V850ES/Jx3-L

I^{2}C Bus EEPROM Control

R01AN1551EJ0100
Rev. 1.00
Mar. 28, 2013

Introduction

This application note describes how to control an external EEPROM by using the I^{2}C bus features. The sample code shown in this application note writes and reads 4-byte (word) data to and from the EEPROM connected to the I^{2}C bus. In addition, the sample code compares the written data and read data and uses the result of comparison to control the LED display.

Target Devices

V850ES/JC3-L

V850ES/JE3-L

V850ES/JF3-L

V850ES/JG3-L

V850ES/JG3-L USB

When applying this application note to other microcontrollers, make the necessary changes according to the specifications of the microcontroller and verify them thoroughly.

Contents

1. Specifications

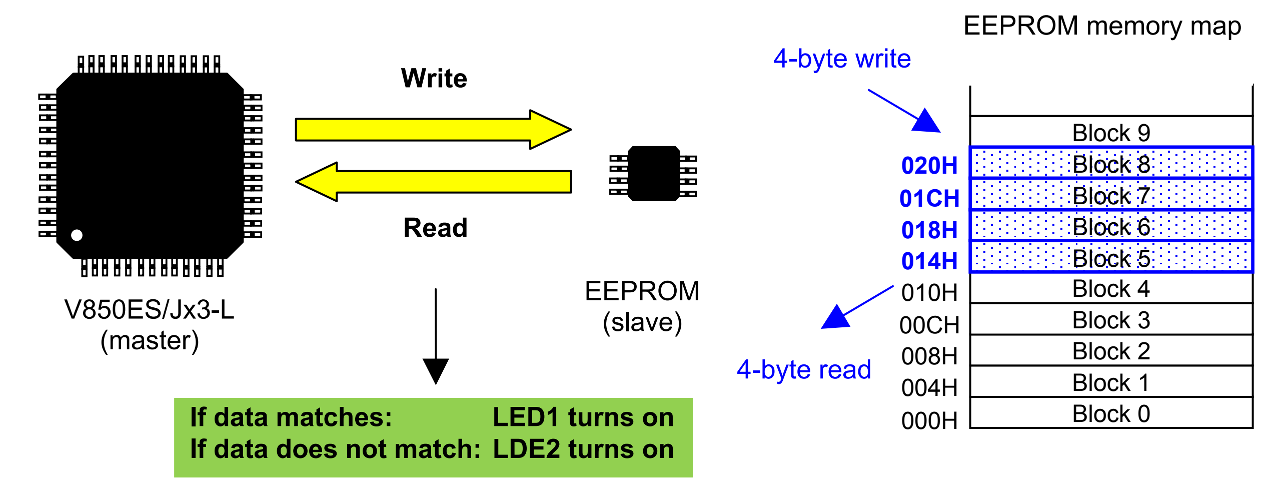
The sample code shown in this application note uses the I^2C bus features to write and read data to and from an external EEPROM connected to the I^2C bus. In addition, the sample code uses the result of comparing the written data and read data to control the LED display.

- Interrupt functions are used to implement API operations for controlling communication by using the I^2C bus features.
- Multiple I^2C communication control parameters are provided to enable control of different EEPROM sizes. The EEPROM size can be selected by the user, from 2 to 512 Kb. In this sample code, a 16 Kb EEPROM, R1EX24016A, and the communication control parameters for it are used to execute communication.
- The sample code writes 4-byte data to a specified EEPROM address by using I^2C communication and then reads the written data. The EEPROM can be accessed in block units, where one block consists of 4 bytes.
- Finally, the sample code compares the written data and read data to judge whether they match. If the data matches, the sample code turns on LED1 by controlling the relevant port. If the data does not match, the sample code turns on LED2.

Table 1.1 shows the peripheral functions used and their applications, and Figure 1.1 shows an overview of I^2C communication.

Table 1.1 Peripheral functions used and their applications

Peripheral Function	Application
I^2C00	I^2C master transmission/reception implemented by using the SCL00 and SDA00 pins



Write: 4 bytes (00H, 01H, 02H, and 03H) are written to address 014H (5th block) in the EEPROM.

Read: 4 bytes are read from address 014H (5th block) in the EEPROM.

Comparison: The written data and read data are compared. If data matches, LED1 is turned on; if data does not match, LED2 is turned on.

Figure 1.1 Overview of I^2C communication

2. Conditions Under Which Operation Has Been Verified

The operation of the sample code shown in this application note has been verified under the conditions shown below.

Table 2.1 Conditions under which operation has been verified

Item	Description
Microcontroller used	V850ES/JG3-L USB (μPD70F3796GC)
Operating frequency	• CPU clock: 16 MHz • Peripheral clock: 16 MHz • Main clock oscillation frequency: 6 MHz
Operating voltage	3.3 V (2.7 V to 3.6 V, when the CPU operates on 16 MHz)
Integrated development environment	CubeSuite+ V1.03.00 made by Renesas Electronics
C compiler	CA850 V3.50 made by Renesas Electronics
Board used	QB-V850ESJG3LUSB-TB + external EEPROM (R1EX24016)

Caution This sample code can be used with V850ES/Jx3-L devices. If you plan to use a device that does not conform to the conditions under which operation has been verified, be sure to choose the operating frequency and operating voltage that suit your device, by using features such as the code generator.

2.1 Notes on using CubeSuite+ code generator

This sample code uses the features of the CubeSuite+ code generator, but some source files have been added or deleted. When generating code for the project described in this document, perform the following:

- Remove `CG_serial_user.c` from the project.
- Add `IIC_EEPROM.c` and `CG_lk.dir` (existing files) to the project.

3. Related Application Notes

Related application notes are shown below. Also refer to these documents when using this application note.

V850ES/JG3-L Initialization (U19479EJ) Application Note

4. Hardware

4.1 Hardware configuration

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

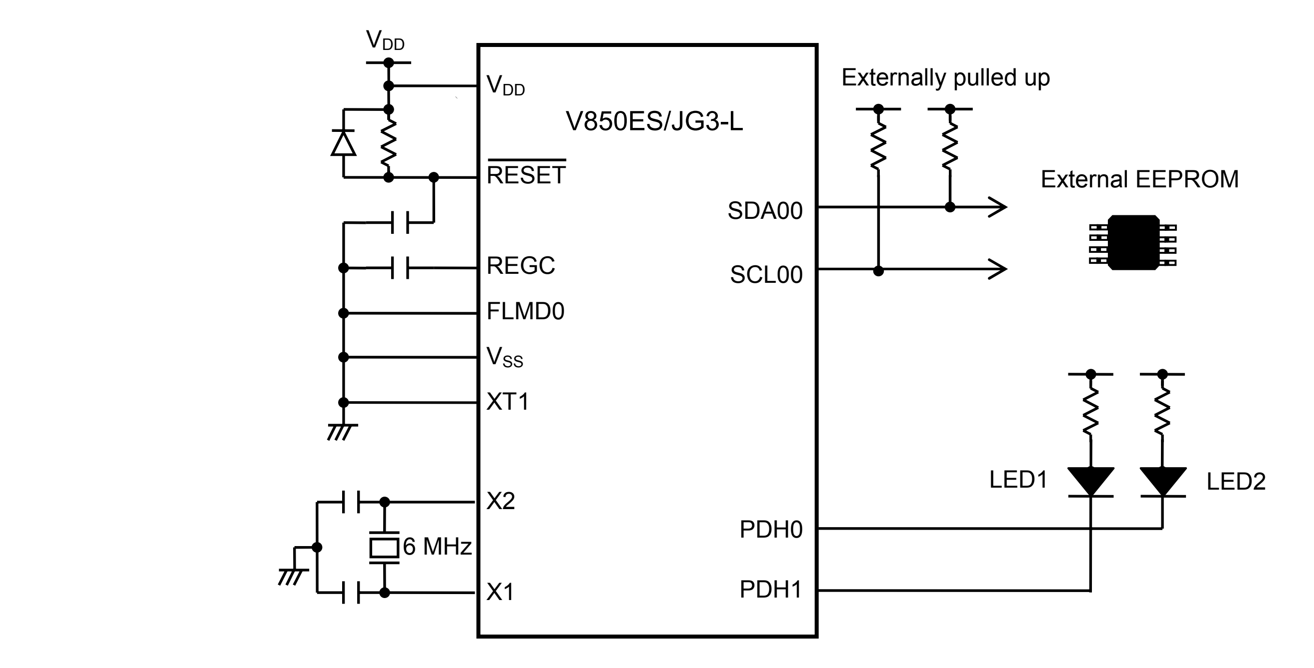


Figure 4.1 Hardware configuration

Cautions 1. This circuit diagram is simplified to show an overview of the circuit connections. When designing your actual circuit, connect pins appropriately so as to satisfy the electrical specifications.
2. Make the potential of the EV_{DD} pin and AV_{REF0} pin the same as V_{DD}.
3. Make the potential of the EV_{SS} pin the same as GND.
4. Connect REGC to GND via a capacitor (recommended value: 4.7 μF).
5. Connect the FLMD0 pin to GND in normal operating mode.

4.2 Pins used

Table 4.1 shows the pins used and their roles.

Table 4.1 Pins used and their roles

Pin Name	I/O	Description
P39/SCL00	I/O	Serial clock output pin for I^2C00
P38/SDA00	I/O	Serial data transmission/reception pin for I^2C00
PDH1	I/O	LED1 pin (LED1 turns on when data comparison results in match)
PDH0	I/O	LED2 pin (LED2 turns on when data comparison results in mismatch)

5. Software

5.1 Operation overview

The sample code in this application note controls (reads and writes) the EEPROM by using I^{2}C master transmission/reception via I^{2}C00. In addition, the sample code compares the data written to and read from the EEPROM and lights the LEDs accordingly.

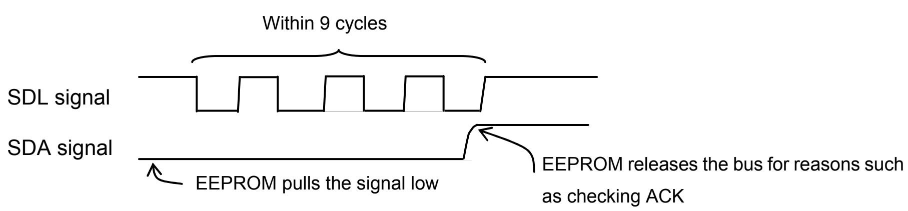
Setting conditions:

- Dividing the I^{2}C0 clock is enabled and the division factor is $f_{XX}/2$.
- The operating mode is high-speed mode. The digital filter is enabled.
- f_{CLK} is specified as the transfer clock.
- The local address is 80H.
- Generating a start condition is enabled after operation is enabled.
- Generating an interrupt request upon detection of a stop condition is disabled.
- An interrupt request is set to be generated at the falling edge of the 9th clock cycle.
- The acknowledge signal is enabled.
- The P39/SCL00 pin is used to output the transfer clock and the P38/SDA00 pin is used to transmit and receive data.

(1) Specify the I^{2}C00 initial settings.
(2) Set the interval counted by timer TMM0 to 100 μs. TMM0 is used to count the wait cycles when writing to the EEPROM.
(3) Specify the communication control parameters for the EEPROM used (16 Kb EEPROM).
(4) Enable I^{2}C00.
(5) Create 4-byte data (00H, 01H, 02H, and 03H).
(6) Specify the slave address (A0H) and the address to be accessed in the EEPROM (5th block (014H)) for the EEPROM access parameter (structure `g_PARAI`).
(7) Write data to the EEPROM.
(8) Read the data written to the EEPROM.
(9) Compare the written data and read data and light the LEDs according to the result (match or mismatch).

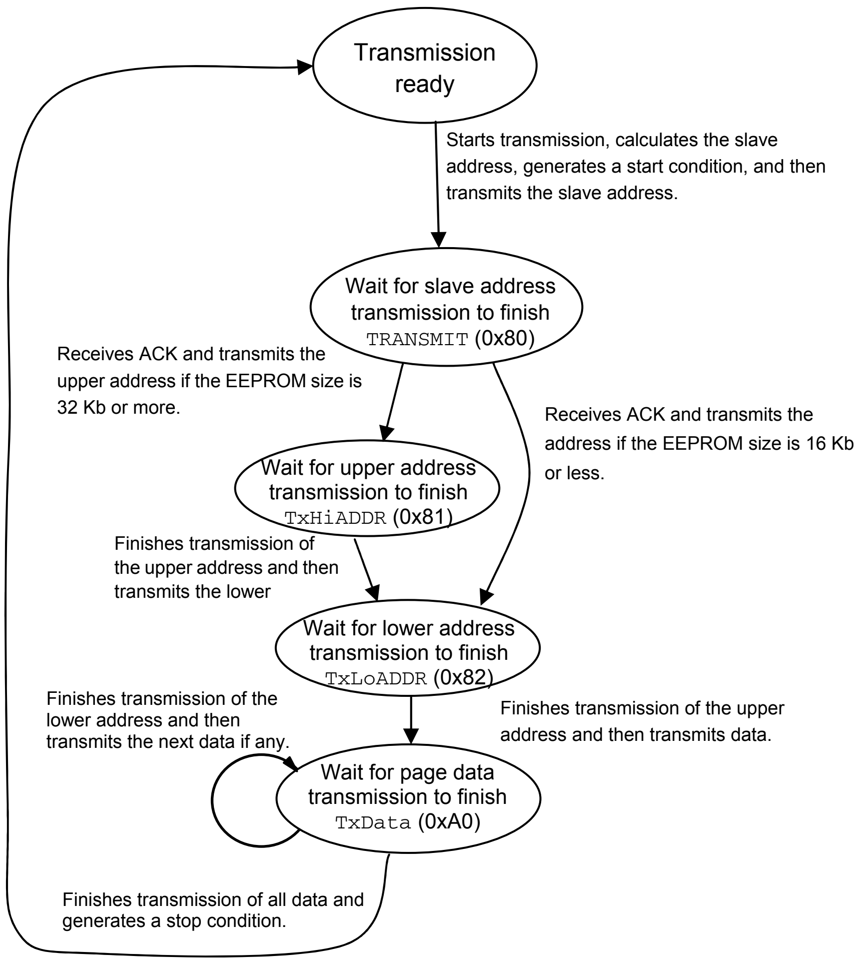
Caution This sample code simply shows an example of controlling the EEPROM (R1EX24016) connected to the I^{2}C bus by using I^{2}C00 of the V850ES/JG3-L. If you use another channel or EEPROM, thoroughly evaluate it before use.

Note If an attempt is made to release the bus by generating a stop condition on the I^{2}C bus, if the EEPROM, which is serving as a slave, is pulling the SDA signal low, the stop condition cannot be generated.
In this case, control the SCL signal by using the program to generate a pseudo I^{2}C bus clock for up to nine clock cycles; this stops the EEPROM driving the SDA signal (normal operation) and the SDA signal can then be pulled high.



5.1.1 EEPROM write processing

Figure 5.1 shows the status transitions during EEPROM write processing. The statuses are defined by the values of the variable g_comstatus (such as TRANSMIT (0x80)). During I²C communication, the current status is determined by referencing the value of g_comstatus during the processing of the transfer complete interrupt function, and the program branches to the next processing based on this value.



Caution Error handling is omitted in the above figure. The parameter below each state is the value of g_comstatus.

During I²C communication, the current status is determined during processing of the interrupt function and the program branches to the next processing based on this value.

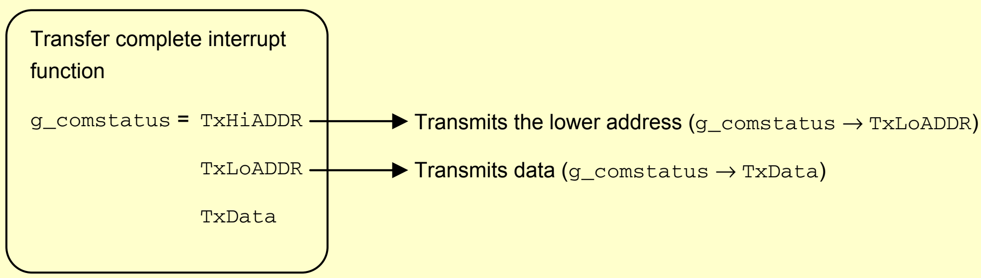
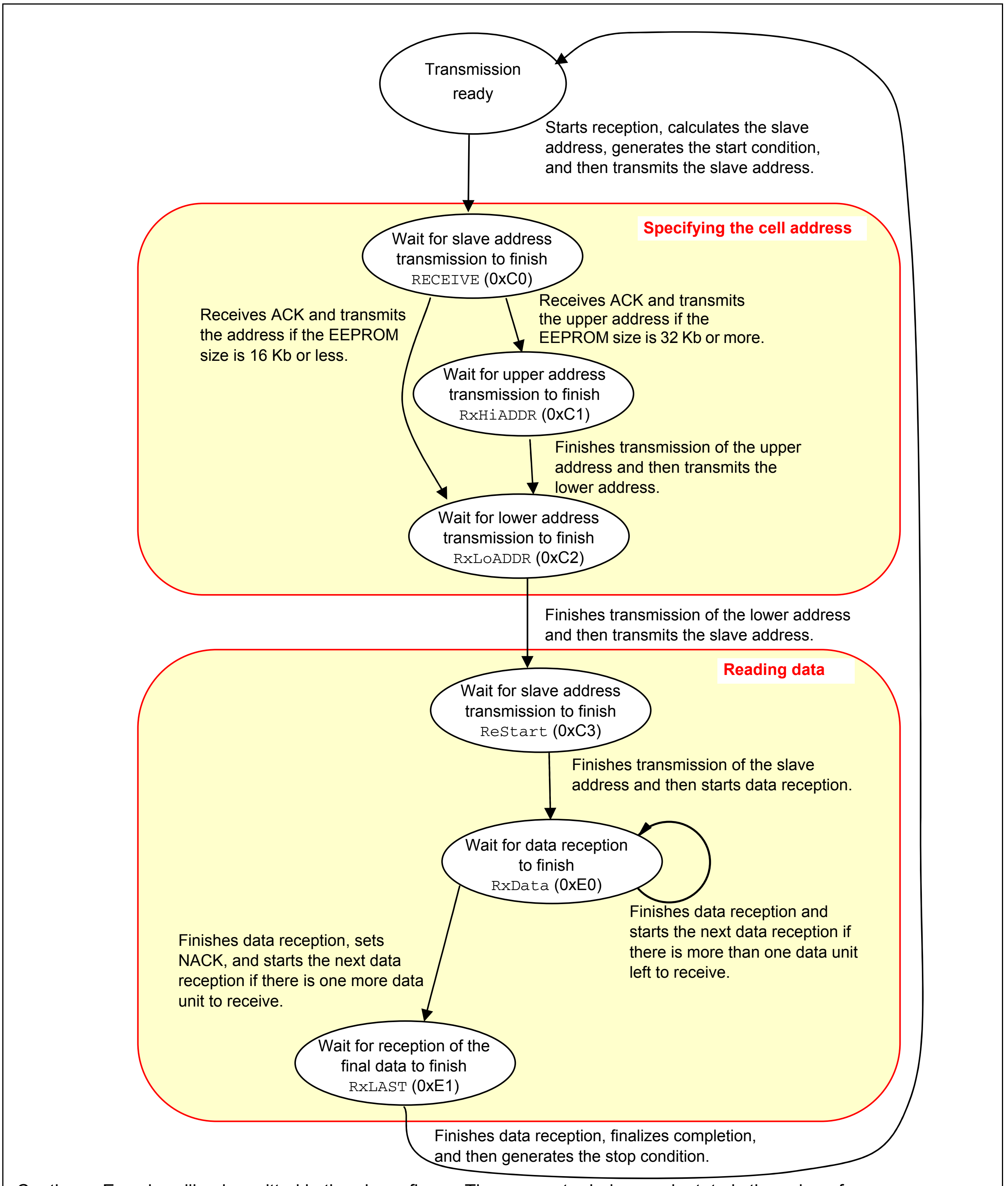


Figure 5.1 Status transition during EEPROM write processing

5.1.2 EEPROM read processing

Figure 5.2 shows the status transitions during EEPROM read processing. Like EEPROM write processing, the statuses are defined by the values of the variable g_comstatus (such as RECEIVE (0xC0)). EEPROM read processing consists of two main sections: specifying the cell address (by informing EEPROM of the EEPROM address to be read) and reading data (by specifying the master as the agent to receive data and generating a restart condition).



Caution Error handling is omitted in the above figure. The parameter below each state is the value of g_comstatus.

Figure 5.2 Status transition during EEPROM read processing

5.2 Option byte settings

Table 5.1 shows the option byte settings.

Table 5.1 Option byte settings

Address	Setting	Description
0000007AH	00000110B	Disable stopping the on-chip oscillator by using software (oscillation stabilization time: 10.92 ms (f_X = 6 MHz))
0000007BH to 0000007FH	00000000B	Specify signal for addresses 007BH to 007FH.

5.3 Constants

Table 5.2 shows the constants used in sample code.

Table 5.2 Constants used in sample code (1/2)

Constant Name	Setting	Description
BLOCK_SIZE	4	EEPROM data block access unit (bytes)
MEMORY_2K	0x0001	Capacity of 2 Kb EEPROM (in 256-byte units)
MEMORY_4K	0x0002	Capacity of 4 Kb EEPROM (in 256-byte units)
MEMORY_8K	0x0004	Capacity of 8 Kb EEPROM (in 256-byte units)
MEMORY_16K	0x0008	Capacity of 16 Kb EEPROM (in 256-byte units)
MEMORY_32K	0x0010	Capacity of 32 Kb EEPROM (in 256-byte units)
MEMORY_64K	0x0020	Capacity of 64 Kb EEPROM (in 256-byte units)
MEMORY_128K	0x0040	Capacity of 128 Kb EEPROM (in 256-byte units)
MEMORY_256K	0x0080	Capacity of 256 Kb EEPROM (in 256-byte units)
MEMORY_512K	0x0100	Capacity of 512 Kb EEPROM (in 256-byte units)
BLOCK_2K	See right	MEMORY_2K * 256 / BLOCK_SIZE
BLOCK_4K	See right	MEMORY_4K * 256 / BLOCK_SIZE
BLOCK_8K	See right	MEMORY_8K * 256 / BLOCK_SIZE
BLOCK_16K	See right	MEMORY_16K * 256 / BLOCK_SIZE
BLOCK_32K	See right	MEMORY_32K * 256 / BLOCK_SIZE
BLOCK_64K	See right	MEMORY_64K * 256 / BLOCK_SIZE
BLOCK_128K	See right	MEMORY_128K * 256 / BLOCK_SIZE
BLOCK_256K	See right	MEMORY_256K * 256 / BLOCK_SIZE
BLOCK_512K	See right	MEMORY_512K * 256 / BLOCK_SIZE
ADDR0BIT	0b00000000	Specifies that slave addresses are not be used as cell addresses
ADDR1BIT	0b00000001	Specify A8 by using bit 1 of the slave address
ADDR2BIT	0b00000011	Specify A9 and A8 by using bits 2 and 1 of the slave address
ADDR3BIT	0b00000111	Specify A10 to A8 by using bits 3 to 1 of the slave address
I2C_OK	0x00	Successful
PARA_ERR	0x20	Parameter error
COMP_ERR	0x21	Compare match error in read or written data
NO_ACK1	0x40	No ACK response for the slave address
NO_ACK2	0x41	No ACK response for the EEPROM address
NO_ACK3	0x42	No ACK response for the transmitted data
SVAMSK	0b11111110	Data for masking bit 0 of the slave address

Table 5.2 Constants used in sample code (2/2)

Constant Name	Setting	Description
R1EX24002A	0x00	Specify the EEPROM to be used These constants are defined by the enumeration constant eeprom_name and are used to allow the eeprom_info structure EEPROM_ADDRESS to reference EEPROM parameters.
R1EX24004A	0x01	
R1EX24008A	0x02	
R1EX24016A	0x03	
R1EX24032A	0x04	
R1EX24064A	0x05	
R1EX24128B	0x06	
R1EX24256B	0x07	
R1EX24512B	0x08	

5.4 Variables

Table 5.3 shows the global variables.

Table 5.3 Global variables

Data Type	Variable Name	Description	Function That Uses This Variable
Structure eeprom_paraA16	g_PARAI	Parameter for specifying EEPROM access	main() check_EEPROM_Addr()
uint8_t	g_comstatus	Operation information/result flag	main() check_EEPROM_Addr() R_EEPROM_R() R_EEPROM_wait_read() R_IIC0_Tx_addr1() R_IIC0_Tx_addr2() R_IIC0_Rx_RST() R_IIC0_RxData_ST() R_IIC0_RxData() R_IIC0_Rx_LastData() R_EEPROM_W() R_EEPROM_wait_write() R_IIC0_TxDataST() R_IIC0_TxData() MD_INTIIC0()
uint8_t array (BLOCK_SIZE)	g_data_bufferW	Write data buffer	main()
uint8_t array (BLOCK_SIZE)	g_data_bufferR	Read data buffer	main()
unit8_t	g_data_counter	Data counter	R_IIC0_TxDataST() R_IIC0_TxData() R_IIC0_RxData_ST() R_IIC0_RxData() R_IIC0_Rx_LastData()
unit8_t	g_celladdr	EEPROM internal address	
uint8_t	g_eeprom_type	Number of EEPROM used	R_device_select()
Structure eeprom_paraA16 uint8_t slaveaddr; uint16_t block_num;	g_PARAA	Parameter for accessing EEPROM	R_EEPROM_R() R_IIC0_Tx_addr1() R_IIC0_Tx_addr2() R_IIC0_Rx_RST() R_IIC0_RxData_ST() R_IIC0_RxData() R_IIC0_Rx_LastData() R_EEPROM_W() R_IIC0_TxDataST() R_IIC0_TxData()
Structure eeprom_paraA16	g_PARAC	Parameter for accessing work EEPROM (a copy of g_PARAA)	check_EEPROM_Addr() R_EEPROM_R() R_EEPROM_W() R_IIC0_TxData() get_slave_Addr()
Structure eeprom_info uint16_t rom_size; uint16_t total_block uint8_t addr_mask;	EEPROM_Info	Variable for saving parameters for the EEPROM used (for processing)	R_device_select() R_IIC0_Tx_addr1() get_slave_Addr()

5.5 Functions

Table 5.4 shows the functions.

Table 5.4 Functions

Function	Overview
R_device_select	This function specifies the EEPROM to be used.
R_EEPROM_R	This function reads data from the EEPROM based on the structure pointed to by the pointer passed by the EEPROM access parameter.
R_EEPROM_wait_read	This function is used to wait for EEPROM reading to finish.
R_EEPROM_W	This function writes data to the EEPROM based on the structure pointed to by the pointer passed by the EEPROM access parameter.
R_EEPROM_wait_write	This function is used to wait for EEPROM writing to finish.
check_EEPROM_Addr	This function checks whether the address specified by the parameter is within the EEPROM size. If the address is within the EEPROM size, this function copies the access parameter.
get_slave_Addr	This function incorporates the upper address of a cell into the slave address when using a 4 Kb to 16 Kb EEPROM.
R_IIC0_Tx_addr1	This function ends transmission to the slave address and transmits the EEPROM cell address.
R_IIC0_Tx_addr2	This function transmits the lower address of an EEPROM cell if it is 2 bytes.
R_IIC0_Rx_RST	After transmitting the EEPROM cell address, this function restarts communication to read the data.
R_IIC0_RxData_ST	This function writes dummy data to the IIC0 register and starts data reception.
R_IIC0_RxData	This function stores the received data in the buffer and starts the next data reception. In addition, this function disables the acknowledge signal before receiving the final data.
R_IIC0_Rx_LastData	This function stores the data received last, generates the stop condition, and then ends reception processing.
R_IIC0_TxDataST	This function starts transmission of data to be written to the EEPROM.
R_IIC0_TxData	If there is more data to send when transmission finishes, this function starts data transmission again. When all data units have been transmitted, this function generates the stop condition and instructs the EEPROM to write data.
R_IIC00_StartCondition	This function generates a start condition and waits for the start condition to be detected.
R_IIC00_StopCondition	This function generates a stop condition and waits for the stop condition to be detected.
MD_INTIIC0	This function checks the ACK response from the slave by using the IIC00 transfer complete interrupt and assigns the next processing according to the communication status.
R_IIC00Start	This function enables IIC00.

5.6 Function specifications

This section shows the specifications of the functions used in the sample code.

`R_device_select`

Overview	Specification of EEPROM to be used	
Header	`r_cg_macrodriver.h` `r_cg_userdefine.h`	
Declaration	`MD_STATUS R_device_select(enum eeprom_name);`	
Description	This function copies the EEPROM parameter specified by the parameter to the `EEPROM_Info` structure.	
Parameters	Name of EEPROM	Name defined by enumeration constant `eeprom_name`
Return value	`I2C_OK`:	Successful
	`PARA_ERR`:	Incorrect name was specified
Remark	None	

`R_IIC00_Start`

Overview	Enabling IIC00
Header	`r_cg_macrodriver.h` `r_cg_userdefine.h`
Declaration	`void R_IIC00_Start(void)`
Description	This function enables IIC00.
Parameters	None
Return value	None
Remark	None

`R_EEPROM_R`

Overview	Reading data from the EEPROM	
Header	`r_cg_macrodriver.h` `r_cg_userdefine.h`	
Declaration	`void R_EEPROM_R(struct eeprom_paraA16 *PARA);`	
Description	This function reads data from the address specified by the structure pointed to by the parameter.	
Parameters	`*PARA`	Pointer to structure `eeprom_paraA16`
Return value	None	
Remark	Processing to wait for reading to finish is performed by `R_EEPROM_wait_read`.	

`R_EEPROM_wait_read`

Overview	Waiting for reading from EEPROM to finish	
Header	`r_cg_macrodriver.h` `r_cg_userdefine.h`	
Declaration	`MD_STATUS R_EEPROM_wait_read (void);`	
Description	This function waits for reading started by `R_EEPROM_R` to finish.	
Parameters	None	
Return value	`I2C_OK`:	Data was read successfully.
	`PARA_ERR`:	The address specified by the parameter was out of the EEPROM address range.
	`NO_ACK1`:	No ACK response for the slave address
	`NO_ACK2`:	No ACK response for the EEPROM address
Remark		

R_EEPROM_W

Overview	Writing data to EEPROM
Header	r_cg_macrodriver.h r_cg_userdefine.h
Declaration	void R_EEPROM_W(struct eeprom_paraA16 *PARA);
Description	This function writes data to the address specified by the structure pointed to by the parameter.
Parameters	*PARA Pointer to structure eeprom_paraA16
Return value	None
Remark	Processing to wait for writing to finish is performed by R_EEPROM_wait_write.

R_EEPROM_wait_write

Overview	Waiting for writing to EEPROM to finish
Header	r_cg_macrodriver.h r_cg_userdefine.h
Declaration	MD_STATUS R_EEPROM_wait_write(void);
Description	This function waits for writing started by R_EEPROM_W to finish.
Parameters	None –
Return value	I2C_OK: Data was written successfully. PARA_ERR: The address specified by the parameter was out of the EEPROM address range. NO_ACK1: No ACK response for the slave address NO_ACK2: No ACK response for the EEPROM address NO_ACK3: No ACK response for the transmitted data
Remark	None

check_EEPROM_Addr

Overview	Checking the EEPROM access area (block number)
Header	r_cg_macrodriver.h r_cg_userdefine.h
Declaration	static MD_STATUS check_EEPROM_Addr(struct eeprom_paraA16 *PARA);
Description	This function checks whether the area to be read or written specified by the EEPROM access parameter is within the EEPROM.
Parameters	*PARA Pointer to structure eeprom_paraA16
Return value	I2C_OK: The area to be accessed is within the EEPROM size. PARA_ERR: The area to be accessed is not within the EEPROM size.
Remark	None

get_slave_Addr

Overview	Calculating the EEPROM slave address
Header	r_cg_userdefine.h
Declaration	static void get_slave_Addr(void);
Description	This function modifies the slave address by using the upper 3 bits of the cell address when using a 4 Kb to 16 Kb EEPROM.
Parameters	None –
Return value	None
Remark	This function modifies slaveaddr, a member of the g_PARAC structure, by using the value of another member eepromaddr.

`R_IIC0_Tx_addr1`

Overview	Transmitting the EEPROM address
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_Tx_addr1(void);`
Description	This function transmits the upper bytes of the cell address when using an EEPROM of 32 Kb or more. When using an EEPROM of 16 Kb or less, this function transmits a one-byte cell address.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC0_Tx_addr2`

Overview	Transmitting the lower address of the EEPROM cell
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_Tx_addr2(void);`
Description	This function transmits the lower bytes of the cell address when using an EEPROM of 32 Kb or more.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC0_Rx_RST`

Overview	Restarting reception to read data from EEPROM
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_Rx_RST(void);`
Description	This function restarts transfer in reception mode in order to read data.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC0_RxData_ST`

Overview	Starting data reception
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_RxData_ST(void);`
Description	This function starts receiving data (by writing dummy data to the IIC0 register) once the slave address has been transmitted in reception mode.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC0_RxData`

Overview	Data reception
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_RxData (void);`
Description	This function stores the received data in the buffer and starts the next data reception.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC0_Rx_LastData`

Overview	Finishing reception of final data
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_Rx_LastData (void);`
Description	This function stores the data received last, generates the stop condition, and then ends reception processing.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC0_TxDataST`

Overview	Starting data transmission
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_TxDataST (void);`
Description	This function starts data transmission once the EEPROM address has been transmitted.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC0_TxData`

Overview	Data transmission
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC0_TxData (void);`
Description	Once 1-byte data has been transmitted, this function transmits the next data. When all data units to be written have been transmitted, this function generates the stop condition and instructs the EEPROM to write data. If there is data left to write, this function transmits the data.
Parameters	None –
Return value	None
Remark	Used during MD_INTIIC0 interrupt servicing

`R_IIC00_StartCondition`

Overview	Generating a start condition
Header	`r_cg_userdefine.h`
Declaration	`static void R_IIC00_StartCondition(void);`
Description	This function generates a start condition and waits for the start condition to be detected.
Parameters	None –
Return value	None
Remark	None

`R_IIC00_StopCondition`

Overview	Generating a stop condition
Header	`r_cg_macrodriver.h` `r_cg_userdefine.h`
Declaration	`Void R_IIC00_StopCondition(void)`
Description	This function generates a stop condition and waits for the stop condition to be detected.
Parameters	None
Return value	None
Remark	None

`MD_INTIIC0`

Overview	IIC0 transfer complete interrupt
Header	`r_cg_userdefine.h`
Declaration	`__interrupt void MD_INTIIC0(void);`
Description	Triggered by the INTIIC0 interrupt request, this function executes the appropriate processing according to the communication status. When a NACK response from the slave is detected in a state other than finishing read from the EEPROM, this function sets the error flag of the corresponding `g_comstatus` bit.
Parameters	None —
Return value	None
Remark	

5.7 Flowcharts

Figure 5.3 shows an overview of the processing flow used in this application note.

Remark Processing of functions other than those for IIC00 is omitted in the flowchart of the initialization function.

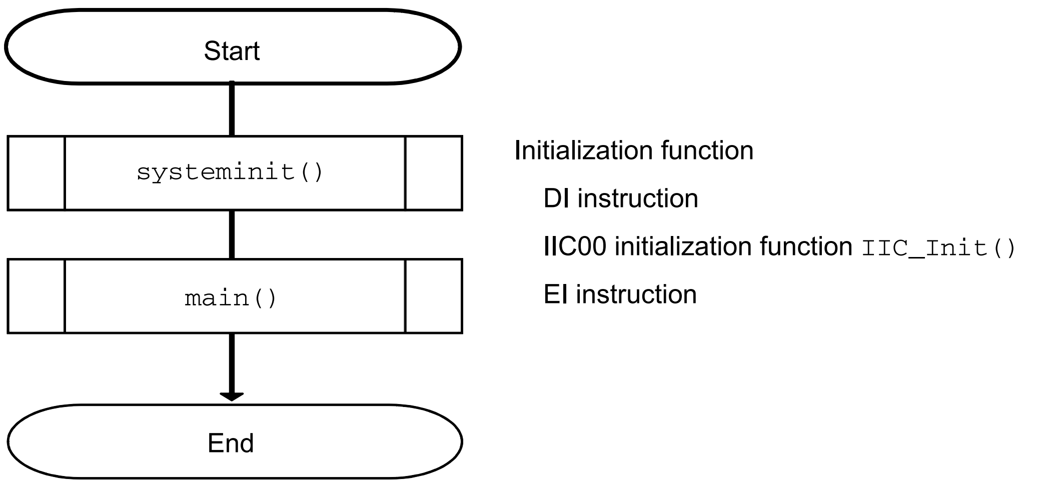


Figure 5.3 Overview of processing flow

5.7.1 IIC00 initialization function

Figure 5.4 shows the flow of the initialization function processing.

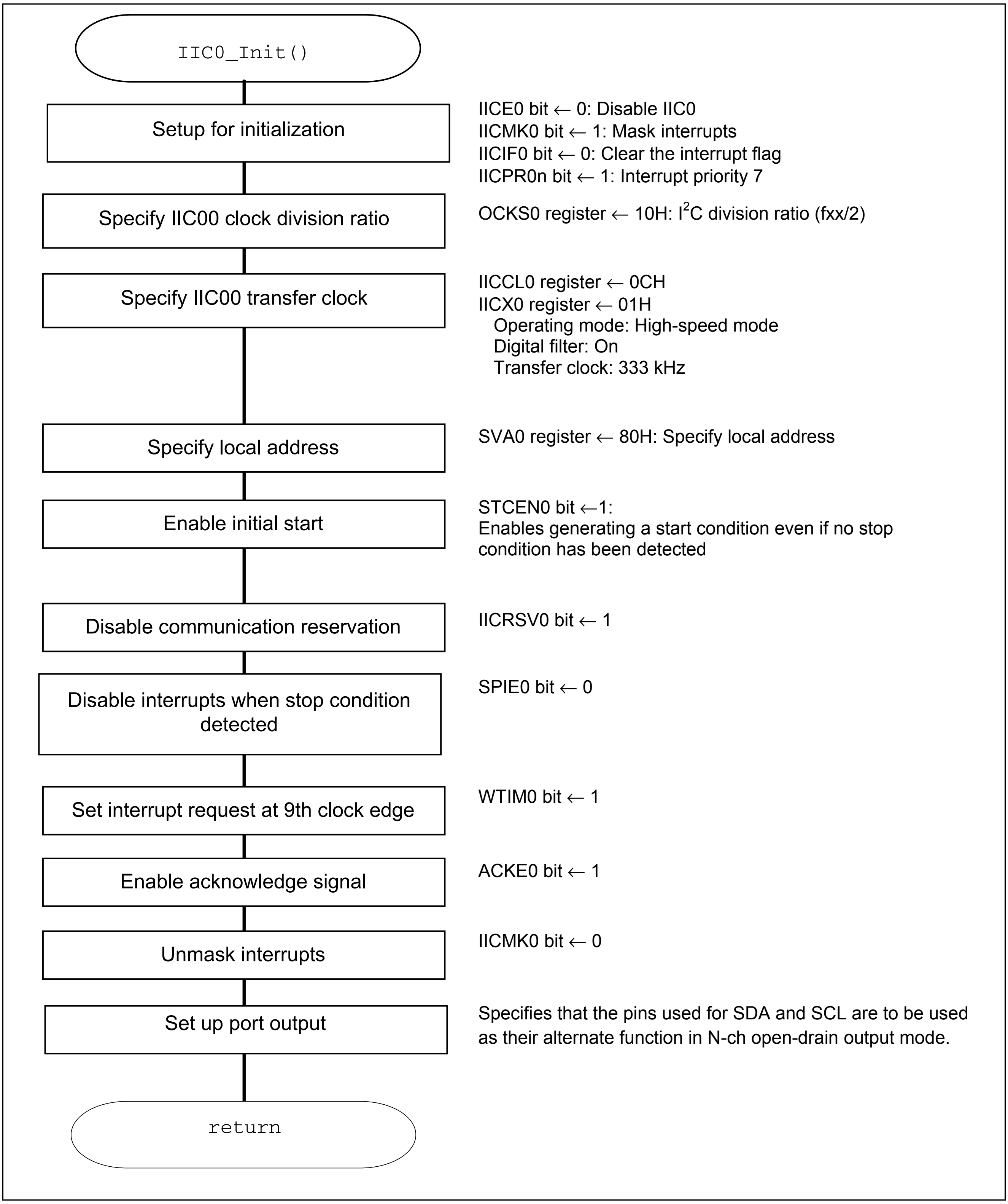


Figure 5.4 IIC0 initialization function

5.7.2 main function

Figures 5.5 and 5.6 show the flow of the main processing. The main function tests reading from and writing to a 16 Kb EEPROM.

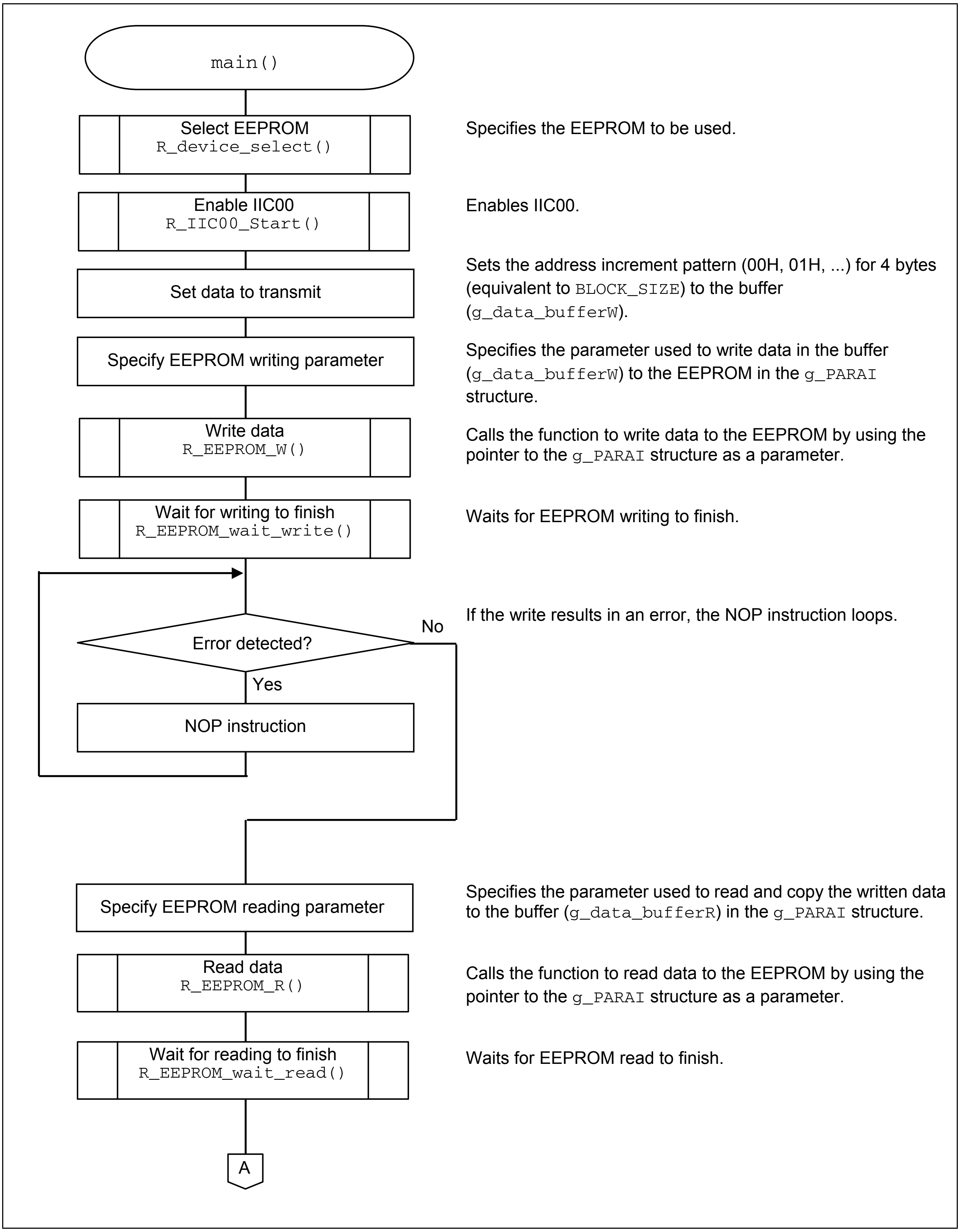
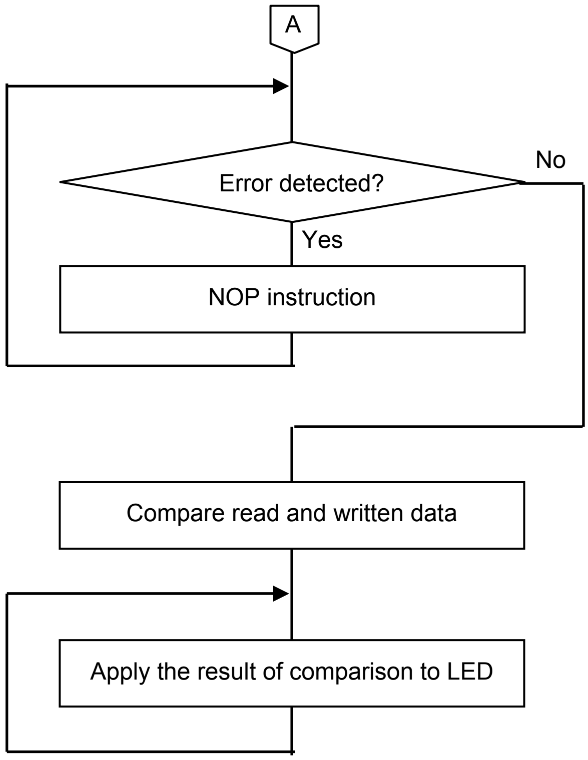


Figure 5.5 main processing (1/2)



If the read results in an error, the NOP instruction loops.

Successively compares data to judge whether the data in the write buffer (`g_data_bufferW`) and read buffer (`g_data_bufferR`) match. If the data does not match, an error flag (`COMP_ERR`) is set.

Turns on LED1 if the data matches, and LED2 if the data does not match.

Figure 5.6 `main` function (2/2)

5.7.3 Selection of EEPROM

Figure 5.7 shows the flow of EEPROM selection.

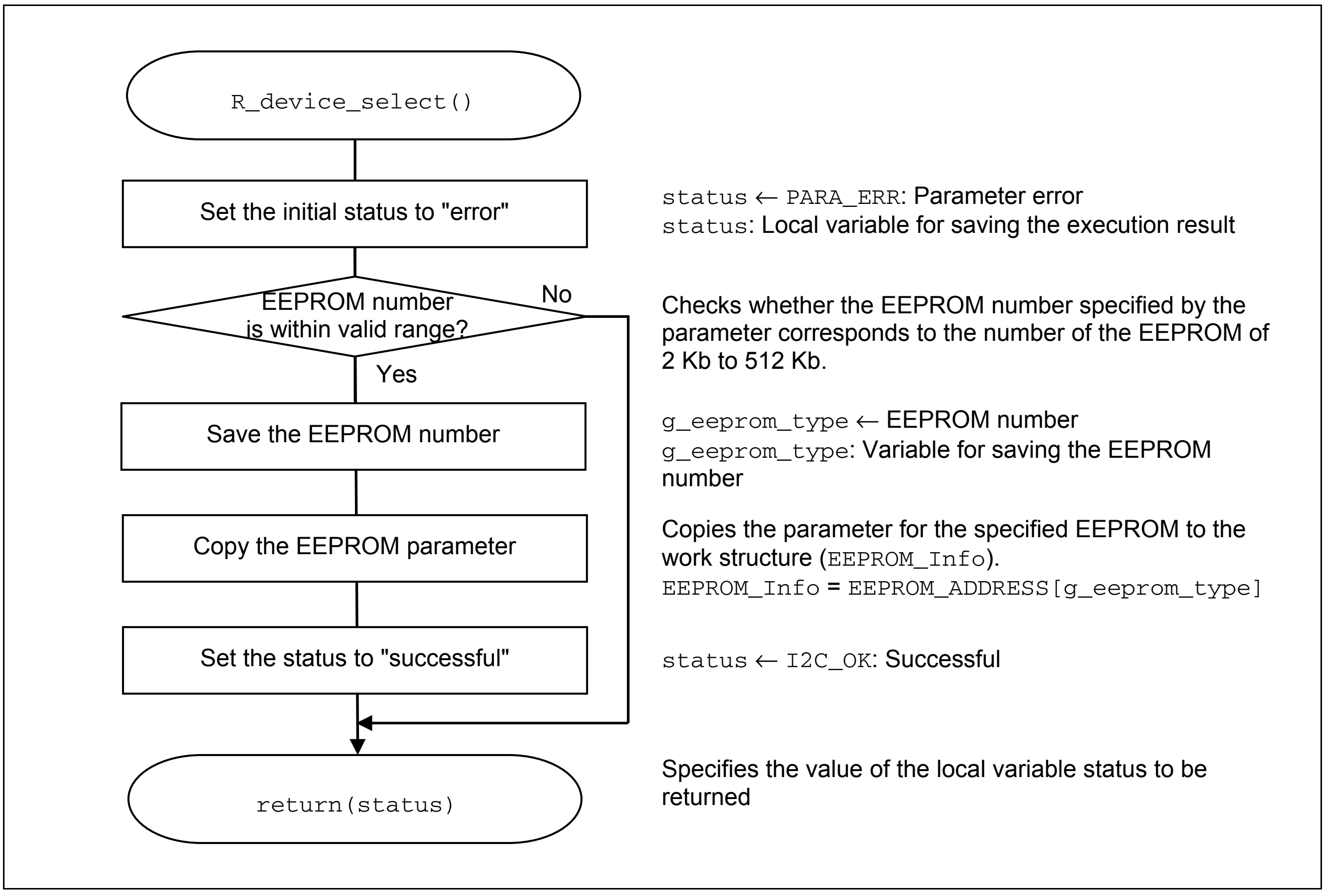


Figure 5.7 Selection of EEPROM

5.7.4 IIC0 start function

Figure 5.8 shows the flow of the IIC0 start function processing.

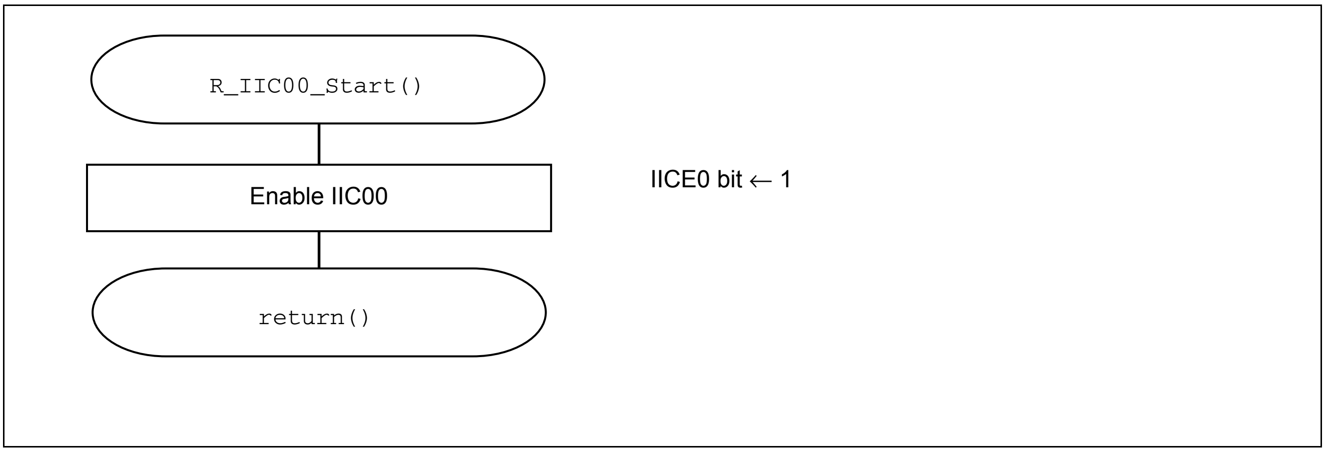


Figure 5.8 IIC0 start function

5.7.5 Start condition generation function

Figure 5.9 shows the flow of the start condition generation function.

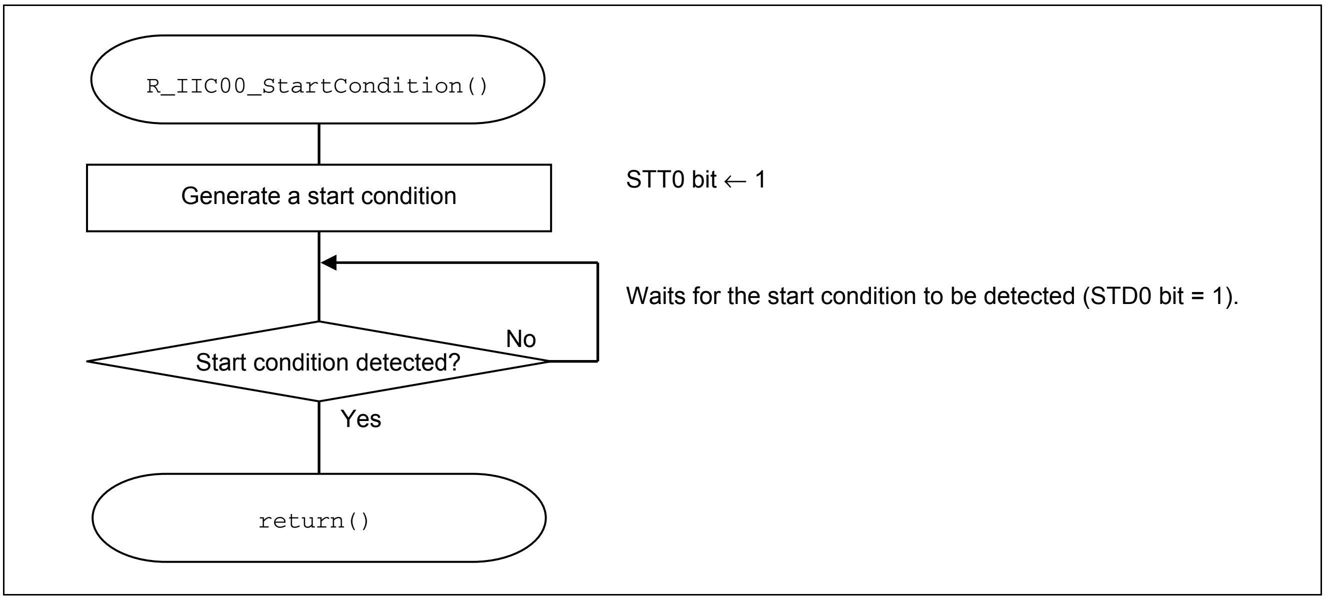


Figure 5.9 Start condition generation function

5.7.6 Stop condition generation function

Figure 5.10 shows the flow of the stop condition generation function.

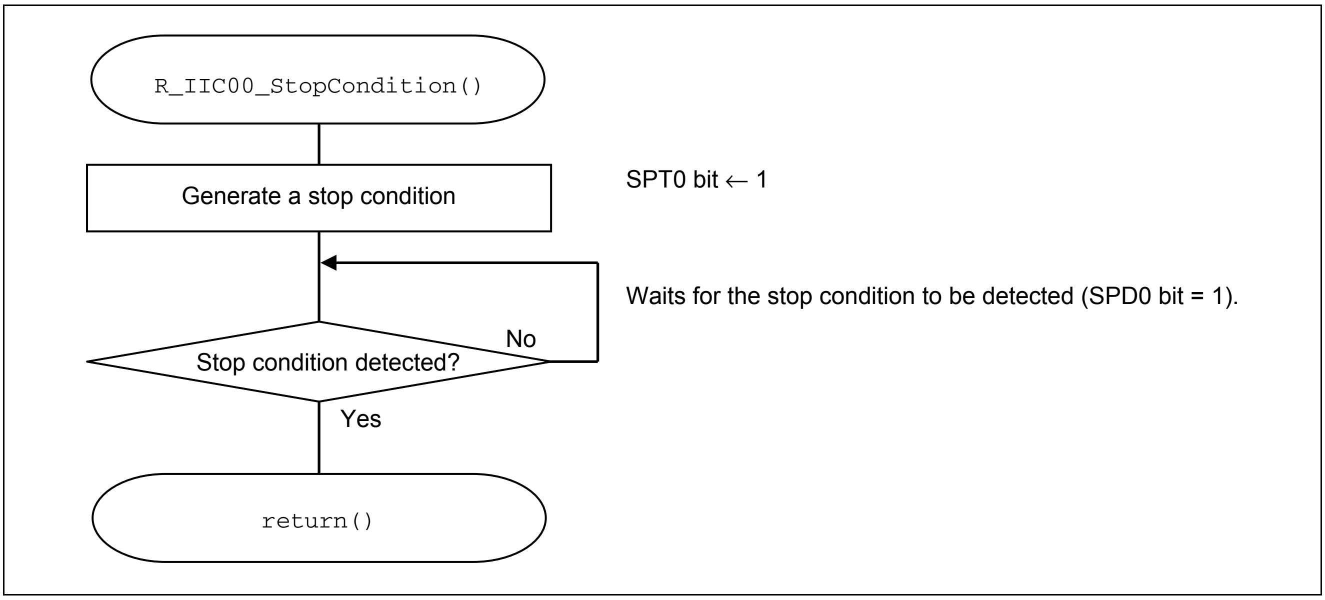


Figure 5.10 Stop condition generation function

5.7.7 EEPROM write processing

Figure 5.11 shows the flow of EEPROM write processing.

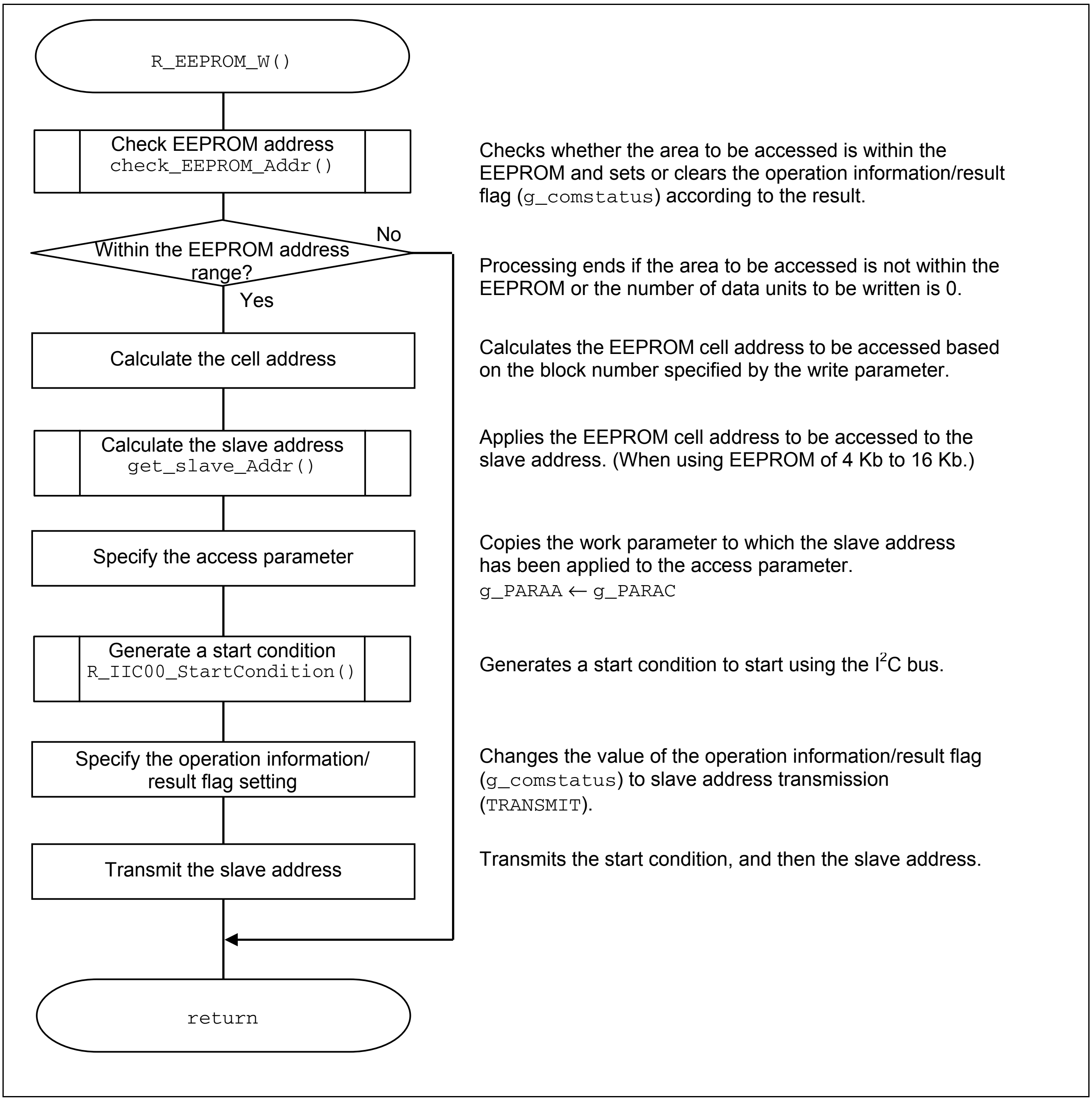
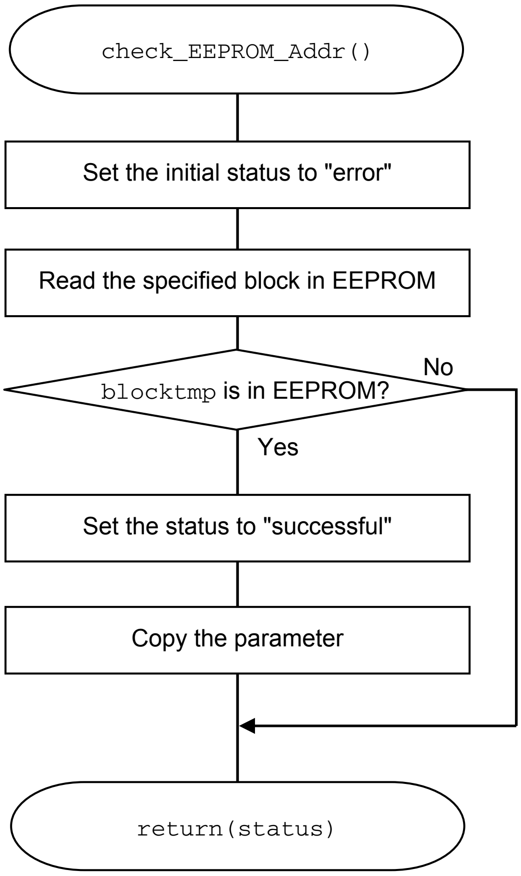


Figure 5.11 EEPROM write processing

5.7.8 EEPROM address check processing

Figure 5.12 shows the flow of EEPROM address check processing.



Sets the initial status to "error".
status ← PARA_ERR
(status: Local variable)

Sets the EEPROM block to be accessed specified by the g_PARAI structure for which the pointer has been passed by using the parameter, to the work variable.
blocktmp ← PARA -> block_num

Processing results in an error and ends if the access start block number (address) is not within the EEPROM address range.

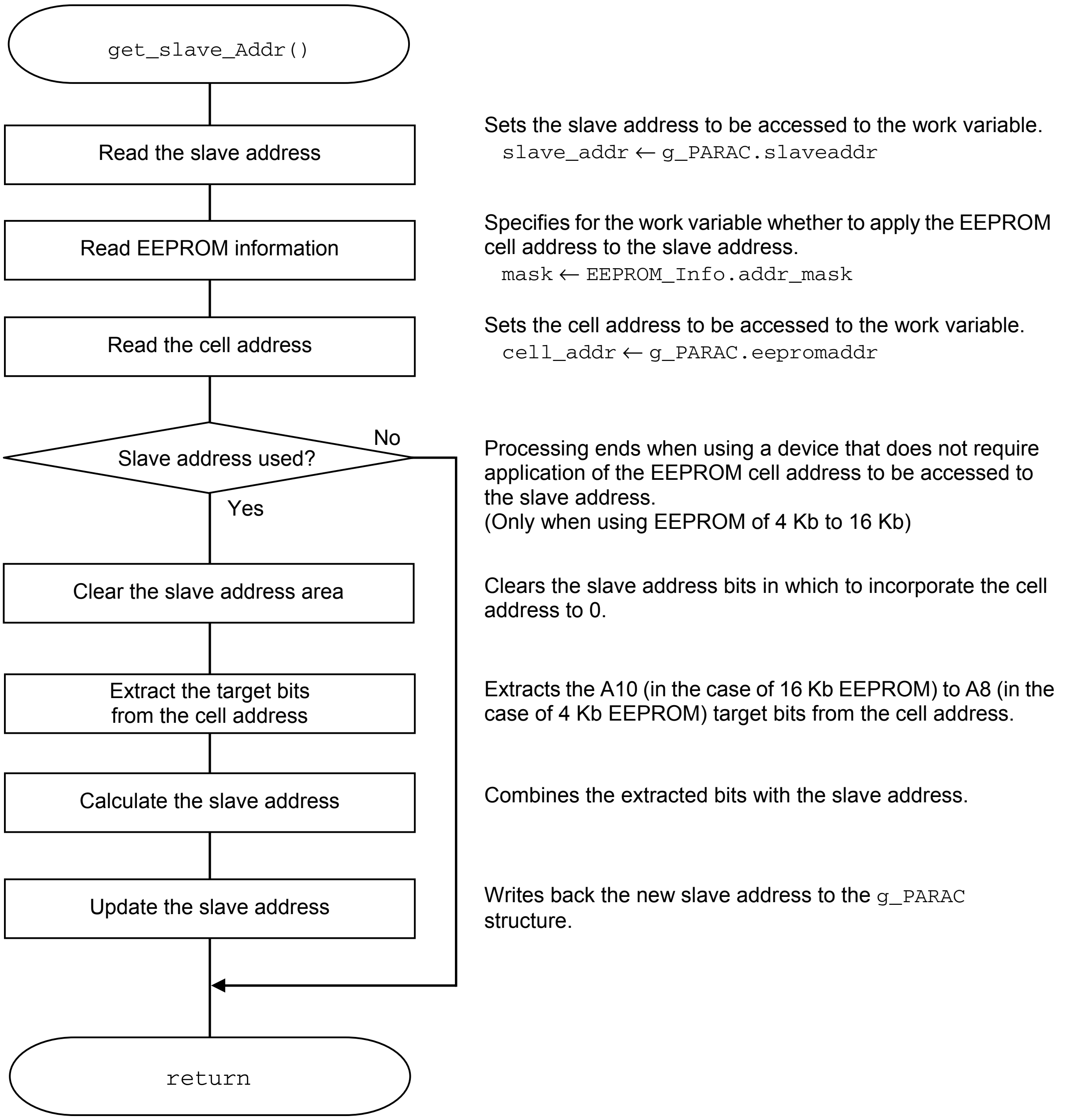
status ← I2C_OK: Successful

Copies the parameter of the structure for which the pointer has been passed by using the parameter to the g_PARAC work structure.
g_PARAC ← *PARA

Figure 5.12 EEPROM address check processing

5.7.9 Slave address calculation

Figure 5.13 shows the flow of slave address calculation.



Remark This processing is required when using a 4 Kb EEPROM (R1EX24004A), 8 Kb EEPROM (R1EX24008A), or 16 Kb EEPROM (R1EX24016A). This processing is skipped when using a 2 Kb EEPROM (R1EX24002A) or 32 Kb EEPROM (R1EX24032A) to 512 Kb EEPROM (R1EX24512B).

Figure 5.13 Slave address calculation

5.7.10 Waiting for writing to EEPROM to finish

Figure 5.14 shows the flow of waiting for writing to EEPROM to finish.

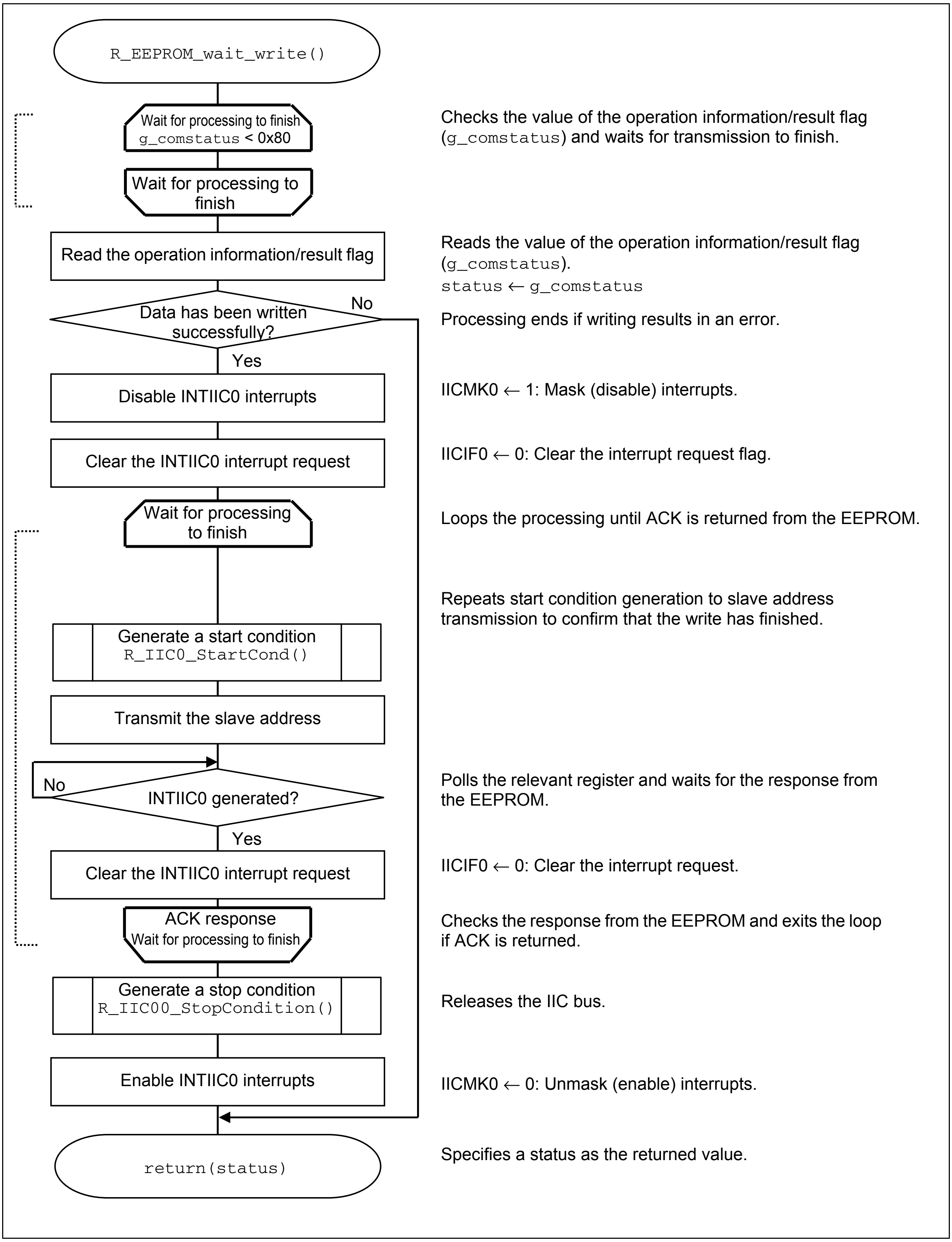


Figure 5.14 Waiting for writing to EEPROM to finish

5.7.11 EEPROM read processing

Figure 5.15 shows the flow of EEPROM read processing.

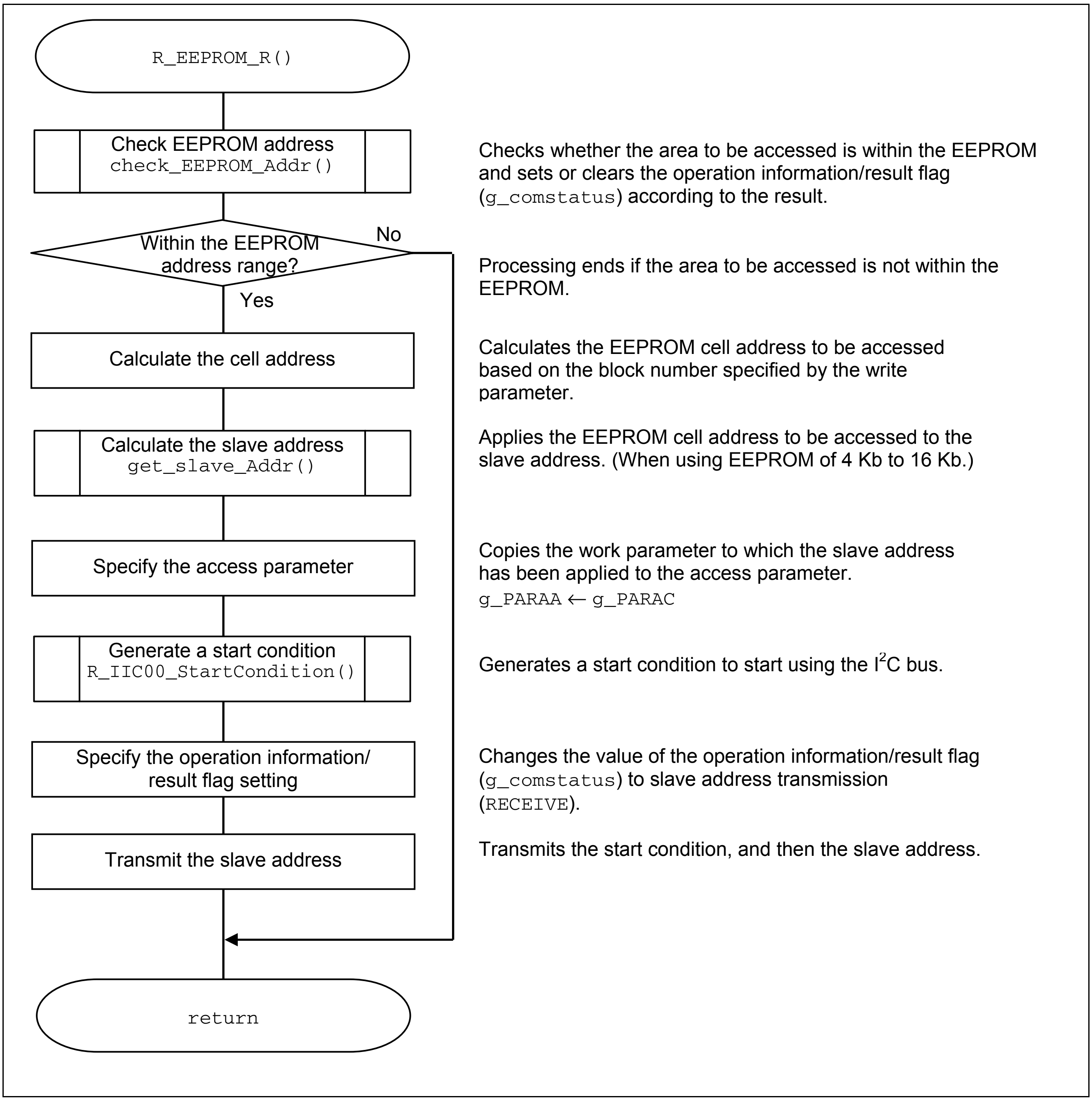


Figure 5.15 EEPROM read processing

5.7.12 Waiting for reading from EEPROM to finish

Figure 5.16 shows the flow of waiting for reading from EEPROM to finish.

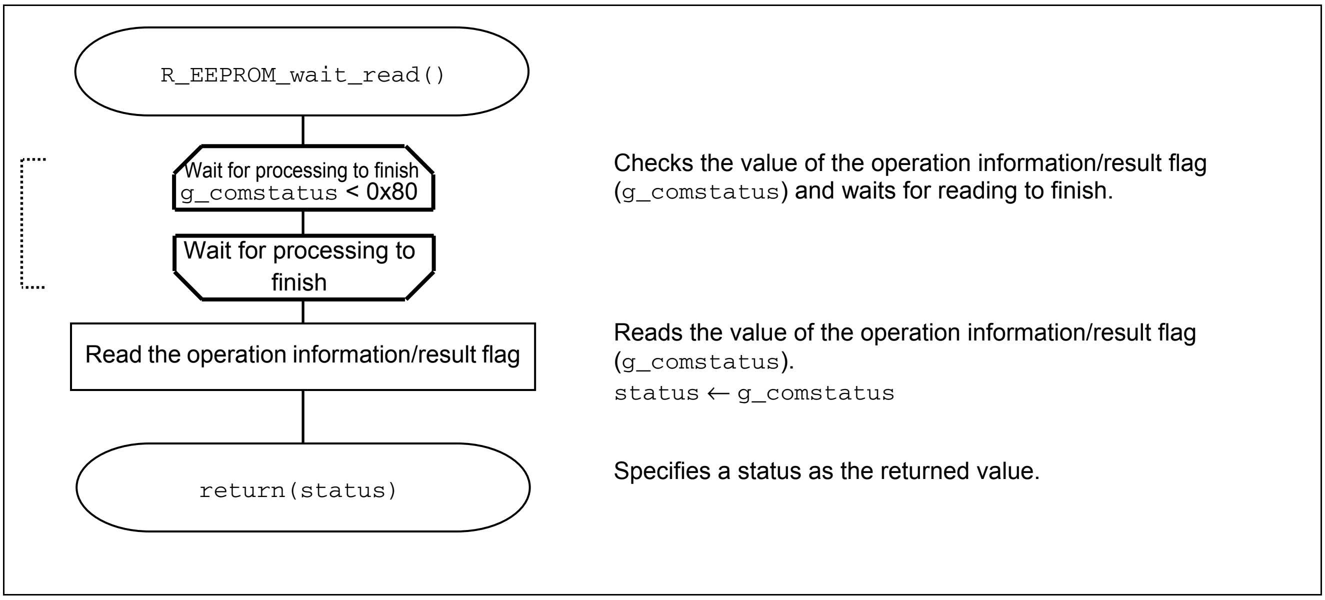


Figure 5.16 Wait for reading to finish

5.7.13 Finishing slave address transmission

Figure 5.17 shows the flow of finishing slave address transmission.

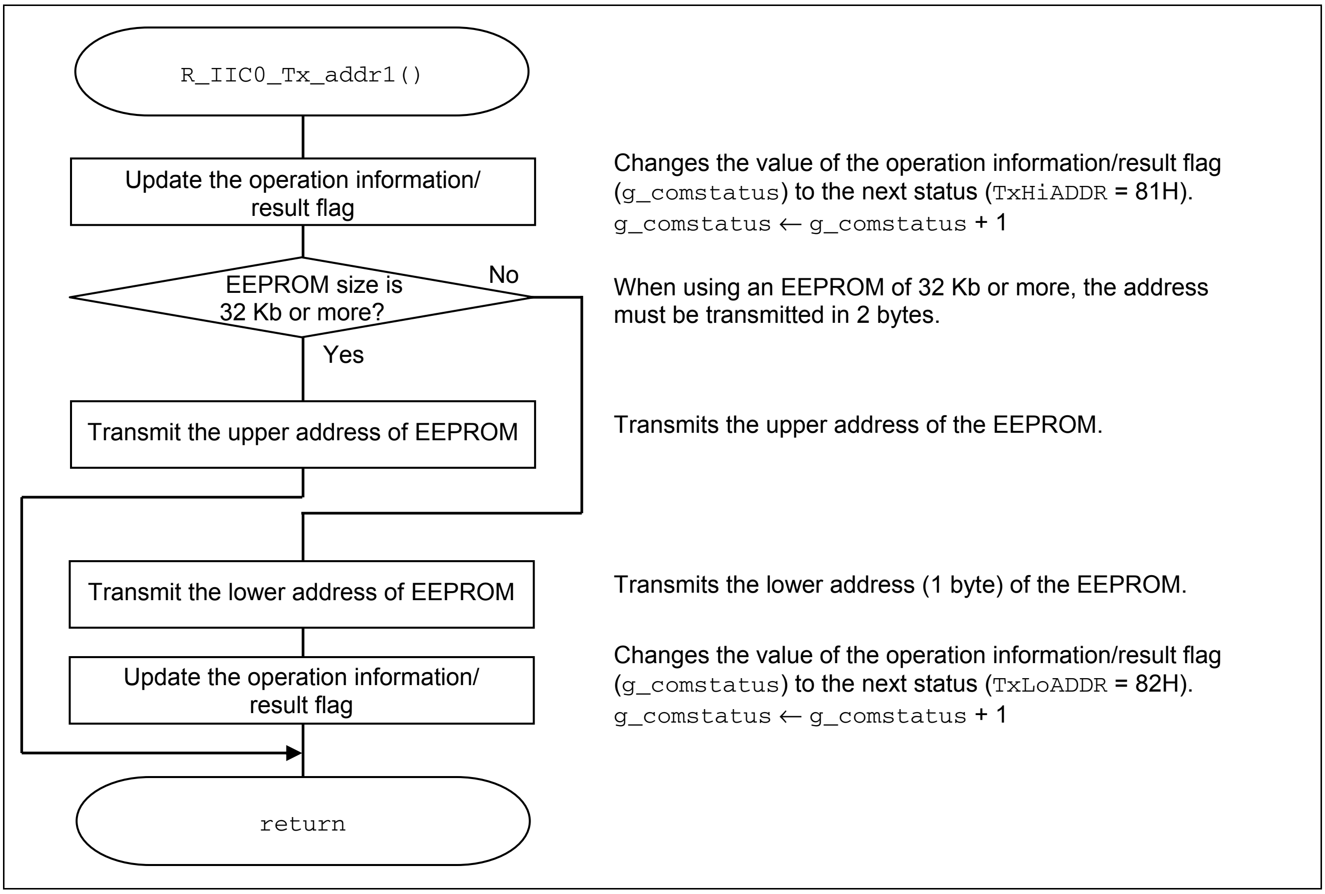


Figure 5.17 Finishing slave address transmission

5.7.14 Finishing upper address transmission

Figure 5.18 shows the flow of finishing upper address transmission.

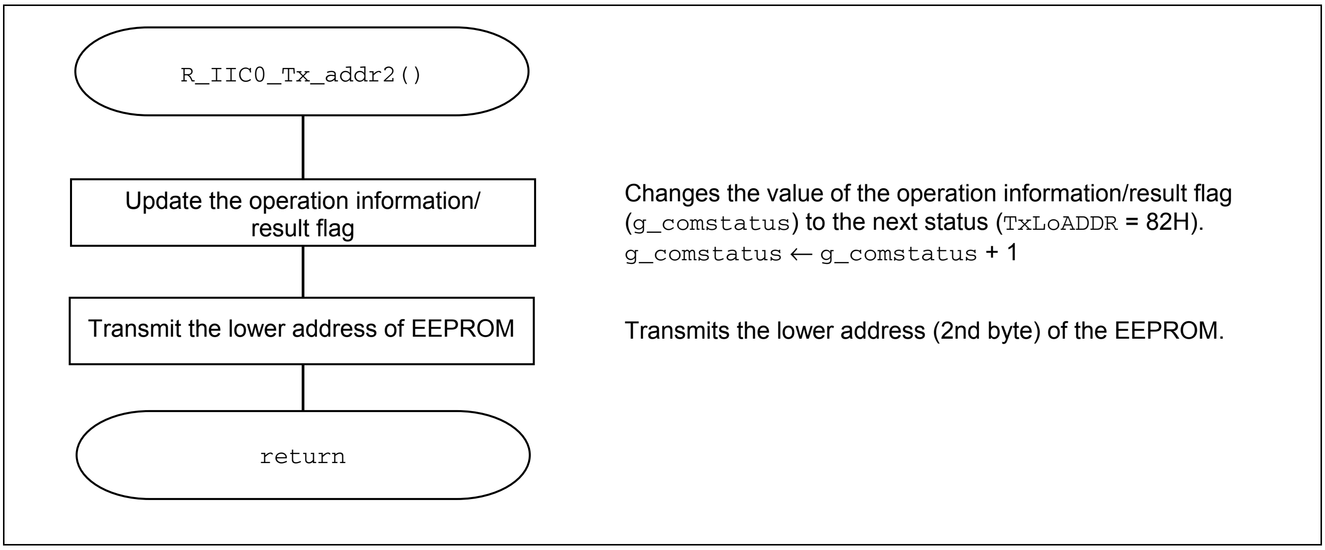


Figure 5.18 Finishing upper address transmission

5.7.15 Restart processing

Figure 5.19 shows the flow of restart processing.

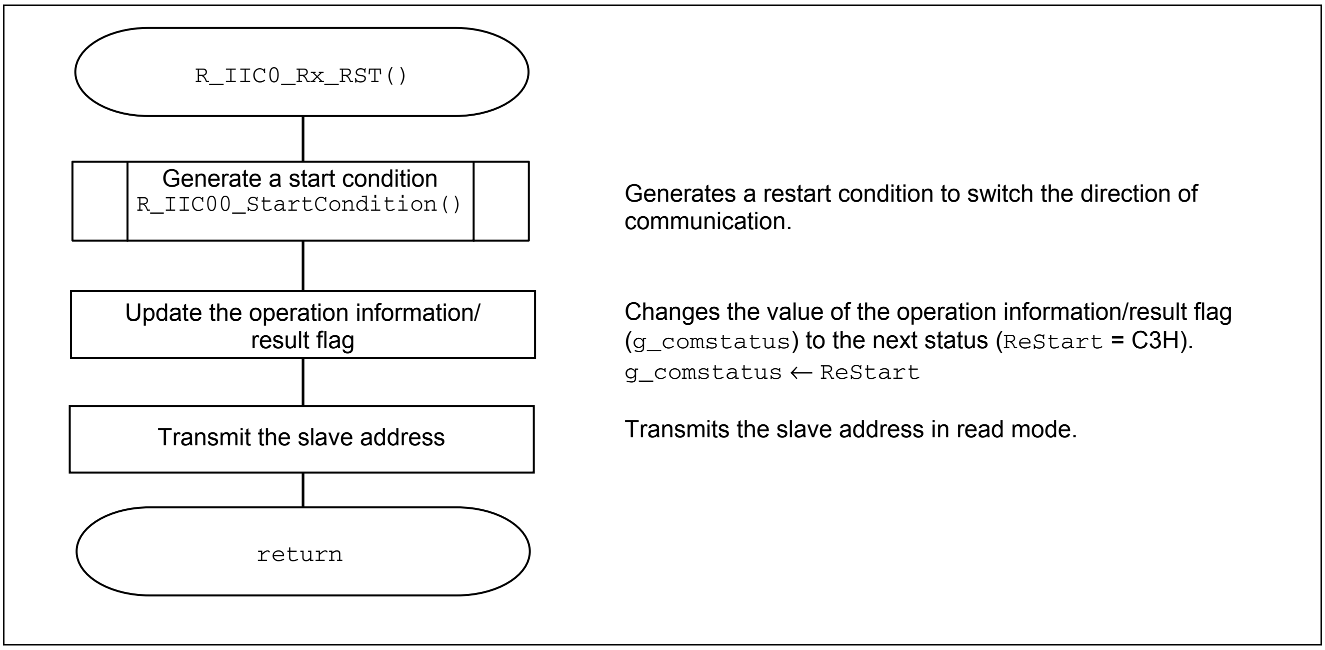


Figure 5.19 Restart processing

5.7.16 Starting data reception

Figure 5.20 shows the flow of data reception start processing.

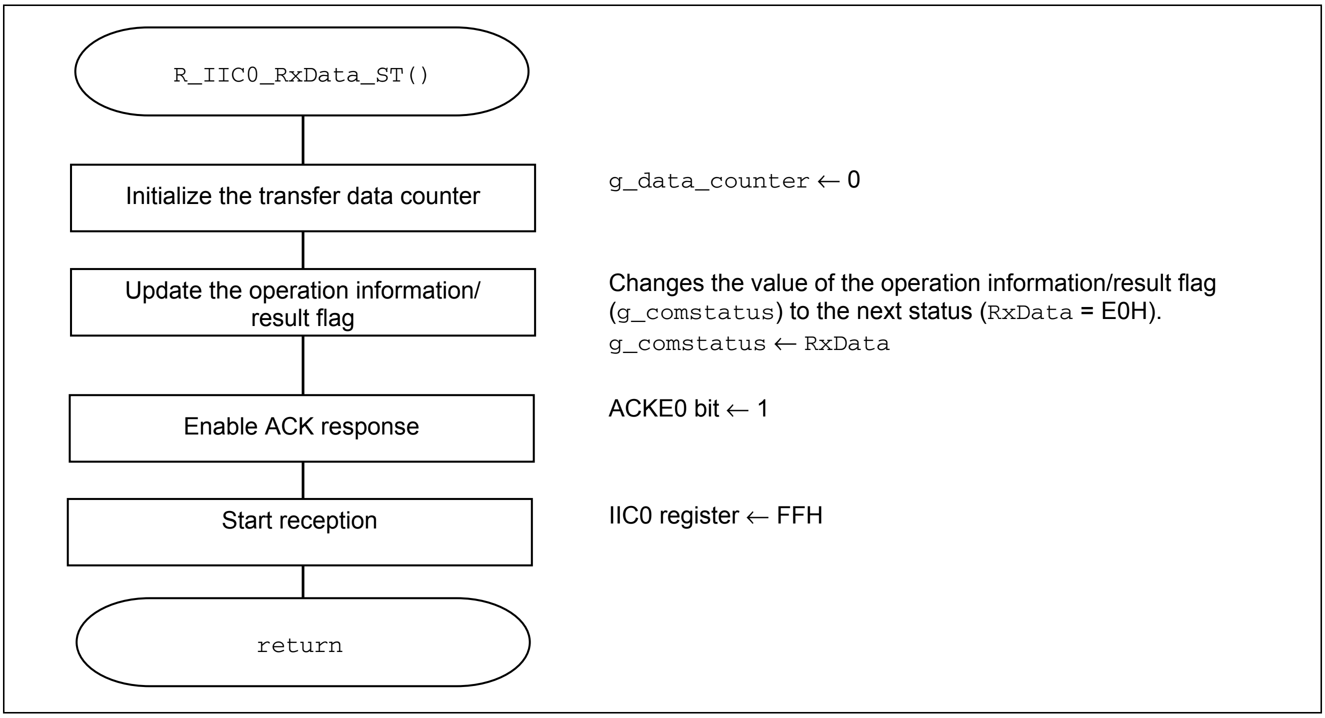


Figure 5.20 Starting data reception

5.7.17 Data reception

Figure 5.21 shows the flow of data reception.

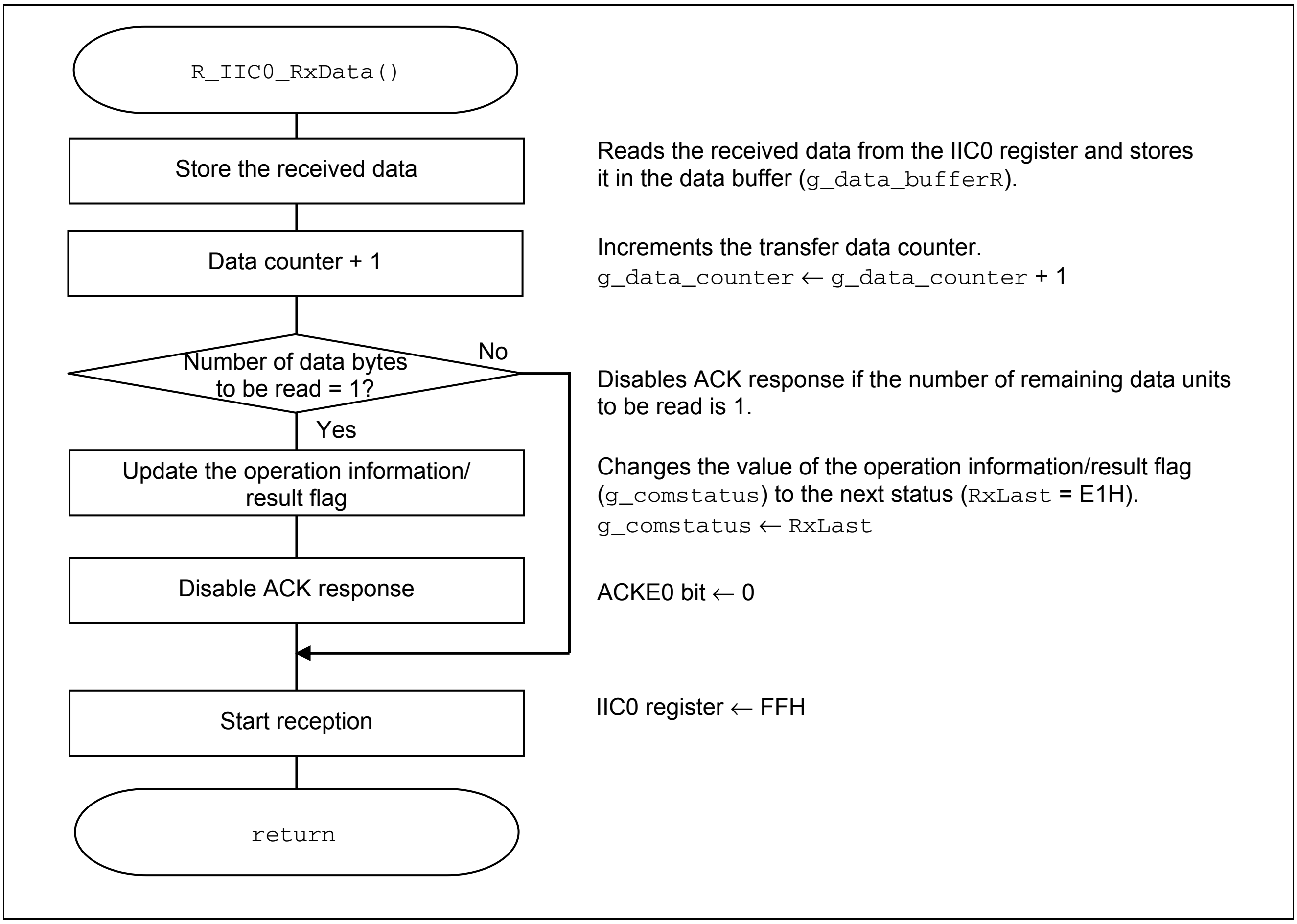


Figure 5.21 Data reception

5.7.18 Reception of final data

Figure 5.22 shows the flow of reception of final data.

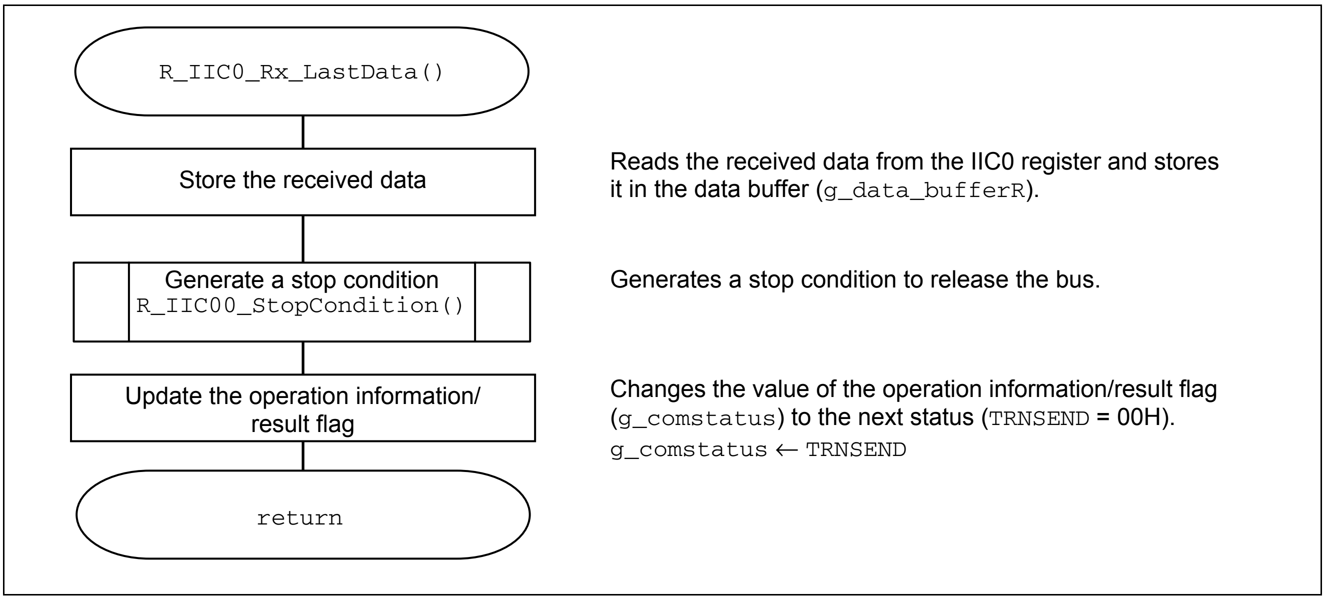


Figure 5.22 Reception of final data

5.7.19 Starting data transmission

Figure 5.23 shows the flow of data transmission start processing.

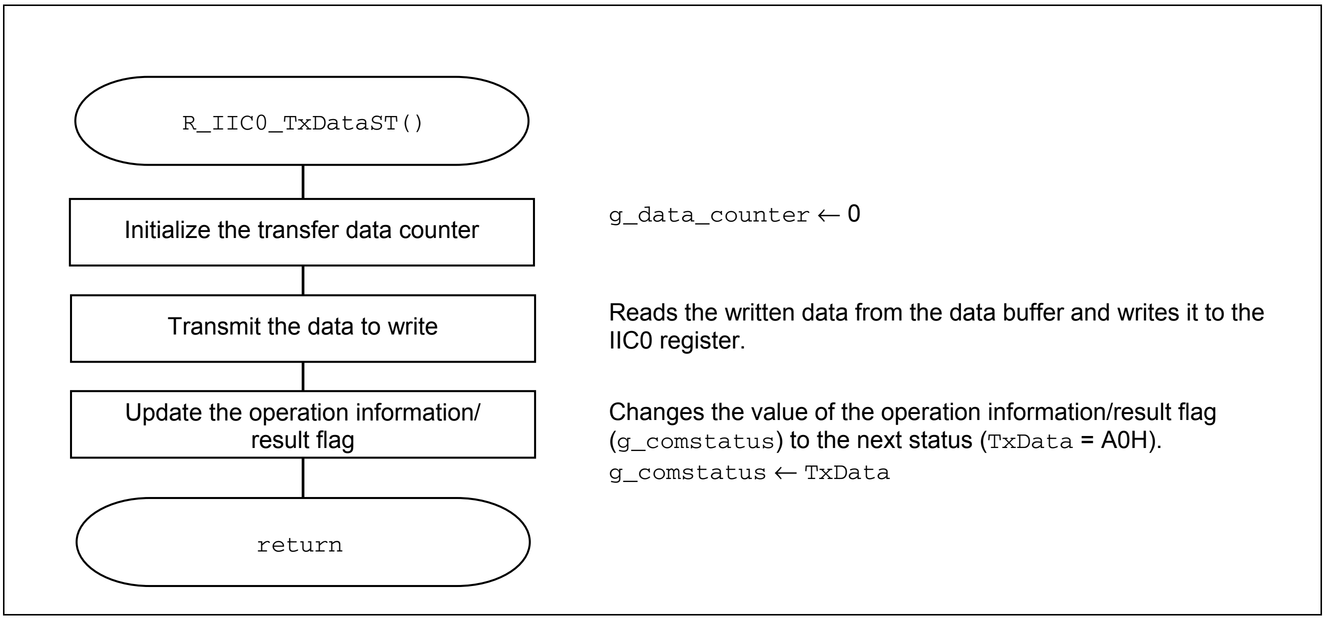
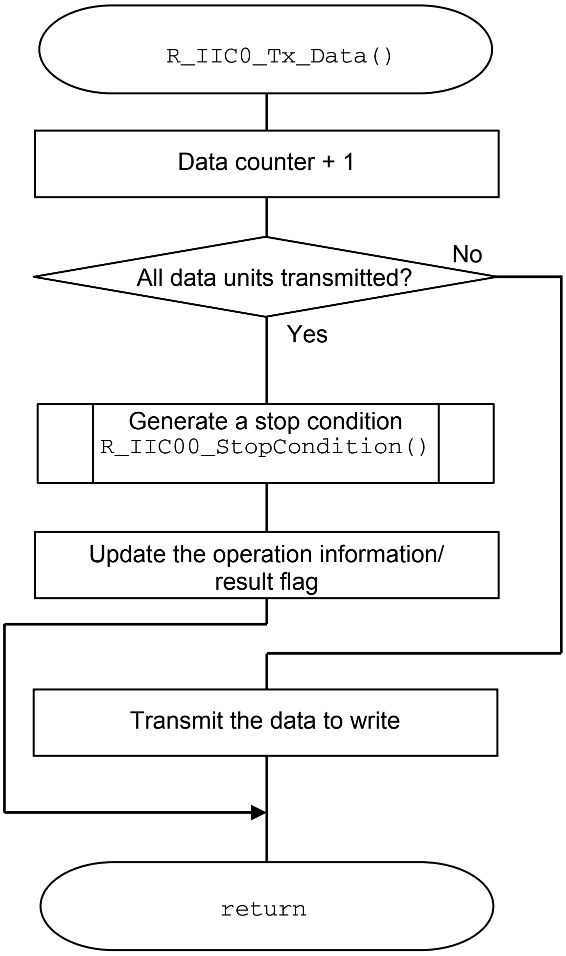


Figure 5.23 Starting data transmission

5.7.20 Data transmission

Figure 5.24 shows the flow of data transmission.



Increments the transfer data counter.
g_data_counter ← g_data_counter + 1

Starts writing to the EEPROM if all the data units have been transmitted (g_data_counter = BLOCK_SIZE), or transmits the next data if the number of remaining data units to write is not 0.

Generates a stop condition to instruct the ERPROM to start writing.

Changes the value of the operation information/result flag (g_comstatus) to the next status (TRNSEND = 00H).
g_comstatus ← TRNSEND

Reads the written data from the data buffer and writes it to the IIC0 register.

Figure 5.24 Data transmission

5.7.21 MD_INTIIC0 interrupt servicing

Figures 5.25 to 5.27 show the flow of INTIIC0 interrupt servicing.

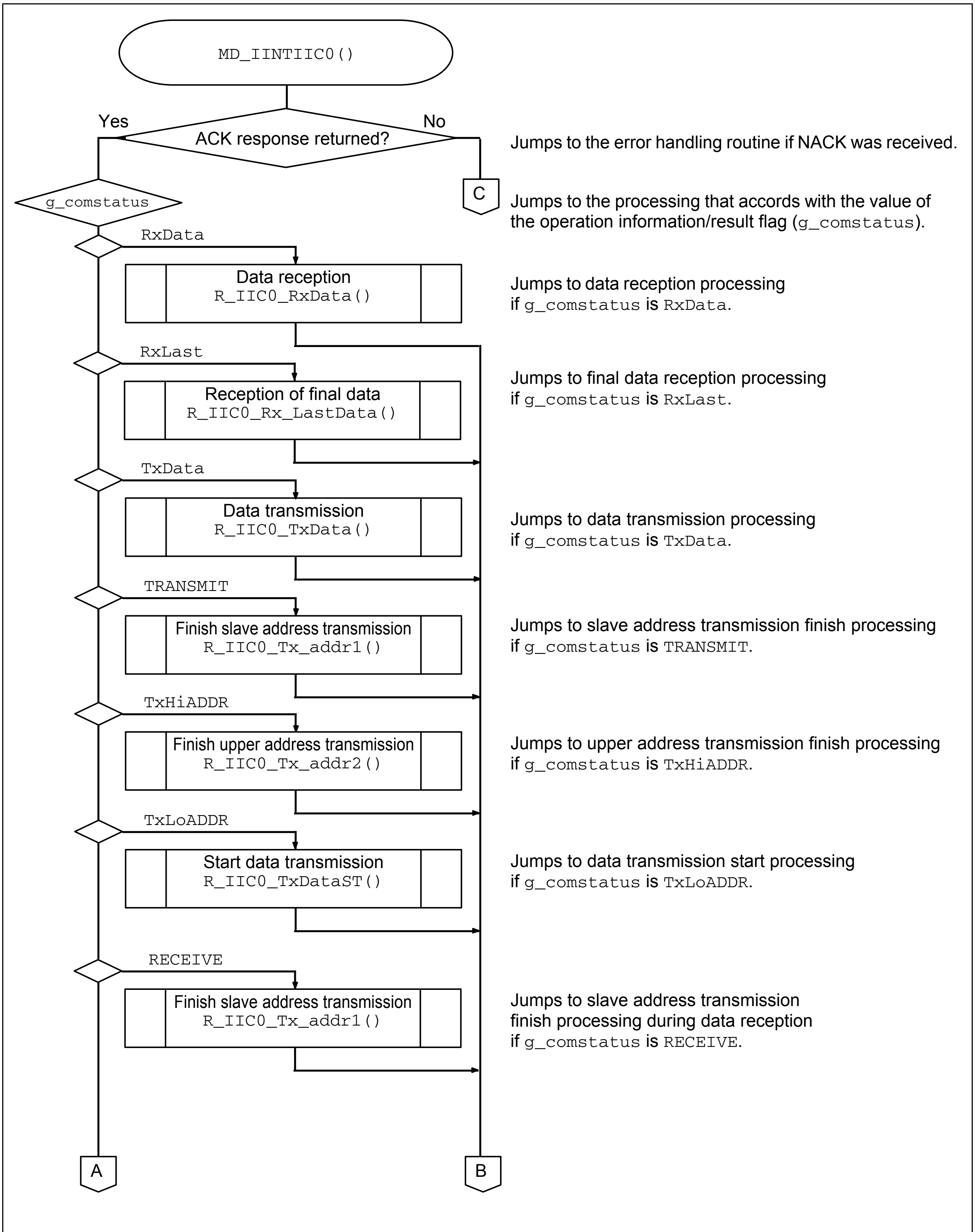


Figure 5.25 MD_INTIIC0 interrupt servicing (1/3)

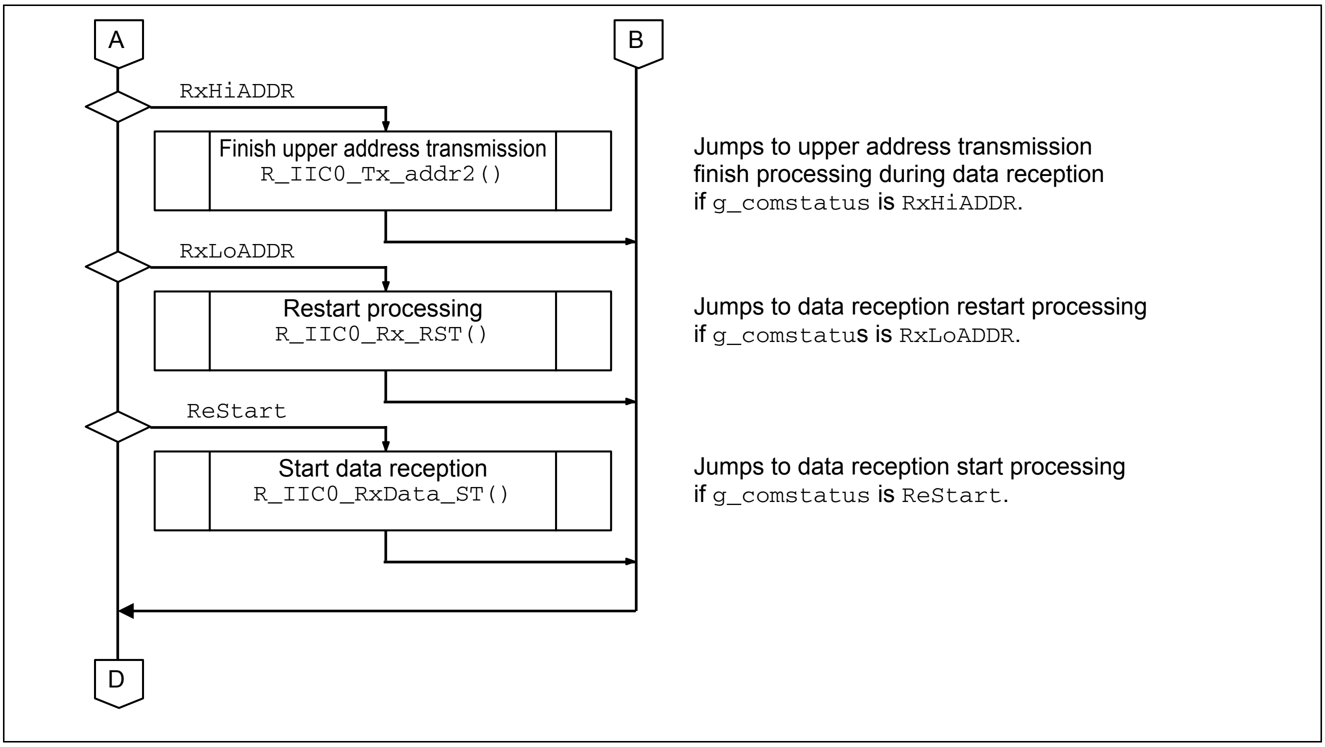


Figure 5.26 MD_INTIIC0 interrupt servicing (2/3)

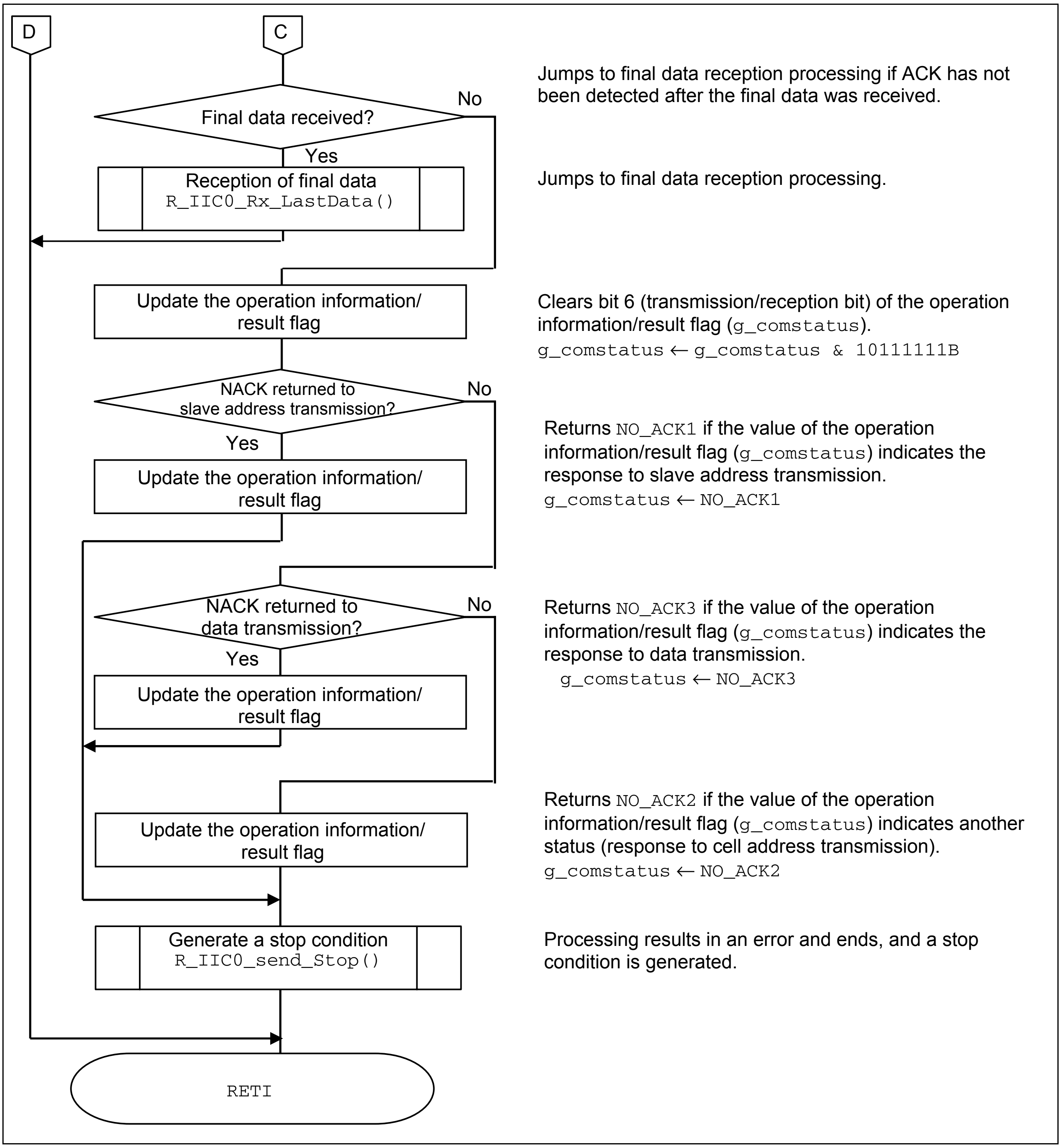


Figure 5.27 MD_INTIIC0 interrupt servicing (3/3)

6. Sample Code

Obtain the sample code from the Renesas Electronics website.

7. Reference Documents

V850ES/JC3-L, JE3-L User's Manual: Hardware (R01UH0018E)
V850ES/JF3-L User's Manual: Hardware (R01UH0017E)
V850ES/JG3-L User's Manual: Hardware (R01UH0165E)
V850ES/JG3-L On-chip USB Controller Hardware User's Manual (R01UH0001E)
(Obtain the latest versions from the Renesas Electronics website.)

Technical updates and technical news
(Obtain the latest versions from the Renesas Electronics website.)

Website and Support

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

Inquiries
http://www.renesas.com/contact

Revision Record	V850ES/Jx3-L I^2C Bus EEPROM Control

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 28, 2013	—	First edition issued

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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord 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 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 one with a different part number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 2.2