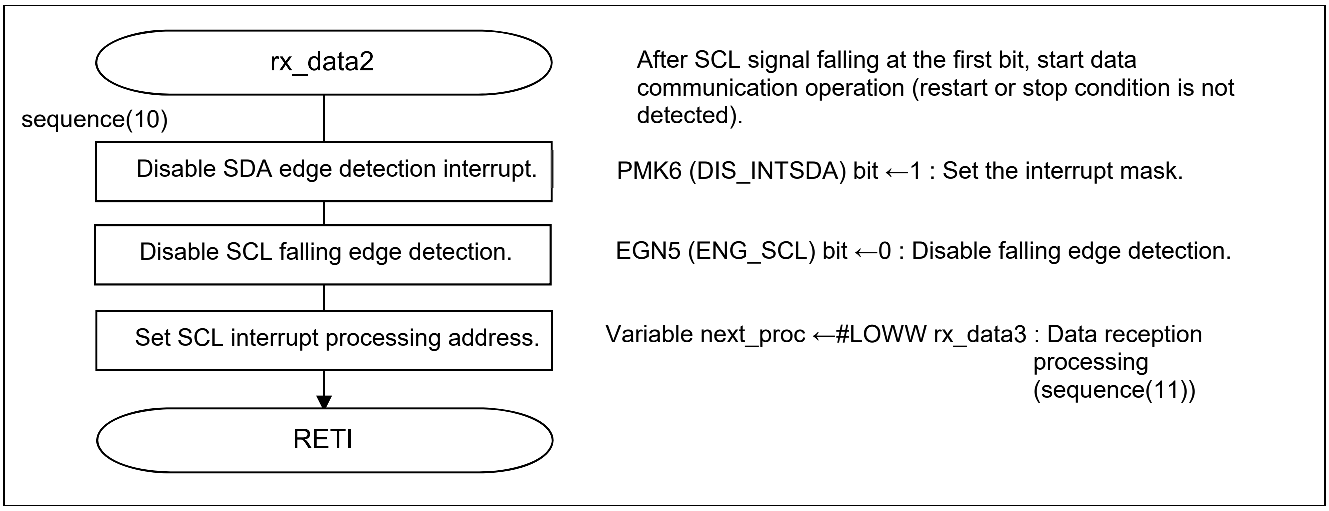
CY flag ← D_SDA bit : Acquire SDA data bits.
Register A ← Variable g_IICA : Read data being shifted
Register A ← CY flag : Shift in received data rightmost bit first.

Variable g_IICA ← Register A : Store data being shifted

If the 8-bit reception is completed, the processing proceeds to ACK response processing (send_ack). If reception continues, the processing exits the interrupt processing and enters the next bit reception wait state.

Figure 5.40 Receiving Data

(7) Starting Reception Operation (sequence(10))


rx_data2
sequence(10)
Disable SDA edge detection interrupt.
Disable SCL falling edge detection.
Set SCL interrupt processing address.
RETI

After SCL signal falling at the first bit, start data communication operation (restart or stop condition is not detected).

PMK6 (DIS_INTSDA) bit ←1 : Set the interrupt mask.

EGN5 (ENG_SCL) bit ←0 : Disable falling edge detection.

Variable next_proc ←#LOWW rx_data3 : Data reception processing (sequence(11))

Figure 5.41 Starting Reception Operation

(8) Requesting Transmission Operation Start (sequence(14))

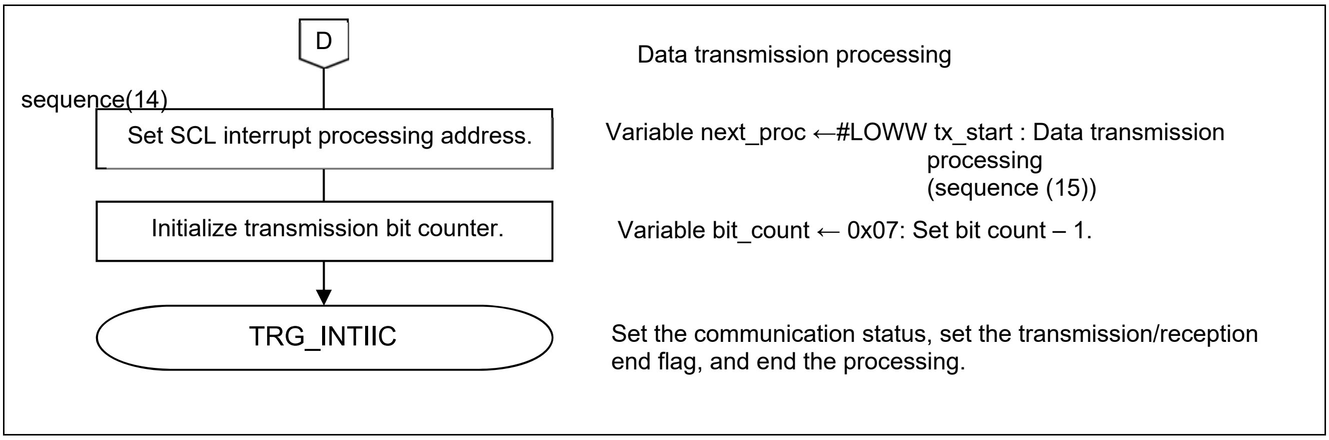

D
Data transmission processing
sequence(14)
Set SCL interrupt processing address.
Variable next_proc ←#LOWW tx_start : Data transmission processing (sequence (15))
Initialize transmission bit counter.
Variable bit_count ← 0x07: Set bit count – 1.
TRG_INTIIC
Set the communication status, set the transmission/reception end flag, and end the processing.

Figure 5.42 Requesting Transmission Operation Start

(9) Starting Transmission Operation (sequence(15))

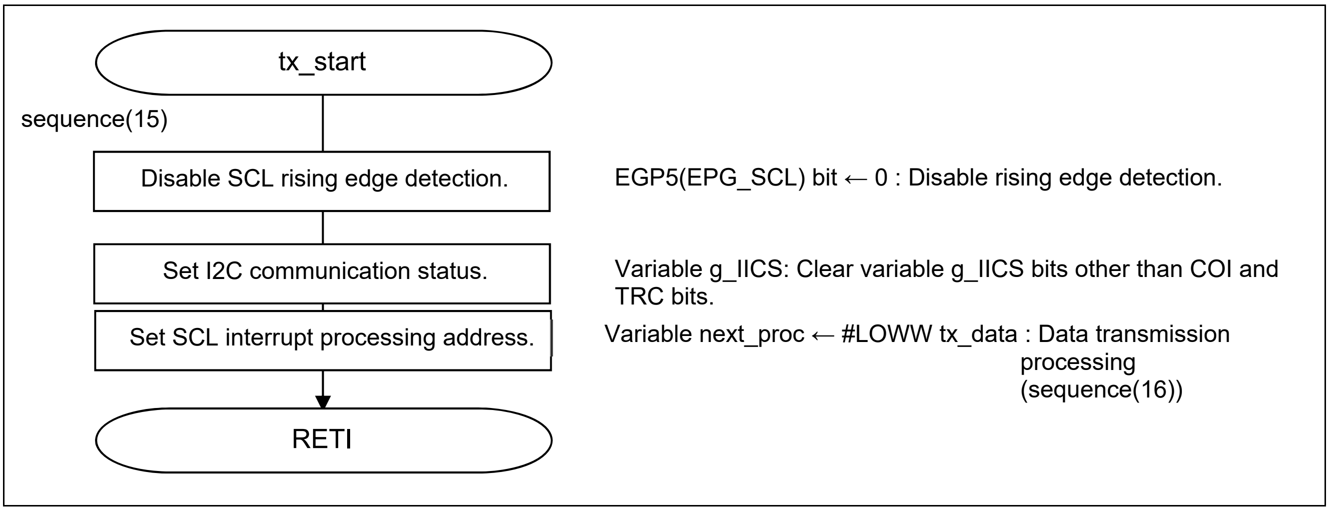

tx_start
sequence(15)
Disable SCL rising edge detection.
EGP5(EPG_SCL) bit ← 0 : Disable rising edge detection.
Set I2C communication status.
Variable g_IICS: Clear variable g_IICS bits other than COI and TRC bits.
Set SCL interrupt processing address.
Variable next_proc ← #LOWW tx_data : Data transmission processing (sequence(16))
RETI

Figure 5.43 Starting Transmission Operation

(10) Transmission Processing (sequence(16))

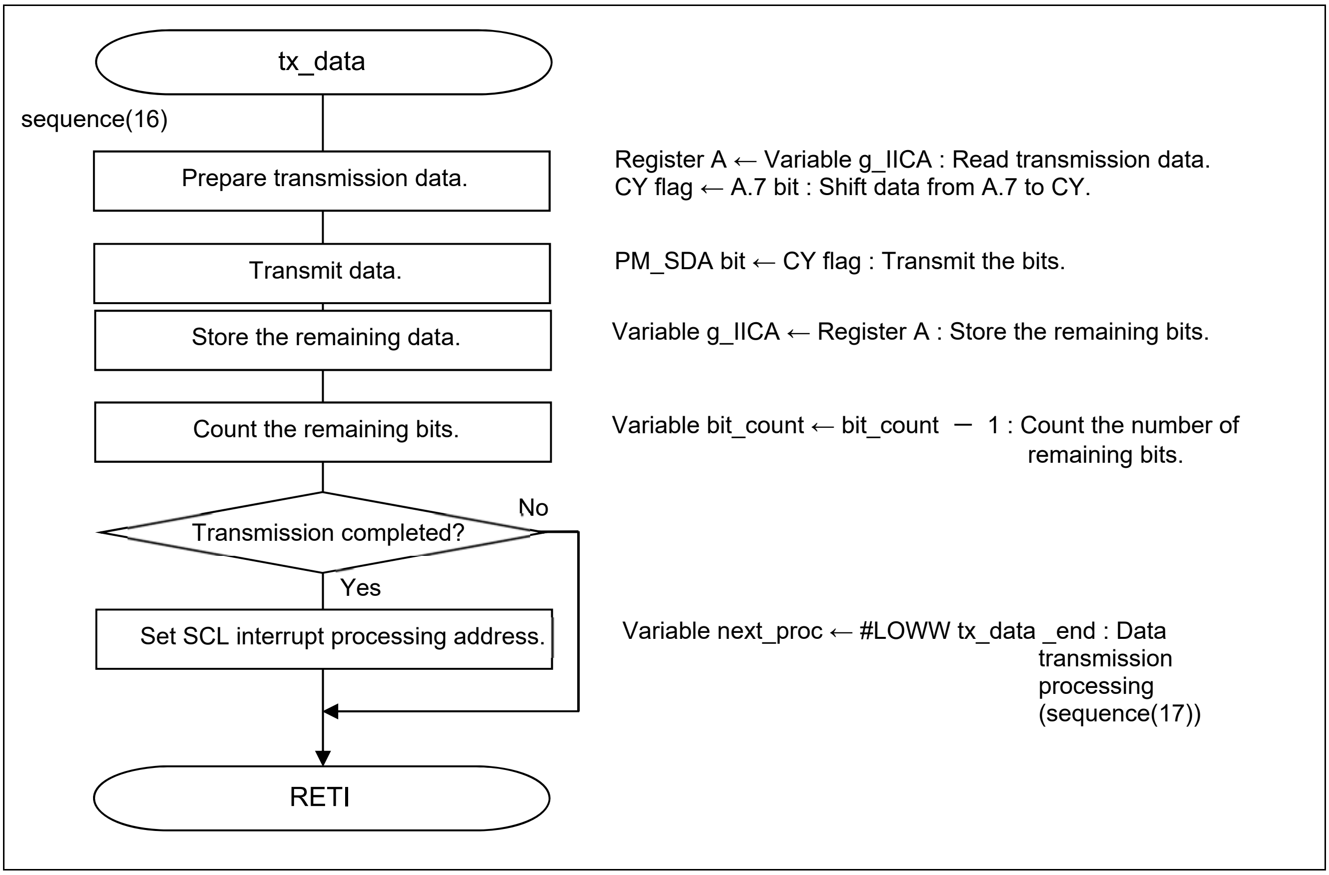

tx_data
sequence(16)
Prepare transmission data.
Register A ← Variable g_IICA : Read transmission data.
CY flag ← A.7 bit : Shift data from A.7 to CY.
Transmit data.
PM_SDA bit ← CY flag : Transmit the bits.
Store the remaining data.
Variable g_IICA ← Register A : Store the remaining bits.
Count the remaining bits.
Variable bit_count ← bit_count − 1 : Count the number of remaining bits.
Transmission completed?
No
Yes
Set SCL interrupt processing address.
Variable next_proc ← #LOWW tx_data _end : Data transmission processing (sequence(17))
RETI

Figure 5.44 Transmission Processing

(11) Transmission End Processing (sequence(17))

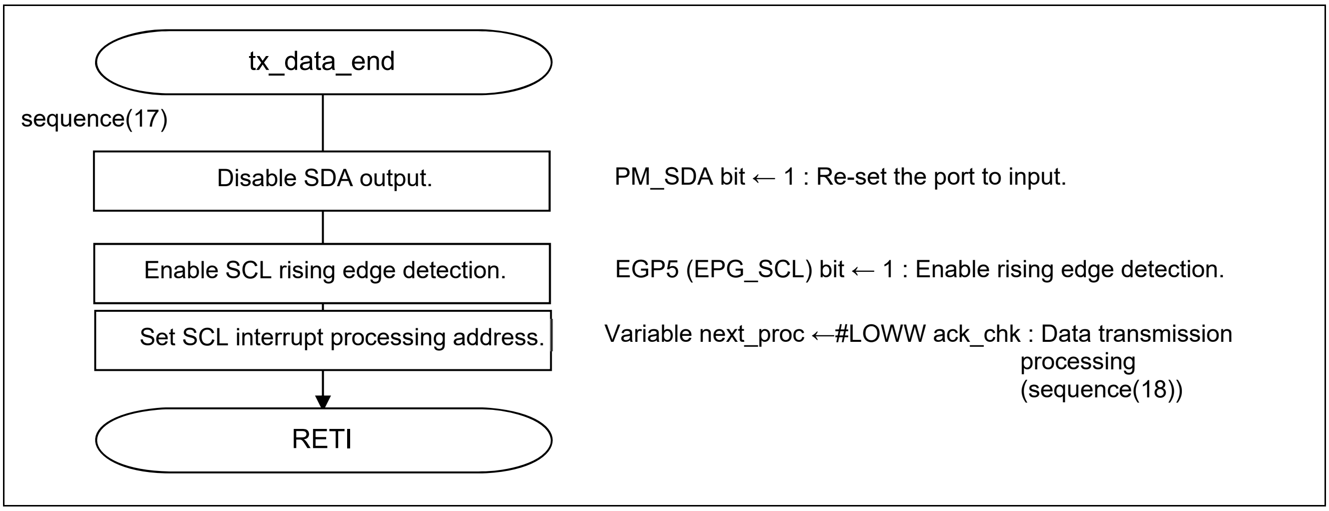

tx_data_end
sequence(17)
Disable SDA output.
PM_SDA bit ← 1 : Re-set the port to input.
Enable SCL rising edge detection.
EGP5 (EPG_SCL) bit ← 1 : Enable rising edge detection.
Set SCL interrupt processing address.
Variable next_proc ←#LOWW ack_chk : Data transmission processing (sequence(18))
RETI

Figure 5.45 Transmission End Processing

(12) Checking ACK (sequence(18))

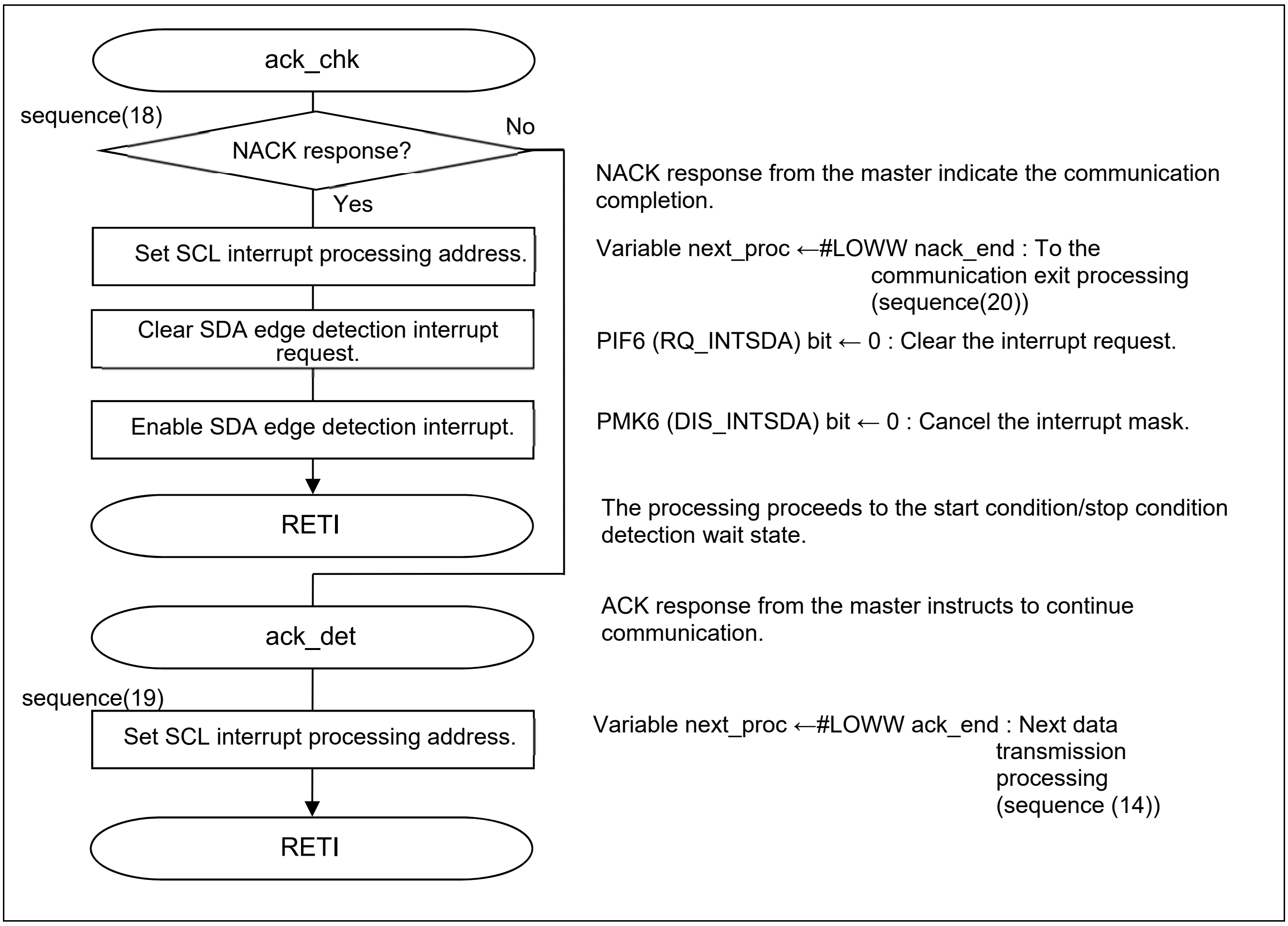

ack_chk
sequence(18)
NACK response?
No
Yes
Set SCL interrupt processing address.
Clear SDA edge detection interrupt request.
Enable SDA edge detection interrupt.
RETI
ack_det
sequence(19)
Set SCL interrupt processing address.
RETI
NACK response from the master indicate the communication completion.
Variable next_proc ←#LOWW nack_end : To the communication exit processing (sequence(20))
PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.
PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.
The processing proceeds to the start condition/stop condition detection wait state.
ACK response from the master instructs to continue communication.
Variable next_proc ←#LOWW ack_end : Next data transmission processing (sequence (14))

Figure 5.46 Checking ACK

(13) NACK Completion Processing (sequence(20))

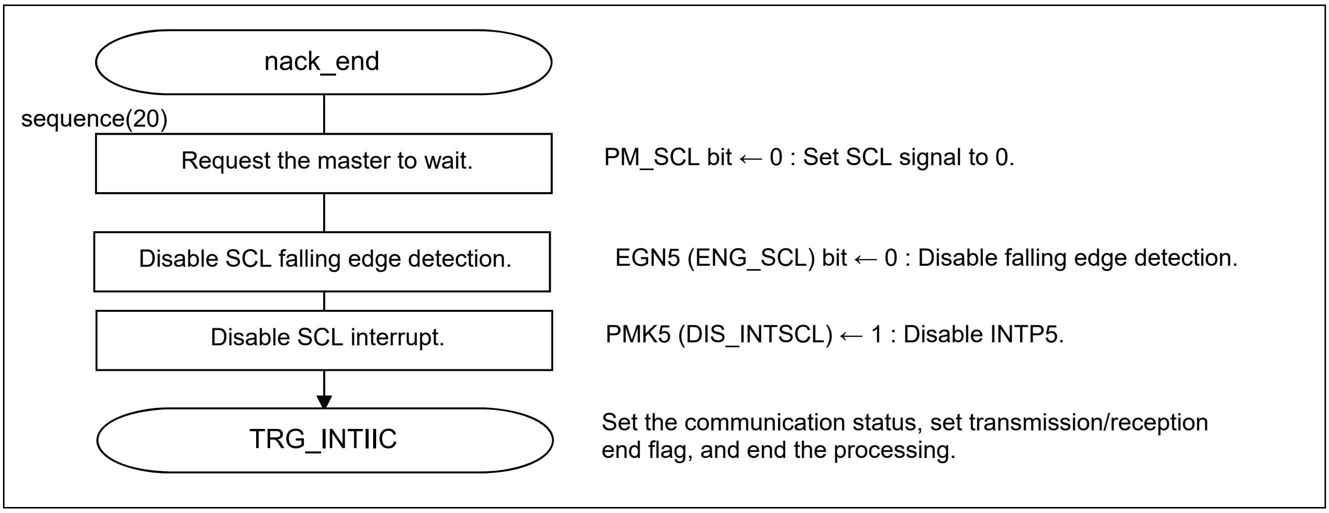

nack_end
sequence(20)
Request the master to wait.
Disable SCL falling edge detection.
Disable SCL interrupt.
TRG_INTIIC
PM_SCL bit ← 0 : Set SCL signal to 0.
EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.
PMK5 (DIS_INTSCL) ← 1 : Disable INTP5.
Set the communication status, set transmission/reception end flag, and end the processing.

Figure 5.47 NACK Completion Processing

(14) Non-Selection Processing (1/2) (sequence(21))

Even when the slave itself is not selected and it does not perform data processing, the slave needs to monitor the communication state by counting the number of SCLs.

In the non-selection processing below, only the SCL counting is performed to reduce the CPU processing during the not-selected state (this operation is referred to as skip-read).

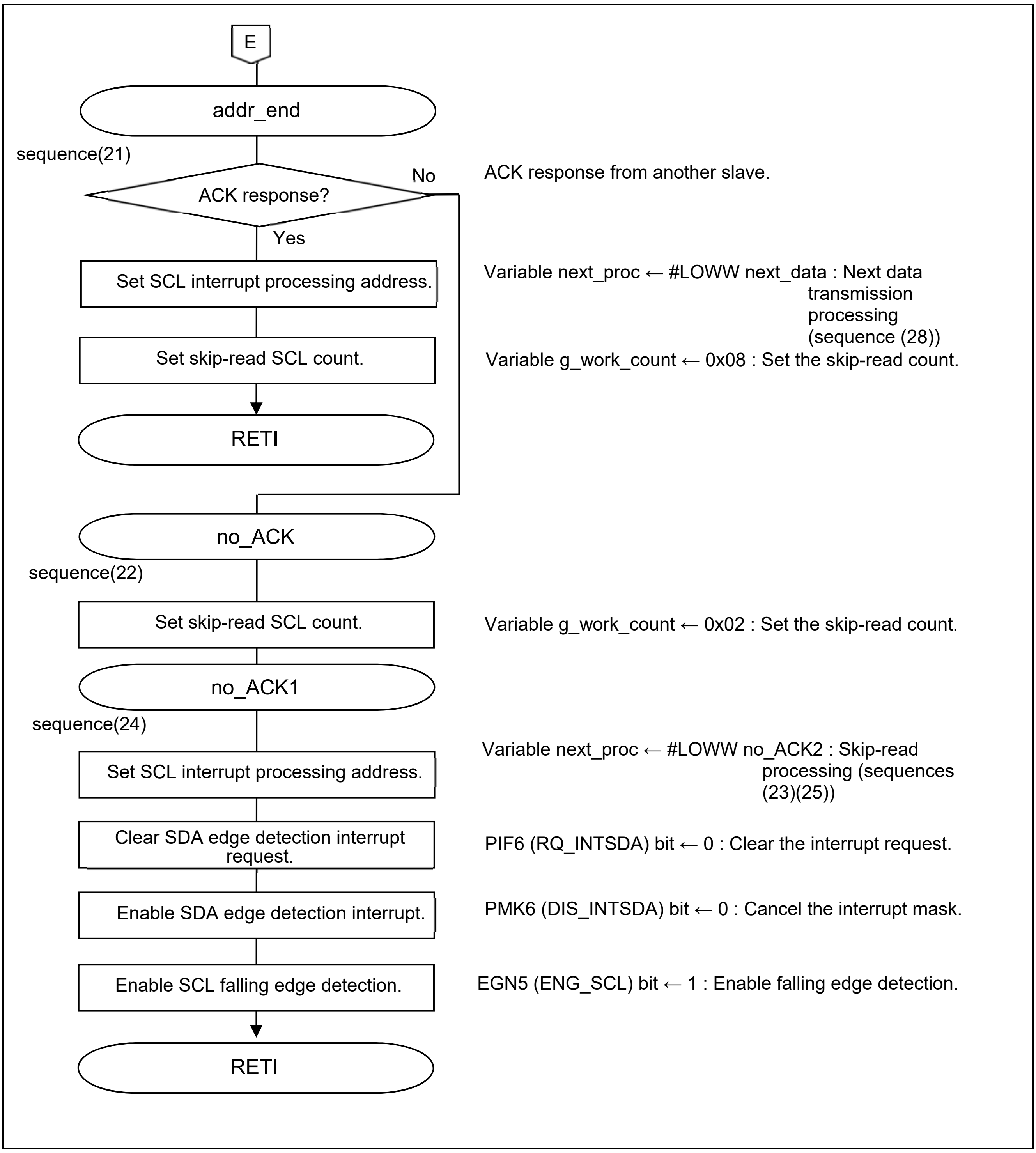

E
addr_end
sequence(21)
ACK response?
No
Yes
ACK response from another slave.
Set SCL interrupt processing address.
Variable next_proc ← #LOWW next_data : Next data transmission processing (sequence (28))
Set skip-read SCL count.
Variable g_work_count ← 0x08 : Set the skip-read count.
RETI
no_ACK
sequence(22)
Set skip-read SCL count.
Variable g_work_count ← 0x02 : Set the skip-read count.
no_ACK1
sequence(24)
Set SCL interrupt processing address.
Variable next_proc ← #LOWW no_ACK2 : Skip-read processing (sequences (23)(25))
Clear SDA edge detection interrupt request.
PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.
Enable SDA edge detection interrupt.
PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.
Enable SCL falling edge detection.
EGN5 (ENG_SCL) bit ← 1 : Enable falling edge detection.
RETI

Figure 5.48 Non-Selection Processing (1/2)

(15) Non-Selection Processing (2/2) (sequences (23), (25))

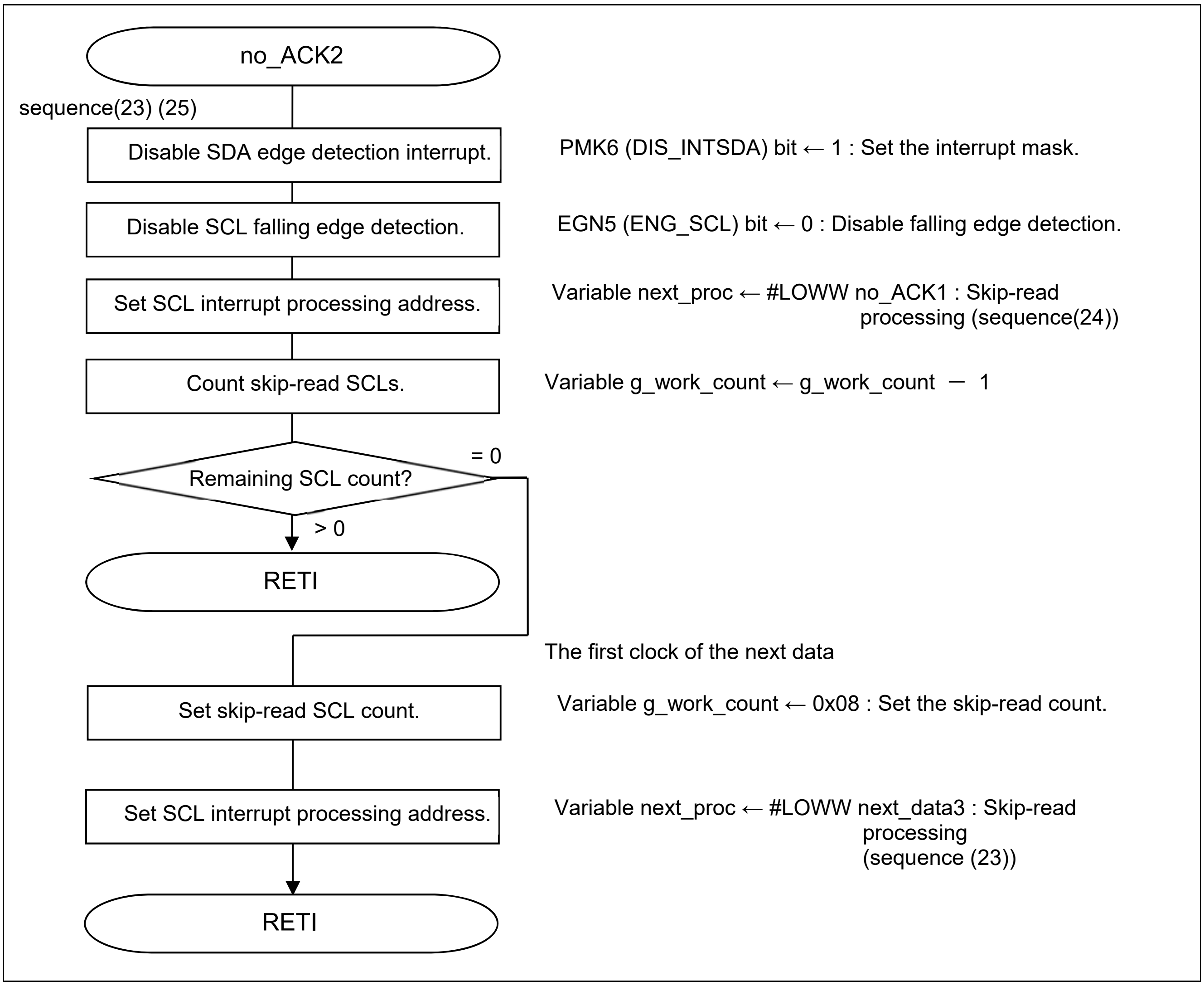

no_ACK2
sequence(23) (25)
Disable SDA edge detection interrupt.
PMK6 (DIS_INTSDA) bit ← 1 : Set the interrupt mask.
Disable SCL falling edge detection.
EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.
Set SCL interrupt processing address.
Variable next_proc ← #LOWW no_ACK1 : Skip-read processing (sequence(24))
Count skip-read SCLs.
Variable g_work_count ← g_work_count − 1
Remaining SCL count?
= 0
> 0
RETI
The first clock of the next data
Set skip-read SCL count.
Variable g_work_count ← 0x08 : Set the skip-read count.
Set SCL interrupt processing address.
Variable next_proc ← #LOWW next_data3 : Skip-read processing (sequence (23))
RETI

Figure 5.49 Non-Selection Processing (2/2)

(16) Skip-Read Processing 2 (sequence(26))

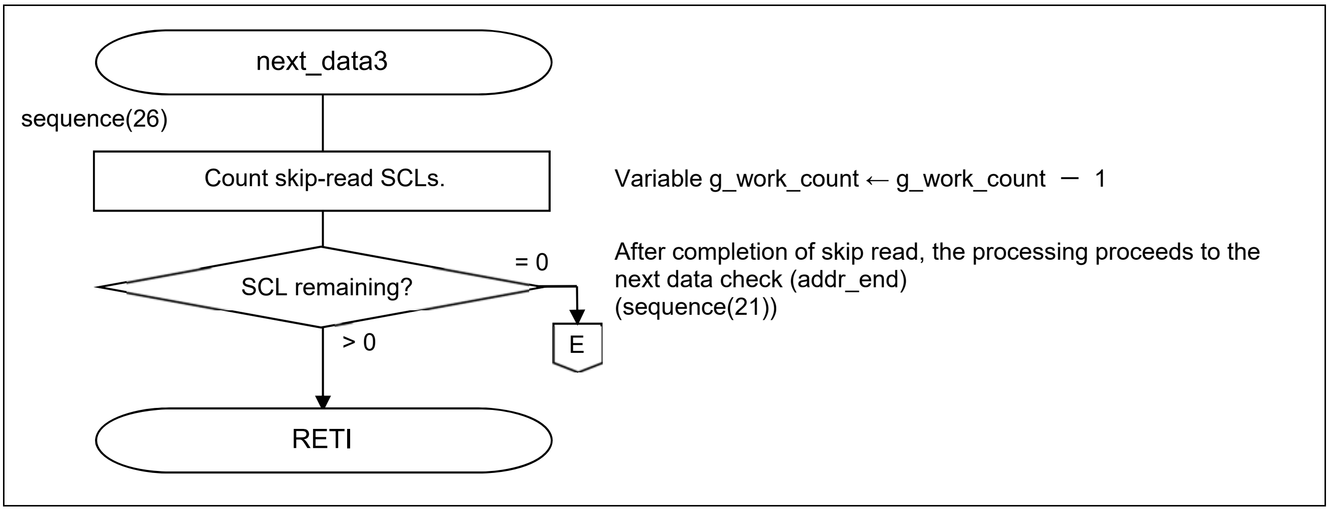

next_data3
sequence(26)
Count skip-read SCLs.
Variable g_work_count ← g_work_count − 1
SCL remaining?
= 0
> 0
E
After completion of skip read, the processing proceeds to the next data check (addr_end) (sequence(21))
RETI

Figure 5.50 Skip-Read Processing 2

(17) Skip-Read Processing 3 (sequence(28))

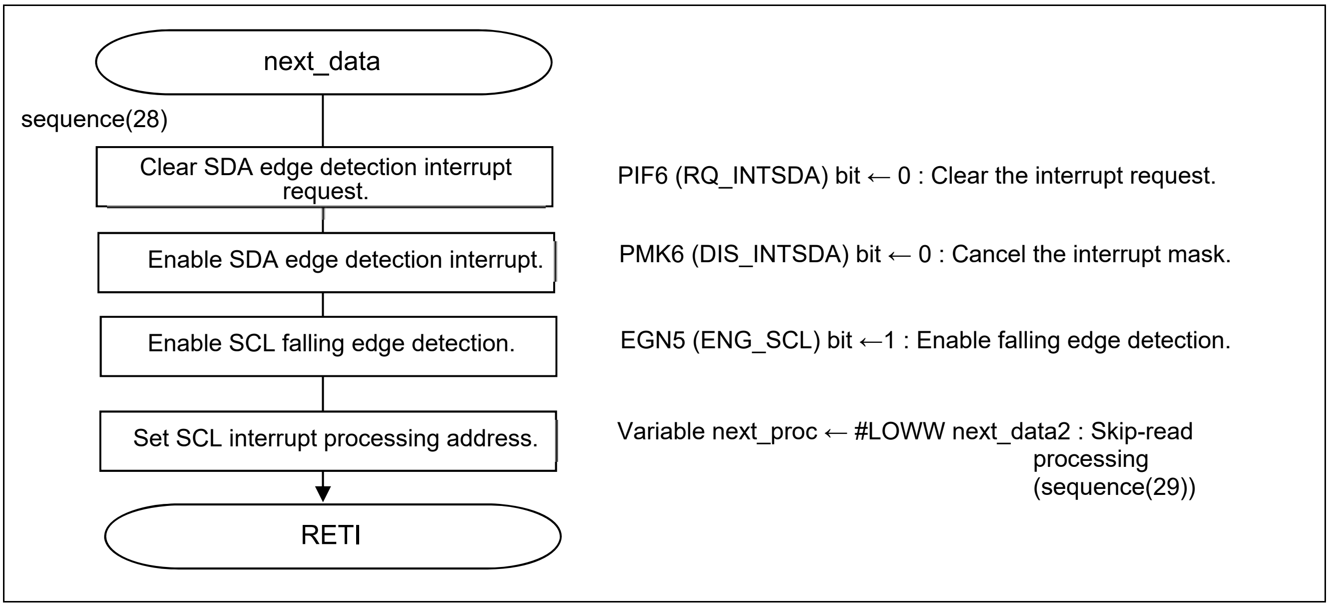

next_data
sequence(28)
Clear SDA edge detection interrupt request.
Enable SDA edge detection interrupt.
Enable SCL falling edge detection.
Set SCL interrupt processing address.
RETI
PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.
PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.
EGN5 (ENG_SCL) bit ←1 : Enable falling edge detection.
Variable next_proc ← #LOWW next_data2 : Skip-read processing (sequence(29))

Figure 5.51 Skip-Read Processing 3

(18) Skip-Read Processing 4 (sequence(29))

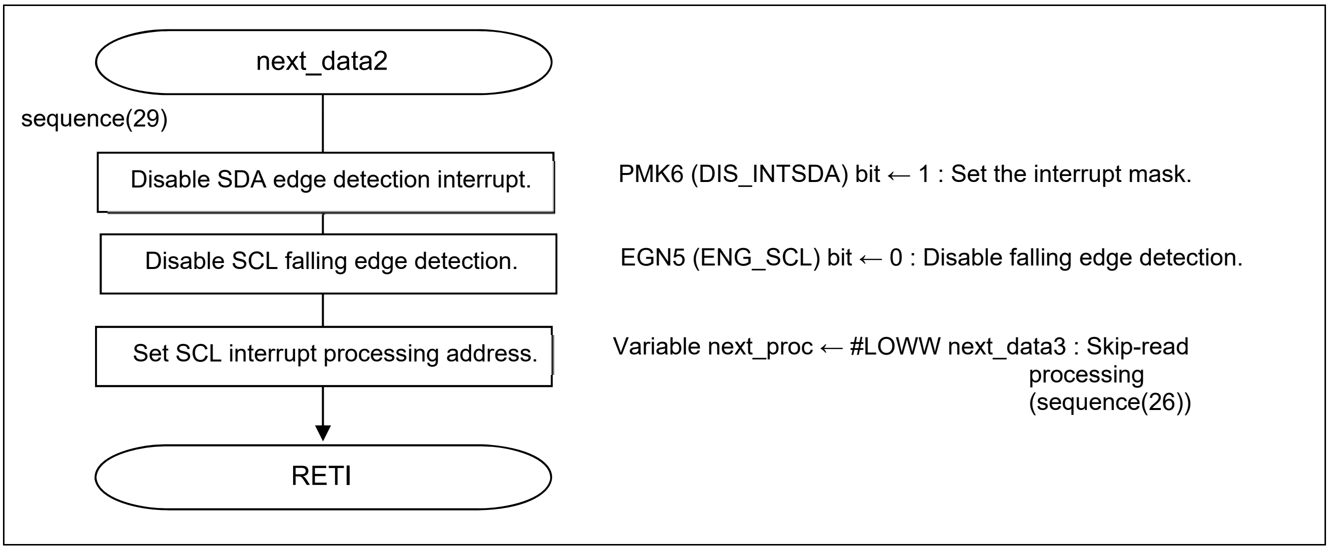

next_data2
sequence(29)
Disable SDA edge detection interrupt.
Disable SCL falling edge detection.
Set SCL interrupt processing address.
RETI
PMK6 (DIS_INTSDA) bit ← 1 : Set the interrupt mask.
EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.
Variable next_proc ← #LOWW next_data3 : Skip-read processing (sequence(26))

Figure 5.52 Skip-Read Processing 4

5.7.30 Starting Next Data Transmission

Figure 5.53 shows the flowchart for starting the next data transmission.

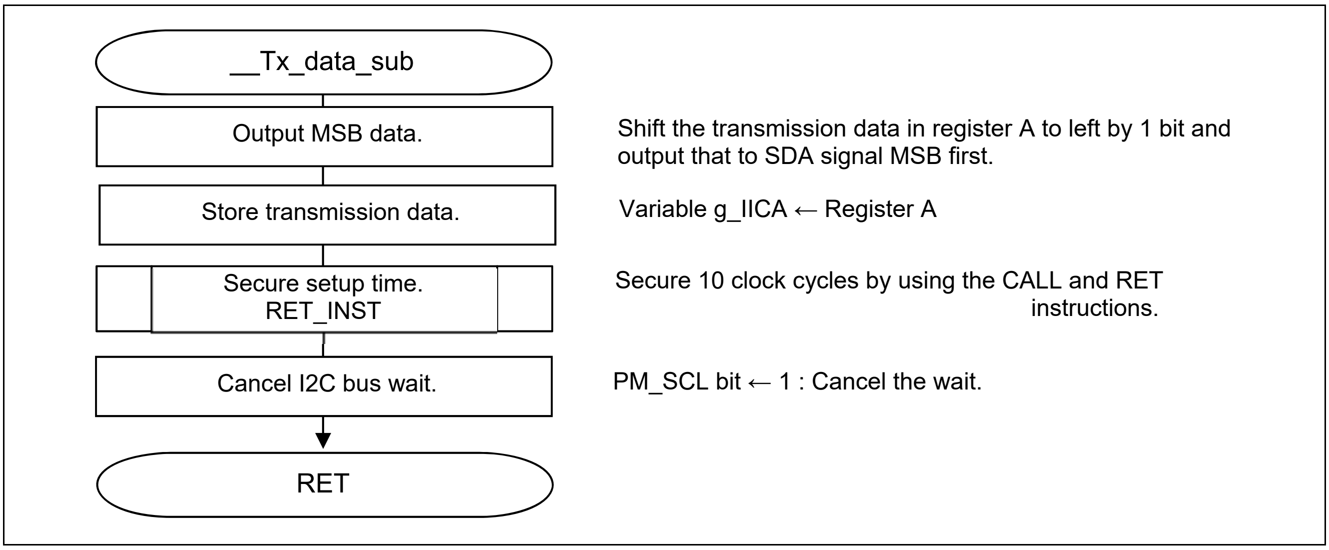

__Tx_data_sub
Output MSB data.
Shift the transmission data in register A to left by 1 bit and output that to SDA signal MSB first.
Store transmission data.
Variable g_IICA ← Register A
Secure setup time.
RET_INST
Secure 10 clock cycles by using the CALL and RET instructions.
Cancel I2C bus wait.
PM_SCL bit ← 1 : Cancel the wait.
RET

Figure 5.53 Next Data Transmission Start Processing

5.7.31 Starting Next Data Reception

Figure 5.54 shows the flowchart for starting the next data reception.

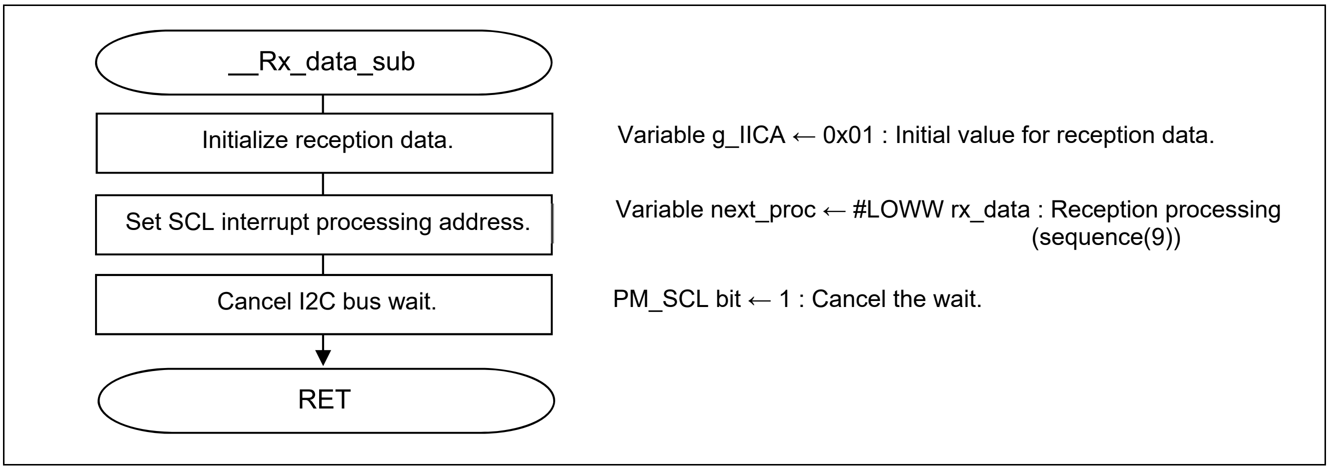

__Rx_data_sub
Initialize reception data.
Variable g_IICA ← 0x01 : Initial value for reception data.
Set SCL interrupt processing address.
Variable next_proc ← #LOWW rx_data : Reception processing (sequence(9))
Cancel I2C bus wait.
PM_SCL bit ← 1 : Cancel the wait.
RET

Figure 5.54 Next Data Reception Start Processing

5.7.32 Aborting Data Transmission

Figure 5.55 shows the flowchart for aborting data transmission.

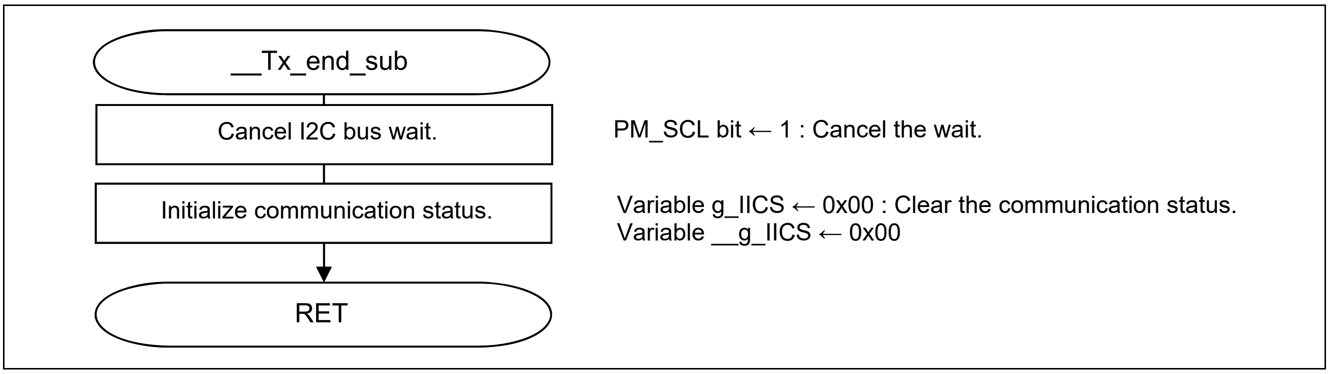

__Tx_end_sub
Cancel I2C bus wait.
PM_SCL bit ← 1 : Cancel the wait.
Initialize communication status.
Variable g_IICS ← 0x00 : Clear the communication status.
Variable __g_IICS ← 0x00
RET

Figure 5.55 Data Transmission Abort Processing

6. I2C Bus Basics

With the I2C bus, the I2C bus master controls communication. Slaves transmit or receive data according to the instructions from the master. Slaves can only send the ACK or NACK response to the data transmitted from the master, and pull the SCL signal low to keep the master waiting for synchronization with the master. However, some masters do not support the wait function and special approach is necessary in this case.

Slaves do not necessarily always follow the master. Slaves follow the master's instructions with respect to the communication protocol details; however, in the upper layer, the master is required to meet the slave's specifications. This is because the functions that slaves provide to the master via the I2C bus are specified by slaves.

Therefore, it is first defined what functions to provide as the slave. The master performs communication according to the definition.

As described above, with slaves, the functions provided determine the processes for access from the I2C bus, and thus hierarchical I2C bus control with slaves is difficult, unlike with the master. The slave performs the appropriate process according to the instructions from the master.

6.1 Communication Implementation through Software

The RL78/I1D does not have the I2C communication functions corresponding to the slaves. Therefore, to connect it as the slave to the I2C bus, it is necessary to prepare the program processes using ports and external interrupts. In this case, there are some limitations on the communication speed, conditions, and signals.

Figure 6.1 shows the SCL signal standard in this application note. The values shown here must also be applied to the setup time and hold time of the start condition and stop condition.

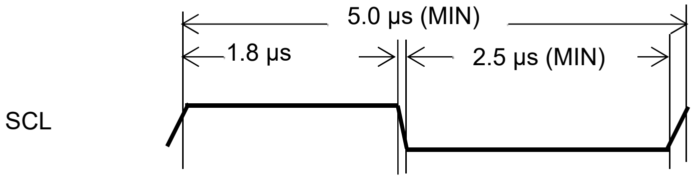


Figure 6.1 Example of Corresponding SCL Waveform

The ports and interrupts shown in figure 6.2 are used. Note that P32 and P33 are not provided with the function to set N-ch O.D. output. The same function is implemented by setting 0 to the output latch and controlling it through PM.

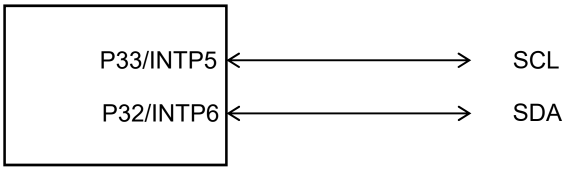


Figure 6.2 Pins Used

Implementation through software requires considerable CPU power. Particularly, special consideration is required since the I2C bus state needs to be constantly monitored even when the CPU itself is not selected.

To perform communication while monitoring the I2C bus state, the appropriate response time to the signal change is important. Since the delay caused by other interrupts has a significant influence, it is necessary to give top priority to the INTP5 and INTP6 interrupts and to enable the other interrupt processes at the beginning of the process. Therefore, for the other interrupts, `enable=true` is additionally declared on #pragma interrupt declaration.

In addition, the I2C hardware control processing part is independent as a library, and is written in the assembly language. For convenient use, the interface part written in the C language is provided, which enables easy use of the library also from the program written in the C language.

6.2 Functions as Slaves

6.2.1 LED Display Function

As the LED display device, eight LEDs are used to display 8-bit data. Two-byte data can be displayed, and either of the 2 bytes of data can be specified according to the SW input. While SW is not being pressed, data at register address 0x00 is displayed and while SW is being pressed, data at register address 0x01 is displayed.

Data is divided into the upper 4 bits and lower 4 bits and displayed in a time-division manner. The display frequency is 10 ms. Display data from the master is fixed as display data when the master issues the stop condition. The fixed data can be displayed after 50 ms at the latest.

6.2.2 A/D Conversion Function

It is possible to convert 4-channel analog input and obtain the moving average of the 16 latest conversions. The specifications of A/D conversion are given below.

- Analog input:	4 channels (channels 0 to 3)
- Conversion method:	Continuous conversion in scan mode
- Conversion resolution:	12 bits
- Conversion time:	18 μs/channel
- Buffer:	16 data/channel (128 bytes in total)

Conversion result of channel 0 is first read out, and then that of 1, 2, and 3 are read out. The upper 4 bits and lower 8 bits of the 12-bit conversion result of each channel are read out in this order. After the lower 8 bits of channel 3 are read out, channel 0 is read out as shown in figure 6.3.

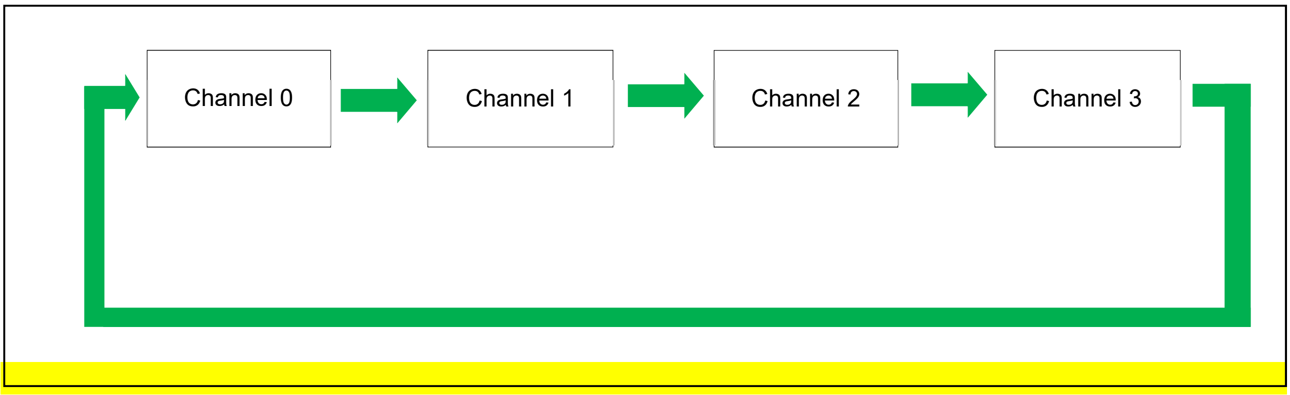


Figure 6.3 Reading A/D Conversion Results

6.2.3 RAM Function

The 128-byte area is provided to temporarily store 128 bytes of data. In the initial state, data is stored at 0x00 to 0x7F. When specified with slave address 0x70, RAM can be accessed. The data is written to RAM immediately after it is received. The address is automatically updated each time RAM is accessed. Access to address 0x7F is followed by access to address 0x00.

6.3 Library Interface Specifications

The library written in the assembly language provides the following three types of interfaces.

- Transmission/reception end flag
- Stop condition detection flag
- Communication restart processing function

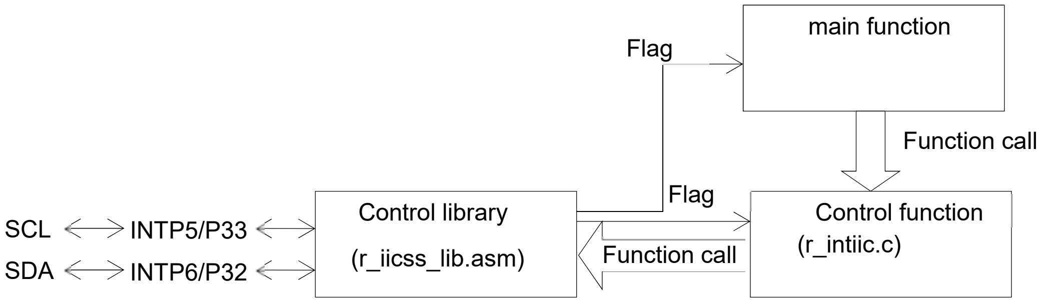


Figure 6.4 I2C Bus Control Structure

6.3.1 I2C Communication Flags

After completion of 1-byte data transmission/reception for the slave itself, the master is kept waiting to stop the communication, and then the following variables/flags are set.

- Variable _g_IICA: Stores data received in reception mode.
- Variable _g_IICS: Stores communication status, like the IICS0 register.
- Variable _g_IIC_IF: Flag indicating that 1-byte communication is completed. (transmission/reception end flag)

If the upper software checks the transmission/reception end flag to find that the flag is set, it refers to the communication status (variable _g_IICS) and executes the appropriate process.

After completing the process, the upper software calls the library functions (shown in section 6.3.2 Next Communication Starting Functions), prepares for the next communication, and cancels the I2C bus wait state to start the next communication.

6.3.2 Next Communication Starting Functions

The following three functions are provided to restart the I2C bus communication.

- _Rx_data_sub function: In the reception process, starts the next data reception.
- _Tx_data_sub function: In the transmission process, starts transmission of the data passed to the argument.
- _Tx_end_sub function: In responding to the NACK response from the master, cancels the wait state and withdraws from communication.

According to the communication status, one of the above functions is called.

6.3.3 Stop Condition Detection Flag

When the library written in the assembly language detects the stop condition, 0x01 is set to the variable _g_stop_det. Unlike the I2C status (variable _g_IICS), it remains set until cleared by the upper program. This is used in such applications that the process is started upon detection of the stop condition.

In the *main* process in this application note, the received data for turning on LED is sent to the program that processes turning on of LED, and is used as a trigger to actually process turning on of LED.

6.4 Slave Address Specification

6.4.1 Slave Address Table

This library holds information of the slave addresses to use as a table. The table is referred to with the upper 7 bits of the slave address transmitted from the master.

The upper 4 bits of the obtained value is the address ID (the lower 4 bits are 0). If the obtained value is 0x00, it means that the received address is not the address of that slave, and thus that slave does not participate in communication.

If the address ID type is one of 1 to F, it means that slave has been selected. In other words, 15 independent slave addresses can be used.

Although a single address ID can be assigned to multiple slave addresses, if the same address ID is used, the same process is basically applied.

The obtained address ID is set to the upper 4 bits of the communication status (variable _g_IICS). Figure 6.5 shows the communication status structure.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ID3	ID2	ID1	ID0	F_TRC	F_ACKD	F_STD	F_SPD

Figure 6.5 Communication Status Structure

The initial ACK response values and address table are stored in r_iicss_adr.asm as constant files. Figure 6.6 shows an example of the slave address table. In this example, 0x10 (address ID is 1), 0x20 (address ID is 2), 0x30 (address ID is 3), and 0x40 (address ID is 4) are set to 0x30 (address is 0x60), 0x38 (address is 0x70), 0x40 (address is 0x80), and 0x48 (address is 0x90), respectively.

```
SADR_TBL:                       ;0/8  1/9  2/A  3/B  4/C  5/D  6/E  7/F
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x00-0x07
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x08-0x0F
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x10-0x17
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x18-0x1F
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x20-0x27
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x28-0x2F
                .DB         0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x30-0x37
                .DB         0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x38-0x3F
                .DB         0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x40-0x47
                .DB         0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x48-0x4F
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x50-0x57
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x58-0x5F
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x60-0x67
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x68-0x6F
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x70-0x77
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x78-0x7F
```

Figure 6.6 Slave Address Table

6.4.2 ACK Response Flag

Each address ID has a flag to control the ACK response. 0x0F is set to send the ACK response, and 0x00 is set not to send the ACK response.

Figure 6.7 shows the ACK response table structure. In this example, address IDs 1 to 4 are set to send the ACK response.

```
ACK_TBL:
;                            0/8  1/9  2/A  3/B  4/C  5/D  6/E  7/F
                .DB         0x00,0x0F,0x0F,0x0F,0x0F,0x00,0x00,0x00 ; 0x00-0x07
                .DB         0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x08-0x0F
```

Figure 6.7 ACK Response Table Structure

6.5 Protocol for Accessing Slaves

6.5.1 Display on LED

Figure 6.8 shows how to access the slave for displaying data on LED.

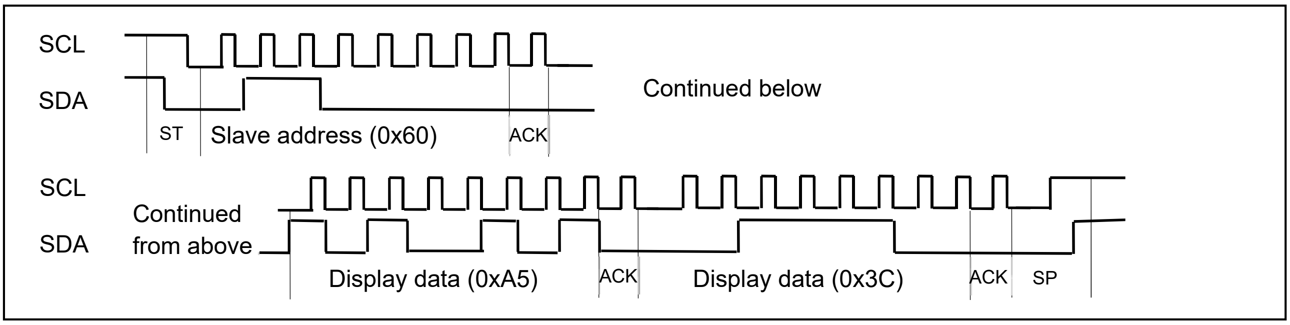


Figure 6.8 Timing of Writing LED Display Data

In this example, 0x3C is written to 0xA5 and 0x01 following the start condition (ST) and slave address (0x60). At the end, the stop condition (SP) is issued to inform the slave of transmission completion.

In responding to that, the slave sends ACK.

6.5.2 Reading A/D Conversion Results

Figure 6.9 shows how to access the slave for reading the A/D conversion results.

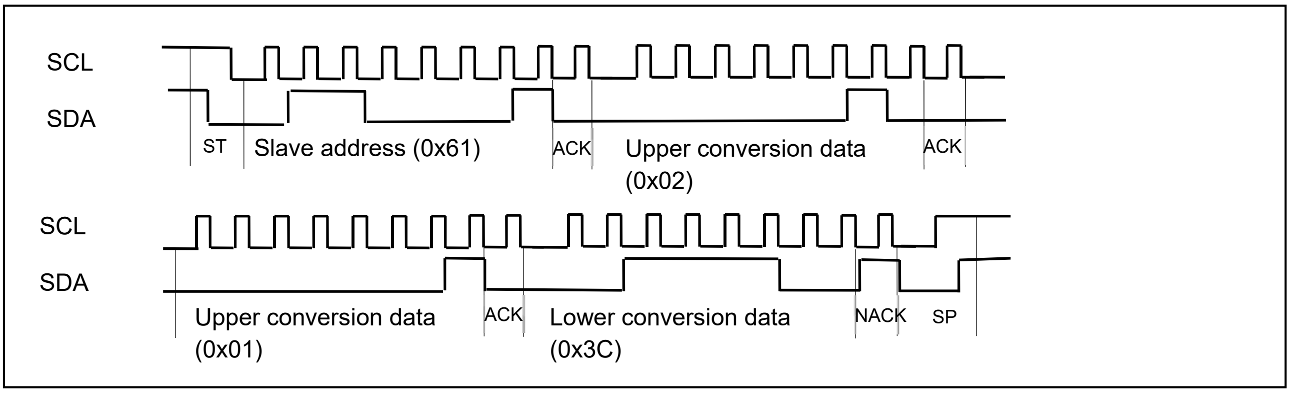


Figure 6.9 Timing of Reading A/D Conversion Results

In this example, by selecting them with the start condition (ST) and slave address (0x61) first, the upper and lower A/D conversion results of channel 0 are read out, in this order. In this figure, the upper A/D conversion result of channel 0 is 0x02.

In the timing shown at the bottom, the master returns the NACK response after reading the conversion result (0x013C), the slave determines it as completion of communication, and withdraws from communication. Finally, the master issues the stop condition (SP) and releases the I2C bus to complete communication.

When 4-channel A/D conversion is completed, the obtained moving average is set to the IIC buffer for conversion result transmission. Meanwhile, the latest conversion result can be obtained by repeatedly reading the A/D conversion results since the upper conversion result of channel 0 is read after the lower conversion result of channel 3 is read.

6.5.3 Reading Data from RAM

Figure 6.10 shows how to access the slave for reading data from RAM.

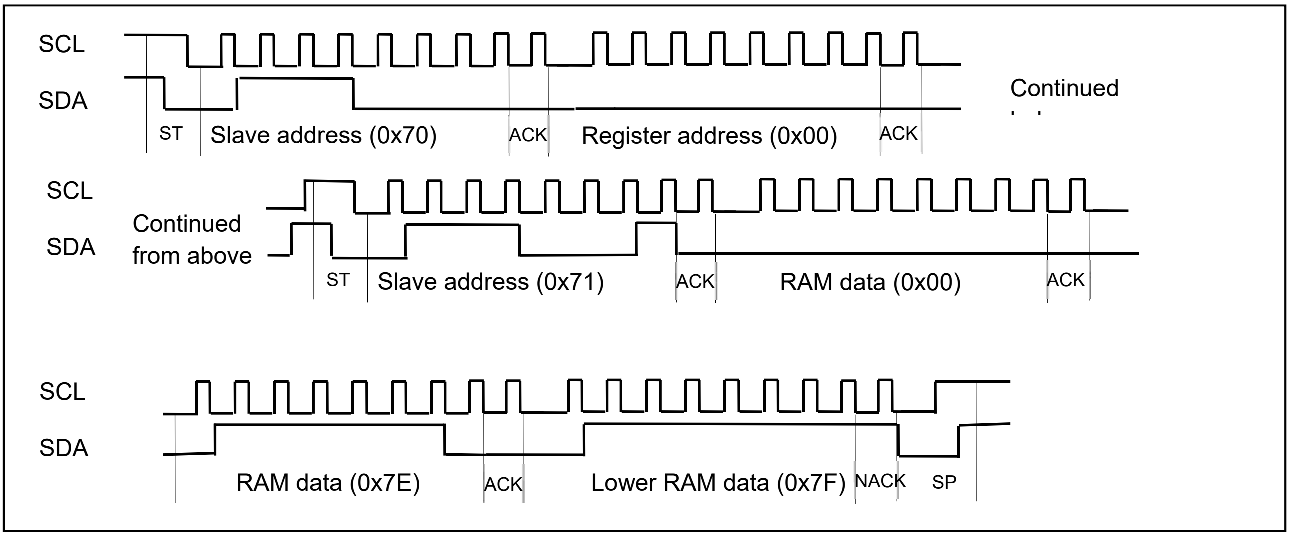


Figure 6.10 Timing of Reading Data from RAM

In this example, following the start condition (ST) and slave address (0x70), register address 0x00 (= RAM address 0x00) is specified.

Then, by restarting and selecting the slave (0x71) for reading, the value is read from the specified RAM address 0x00. In this example, the value at address 0x00 is 0x00.

In the timing shown at the bottom, the master returns the NACK response after reading the values from RAM addresses 0x7E and then 0x7F, the slave determines it as completion of communication, and withdraws from communication. Finally, the master issues the stop condition (SP) and releases the I2C bus to complete communication.

6.5.4 Writing Data to RAM

Figure 6.11 shows how to access the slave for writing data to RAM.

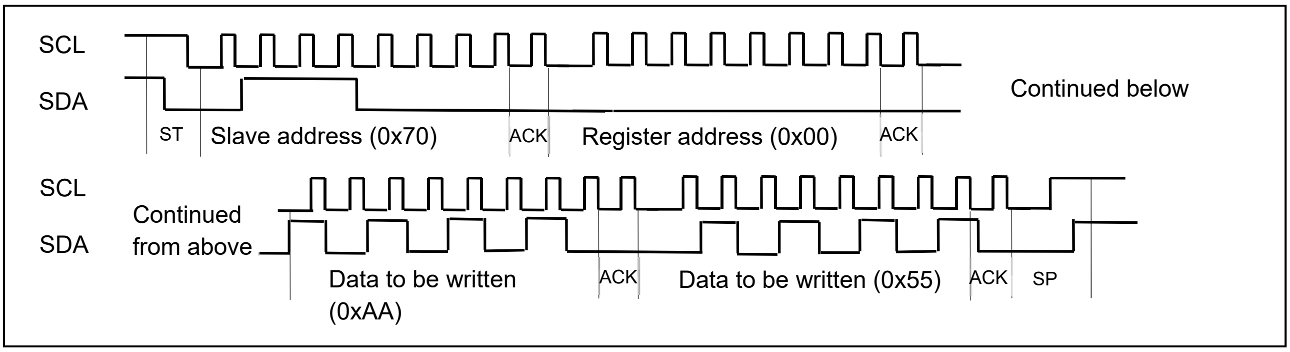


Figure 6.11 Timing of Writing Data to RAM

In this example, following the start condition (ST) and slave address (0x70), register address 0x00 is specified.

Then, 0xAA and 0x55 are transmitted as data to be written to addresses 0x00 and the next address, respectively. After transmitting 2-byte data and completing communication, the master issues the stop condition and releases the I2C bus.

7. Basic Control of I2C Bus through Software

To implement the I2C bus slave function through software, it is necessary to detect the rising and falling edges of the SCL and SDA signals.

For detection, INTP5 (SCL signal edge detection) and INTP6 (SDA signal edge detection) are used.

7.1 Edge Detection Interrupts

To process the detected edges in the limited time, the edge detection interrupts are given top priority over the other interrupts in the nested interrupt system. Even so, it takes 9 to 16 clock cycles for hardware to start the interrupt process. If the interrupt request is generated immediately after the lower priority interrupt is accepted, it takes another 9 clock cycles. Taking this into consideration, the process is performed as quickly as possible.

7.1.1 SCL Edge Detection

Basically, the SCL edge is used for data transmission/reception. Therefore, the edge to be detected is frequently switched between rising and falling to minimize the total processing time. Figure 7.1 shows the valid edges of the SCL and SDA signals when the address is received.

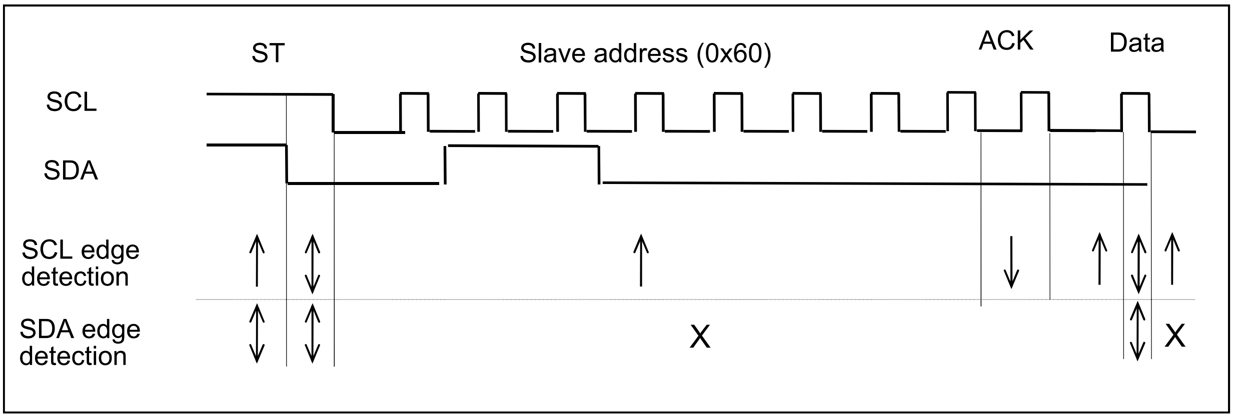


Figure 7.1 SCL and SDA Edges Used According to Timing

Until the start condition is detected, detection of the both edges of SDA is valid. Even after the stop condition is detected at the falling edge of the SDA signal, detection of all the edges is valid until the falling edge of the SCL signal is detected and slave address reception is started. This enables support of the case in which the stop condition is issued immediately after the start condition.

When slave address reception is started, the SDA signal edge detection interrupt is disabled. When slave address reception is completed at the rising edge of the eight SCL clock cycle, the addresses are compared. If the addresses agree, the program is used to wait for the SCL signal falling edge, and ACK response is started, and the edge of the SCL signal to be detected is changed to the falling edge. (The reason why the program is used to wait for the SCL signal falling edge is accepting the interrupt twice a cycle is too wasteful while there is much processing to do.)

After that, ACK response is ended at the falling edge of the SCL signal and slave address reception is completed. Detection of the rising edge of the SCL signal is enabled for the next.

In the period in which the next SCL signal is high (first clock cycle), detection of all the edges is valid because the start or stop condition may be issued.

7.1.2 SDA Edge Detection

Normal communication operation is sequential operation during SCL edge detection. Meanwhile, edge detection of the SDA signal is used to terminate the sequential operation. Therefore, when to enable acceptance is very important. As shown in figure 7.1, acceptance is enabled only while the SCL signal of the specific timing is high.

7.2 Control Processes

To reduce the processing time, the interrupt processing part is written in the assembly language. Besides, in the INTP5 interrupt process, the processing address according to the contents of the next process is set to the variable next_proc in advance.

This is because the process based on the SCL signal edge is sequential and thus the next process is limited. On the other hand, the process based on the SDA signal edge, which involves SP (stop condition) and ST (start condition), suspends the sequential process, like the interrupt process does.

Furthermore, at the beginning of the INTP5 interrupt process, the state of the input ports for the SCL and SDA signals are taken in the variable g_P_image.

To use such a process without paying any attention, the interface to the upper software is restricted.

7.2.1 Sequences based on SCL Edge Detection Interrupt (1)

Sequences are defined assuming sequence 1 is the state in which both the SCL and SDA signals are high before communication starts. Table 7.1 shows the sequences for address reception and non-selection.

Table 7.1 Processes for Address Reception and Non-Selection

Sequence No.	State/Process
Sequence 1	Initial state (SCL and SDA signals are high.)
Sequence 2	Detection of stop condition (waits for start condition next.)
Sequence 3	Detection of start condition (waits for SCL falling.)
Sequence 4	SCL falling after start condition detection (waits for address reception.)
Sequence 5	SCL rising edge (takes in slave address.)
Sequence 6	8th SCL rising (slave addresses agree.)
Sequence 6’	8th SCL rising (slave addresses disagree.)
Sequence 7	8th SCL falling (starts ACK response.)
Sequence 8	9th SCL falling (completes ACK response.)
Sequence 21	9th SCL rising (slave addresses disagree.)
Sequence 22	9th SCL rising without ACK response (waits for ST and SP.)
Sequences 23 and 25	9th or 1st SCL falling (discontinues waiting for ST and SP; skips reading SCL.)
Sequence 24	9th SCL rising (waits for ST and SP.)
Sequence 26	Counting SCL rising (waits for 9th SCL rising.)

7.2.2 Sequences based on SCL Edge Detection Interrupt (2)

Table 7.2 shows the reception process sequences. In the reception process, when 8-bit data is complete, the SCL bus is placed in the wait state to inform the upper process (received data is set in the variable _g_IICA, the communication status is set in the variable _g_IICS, and _g_IIC_IF is set to 0x01).

At the first SCL cycle, restart/stop or start of next data reception may occur; therefore, all the sources are enabled beforehand. If restart/stop is detected, the reception process sequence is cancelled. If SCL falls, it means continuation of reception, and thus SDA edge detection is disabled and process is continued.

Table 7.2 Reception Process Sequences

Sequence No.	State/Process	Remarks
Sequence 8’	9th SCL falling (informs the upper software of information at reception completion.)	_Rx_data_sub is performed next.
_Rx_data_sub	Releases I2C bus from wait state.	Sequence 9 is performed next.
Sequence 9	1st SCL rising (takes in received data.)	Sequence 11 is performed next.
Sequence 6	8th SCL rising (prepares for ACK response.)	
Sequence 7	8th SCL falling (starts ACK response.)	Sequence 8→8’ is performed next.
Sequence 11	Takes in data at SCL rising (waits for ST, SP, and SCL falling edge.)	Sequence 6 is performed next.
Sequence 10	1st SCL falling (discontinues waiting for ST and SP.)	Sequence 11 is performed next.

Having processed the received data in the variable _g_IICA, the upper process calls the function _Rx_data_sub to release the I2C bus from the wait state and restart the next reception.

7.2.3 Sequences based on SCL Edge Detection Interrupt (3)

Table 7.3 shows the transmission process sequences. In the transmission process, after ACK response upon agreement of slave addresses, the SCL bus is placed in the wait state to inform the upper process (the communication status is set in the variable _g_IICS and _g_IIC_IF is set to 0x01).

Table 7.3 Transmission Process Sequences

Sequence No.	State/Process	Remarks
Sequence 14	Selection for transmission (informs the upper software of information upon address reception completion.)	_Tx_data_sub is performed next.
_Tx_data_sub	Outputs1st bit; releases I2C bus from wait state.	Sequence 15 is performed next.
Sequence 15	1st SCL rising; clears ACKD, STD, and SPD.	Sequence 16 is performed next.
Sequence 16	SCL falling (data transmission timing; repeats 7 times.)	
Sequence 17	8th SCL falling (starts ACK reception.)	Sequence 18 is performed next.
Sequence 18	Receives ACK at 9th SCL rising.	
Sequence 19	Detects ACK at 9th SCL rising.	Sequence 14 is performed next.
Sequence 20	9th SCL falling (informs the upper software of NACK.)	_Tx_end_sub is performed next.
_Tx_end_sub	Clears communication status and releases I2C bus.	

Having set the transmission data in the argument (A register), the upper process calls the function _Tx_data_sub and outputs the next data to SDA to release the I2C bus from the wait state and restart the next data transmission.

For slave transmission, the slave is driving the SCA signal except for the 9th clock cycle; therefore, when the master is to perform any operation, it is necessary to return NACK response to the slave to stop transmission. When the slave receives NACK, it informs the upper software of NACK response, and releases the I2C bus from the wait state by using the function _Tx_end_sub called by the upper software to withdraw from communication (SCL edge detection is disabled).

7.3 I2C Slave File Configuration

The library of the I2C slave functions provided by this software consists of the following three files.

- r_iicss_lib.asm:	Program body that controls I2C (recommended not to be modified)
- r_iicss_adr.asm:	Definitions including slave addresses on the I2C bus. Modify as necessary.
- r_intiic.c:	Equivalent part of INTIICA0 processing part in IICA0. Write the processes using the I2C bus. In this sample program, the functions described in 6.2 Functions as Slave are implemented. To use the other functions, modify the processes here.

8. Settings through Code Creation

Set the following items with "API function output control" of "File creation mode" of Property set to "Output only initialization functions".

(1) Clock generation circuit settings

(a) Pin assignment settings: Fix as they are.

(b) Clock settings

- Operating mode settings: High-speed main mode 2.7(V) ≤ VDD ≤ 3.6(V)
- Main system clock (fMAIN) settings: High-speed on-chip oscillator clock (fIH)
- High-speed on-chip oscillator clock settings: 24 (MHz)
- Middle-speed on-chip oscillator clock settings: Do not check "operate".
- High-speed system clock settings: Do not check "operate".
- Subsystem clock (fSUB) settings: Do not check "operate".
- Low-speed on-chip oscillator clock (fIL) settings: Frequency 15 (kHz)
- RTC, FMC, interval timer, PCLBUZ operating clock settings: fIL
- CPU and peripheral clock settings: 24000 (fIH) (kHz)

(c) On-chip debugging settings

- On-chip debugging operation settings: Use.
- RRM function settings: Do not use.
- Security ID settings: Set security ID.
- Settings upon security ID authentication failure: Delete flash memory data.

(d) Reset source checking

- Reset source checking function output: Remove the check mark.

(e) Safety functions: Select "Do not use" for all.

(f) Data flash: Prohibit access to data flash.

(2) Port settings

- Set P0.0 to P0.3 to output. (data: 0)
- Set P6.0 and P6.1 to output 1.

Leave the other ports as default (Do not use).

(3) Timer settings

(a) General settings Channel 3: Interval timer

(b) Channel 3

- Operating mode settings: 16 bits
- Interval time (16 bits) settings: 50 ms

Leave the other settings as default.

(4) Frequency measurement circuit settings

Leave as default (Do not use).

(5) 12-bit interval timer settings

Leave as default (Do not use).

(6) 8-bit interval timer settings

Leave as default (Do not use).

(7) Clock output/buzzer output settings

Leave as default (Do not use) for all.

(8) Watchdog timer settings

- Operation settings in HALT/STOP/SNOOZE mode: Stop.

- Watchdog timer operation settings: Do not use.

(9) A/D converter settings

- A/D converter operation settings: Use
- Comparator operation settings: Enable
- Resolution settings: 10 bits
- VREF(+) settings: AVDD
- VREF(-) settings: AVSS
- Trigger mode settings: Software trigger mode
- Operating mode settings: Continuous scan mode
- ANI0 – ANI3 analog input pin settings: ANI0 – ANI3
- ANI16 – ANI18 analog input pin settings: Remove all check marks.
- Conversion start channel settings: ANI0 – ANI3
- Reference voltage: 2.7V ≤ AVDD ≤ 3.6V
- Conversion time mode: Standard 1
- Conversion time: 9 (216/fCLK) (μs)
- Conversion result upper/lower limit settings: Interrupt request signal (INTAD) generated when ADLL ≤ ADCRH ≤ ADUL
- Interrupt settings: Check the A/D interrupt enable (priority level = 1)

(10) Comparator settings

Leave as default (Do not use).

(11) Op amp. settings

Leave as default (Do not use).

(12) Serial array unit settings

Leave as default (Do not use).

(13) Data operation circuit settings

Leave as default (Do not use).

(14) Data transfer controller settings

Leave as default (no check marks).

(15) Event link controller settings

Leave as default (no check marks).

(16) Interrupt settings

- INTP5 setting: Rising edge, priority level: high

- INTP6 setting: Both edges, priority level: high

Leave the other settings as default (no check marks).

(17) Key interrupt function settings

Leave as default (no check marks).

(18) Voltage detection circuit settings

- Voltage detection operation settings: Use.
- Operating mode settings: Reset mode
- Detected voltage settings: 2.75 (V)

9. Sample Code

The user can get the sample code from the Renesas Electronics website.

10. Reference Documents

RL78/I1D User’s Manual: Hardware Rev.2.10 (R01UH0474J)

RL78 Family User’s Manual: Software Rev.2.00 (R01US0015J)

(Get the latest version from the Renesas Electronics website.)

Technical Updates/Technical News

(Get the latest information from the Renesas Electronics website.)

Website and Support

Renesas Electronics Website
http://japan.renesas.com/

Inquiries
http://japan.renesas.com/inquiry

Revision History

Rev.	Date	Revision Contents	
		Page	Point
1.00	2016.11.15	—	Newly created.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

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

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

2. Processing at Power-on

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

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

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

4. Clock Signals

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

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

5. Differences between Products

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

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "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 etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0