RX Family, RL78 Family

R01AN1075EJ0103
Rev.1.03
Mar 31, 2016

Renesas R1EX24xxx Series Serial EEPROM Control Software

Introduction

This application note explains the method of controlling R1EV24xxx, R1EX24xxx, and HN58X24xxx series I^2C serial EEPROM, manufactured by Renesas Electronics, by using a Renesas Electronics MCU, and also describes the usage of the supplied sample code.

In the sample code, an upper layer of software controls the slave devices and a lower layer of software implements I^2C single master basic protocol control. The protocols provided by the upper and lower layers are used in combination to control the slave devices.

The sample code comprises the upper layer of software for controlling the serial EEPROM as slave devices.

Lower layer software for implementing I^2C single master control in the MCU as master device is available separately in versions for specific MCU models, so please obtain this from the following URL as well. In addition, when a new microcontroller is added to the clock synchronous single-master control software, update of this application note may not be in time. Refer to 'I^2C Single Master Control Software (Lower-level layer of the software)' information in the following URL for the combination information on the latest supported microcontroller and its single-master control software.

- I^2C Serial EEPROM Driver
  http://www.renesas.com/driver/i2c_serial_eeprom

Target Device

Serial EEPROM

Renesas Electronics R1EV24xxx, R1EX24xxx, or HN58X24xxx series I^2C serial EEPROM

MCU on which operation has been verified

RL78/G1x series : RL78/G14, RL78/G1C group (using IICA serial interface)

RL78/L1x series : RL78/L12, RL78/L13, RL78/L1C group (using IICA serial interface)

RX600 series : RX62N, RX63N, RX63T group (using RIIC)

RX200 series : RX210, RX21A group (using RIIC)

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Contents

1. Specifications

The method of controlling R1EV24xxx, R1EX24xxx, and HN58X24xxx series I^{2}C serial EEPROM, manufactured by Renesas Electronics, by using a Renesas Electronics MCU is described below.

Communication with the I^{2}C serial EEPROM can be accomplished by using the software sample code provided with this application note in combination with lower layer I^{2}C single master control software. The software sample code is for communication control, and the processing of reading and writing data, etc., is handled by the lower layer software.

Table 1.1 lists the peripheral device used and its application, and Figure 1.1 shows a usage example.

The following is an overview of the functionality of the control software.

- The control software is a block device driver for a system in which a Renesas Electronics MCU functions as the master device and Renesas Electronics R1EV24xxx, R1EX24xxx, or HN58X24xxx series I^{2}C serial EEPROM functions as slave devices.
- I^{2}C serial EEPROM write operation, acknowledge polling (EEPROM rewrite state confirmation), and read operation are supported.
- The I^{2}C single master control software's master transmit mode (pattern 1)*[1] is used for EEPROM write operation. The bus is released after transmission completes (after a stop condition occurs). Acknowledge polling is used to determine EEPROM data rewrite completion.
- The I^{2}C single master control software's master transmit mode (pattern 3)*[1] is used for acknowledge polling. Completion of a rewrite to the EEPROM is determined by receiving an ACK or NACK response when the slave address is transmitted.
- The I^{2}C single master control software's master composite mode (master transmit → master receive)*[1] is used for EEPROM read operation.
- Communication is implemented by means of a start function, which starts each control operation, and an advance function, which monitors the communication and causes protocol processing to proceed. After communication starts it is necessary to call the advance function to complete the communication.
- Multiple channels are supported.
- Multiple slave devices with different type name can be controlled on a single channel bus.*2 However, while communication is in progress (the period from when the start condition occurs to when the stop condition occurs), communication with other devices is not possible.
- The methods of specifying the *address that designates the slave device* and the *EEPROM internal address* differ depending on the capacity of the I^{2}C serial EEPROM used. The sample code performs address conversion by setting the necessary information.
- The write protect (WP = H) state control function is not supported. Communication starts even if the target device is in the write protect state, but an error is returned as a NACK response from the EEPROM when the data is transmitted.
- If, when communication is in progress, communication stops for more than a certain amount of time or a bus hang-up occurs due to noise, etc., the control software can perform EEPROM recovery processing to release the bus.

Notes: 1. The details depend on the specifications of the I^{2}C single master control software version used.
2. The number of serial EEPROM devices that can be connected as slave devices differs depending on the capacity. Make sure to keep this in mind when connecting multiple slave devices to a single channel. Table 5.2 lists the number of slave devices that can be connected by capacity.

Table 1.1 Peripheral Function and Its Application

Peripheral Function	Application
On-chip I^2C bus control function of MCU	I^2C bus communication function 1 channel (required)

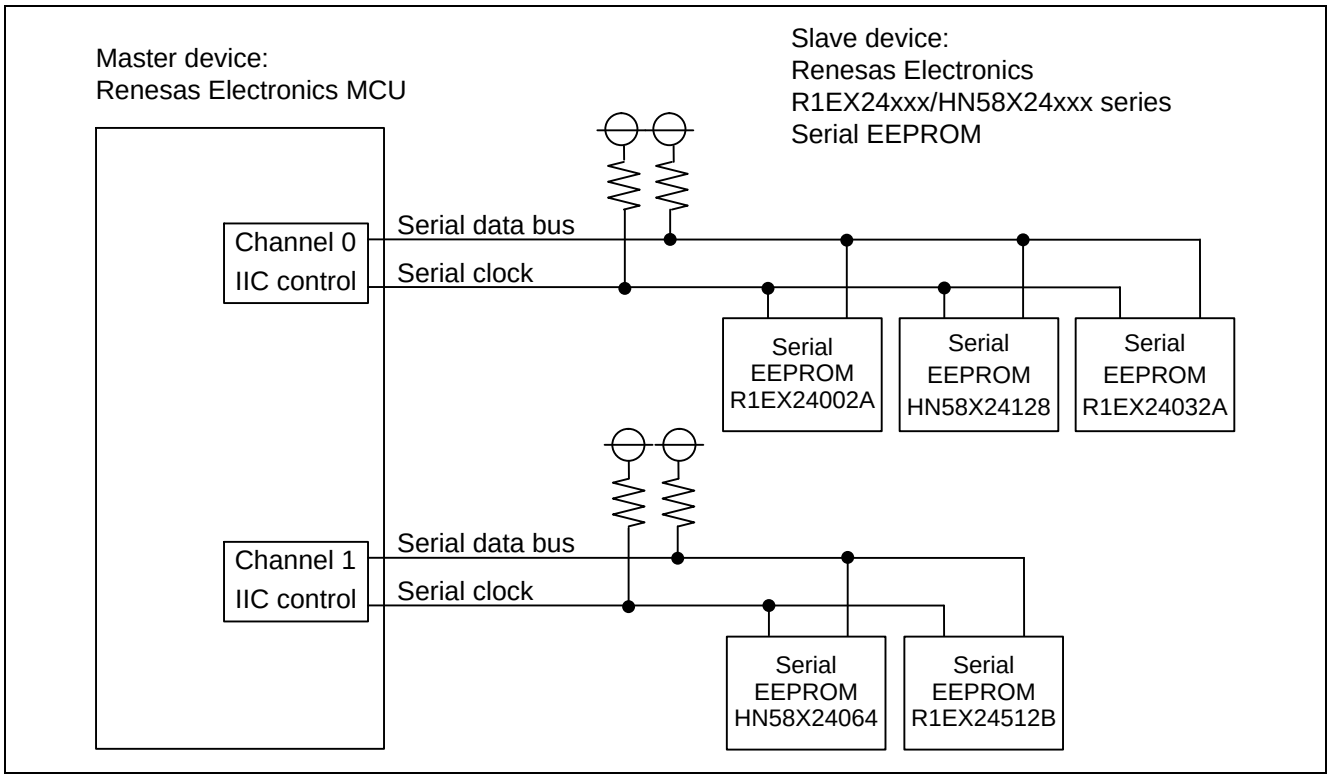


Figure 1.1 Usage Example

2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

2.1 RL78

(1) **RL78/G14 IICA Integrated Development Environment CS+ for CA,CX (Compiler: CA78K0R)**

Table 2.1 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^2C Serial EEPROM
MCU used for evaluation	RL78/G14 group (program ROM: 256 KB, RAM: 24 KB)
Operating frequency	Main system clock: 32 MHz Peripheral hardware clock: 32 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics CS+ for CA,CX V3.01.00
C compiler	Renesas Electronics RL78,78K0R compiler CA78K0R V1.71
	Compiler options: The default settings (-qx2) for the integrated development environment are used.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL76/L12, RL78/L13, RL78/L1C Group I²C Bus Single Master Control Software Using IICA Serial Interface (R01AN1074EJ), Ver. 1.03
Board used	Renesas Starter Kit for RL78/G14

(2) **RL78/G14 IICA Integrated Development Environment CS+ for CC (Compiler: CC-RL)**

Table 2.2 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^2C Serial EEPROM
MCU used for evaluation	RL78/G14 group (program ROM: 256 KB, RAM: 24 KB)
Operating frequency	Main system clock: 32 MHz Peripheral hardware clock: 32 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics CS+ for CC V3.03.00
C compiler	Renesas Electronics RL78 compiler CC-RL V1.02.00
	Compiler options: The default settings (Perform the default optimization(None)) for the integrated development environment are used.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL76/L12, RL78/L13, RL78/L1C Group I²C Bus Single Master Control Software Using IICA Serial Interface (R01AN1074EJ), Ver. 1.03
Board used	Renesas Starter Kit for RL78/G14

(3) **RL78/G14 IICA Integrated Development Environment IAR Embedded Workbench**

Table 2.3 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/G14 group (program ROM: 256 KB, RAM: 32 KB)
Operating frequency	Main system clock: 32 MHz Peripheral hardware clock: 32 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	IAR Systems IAR Embedded Workbench for Renesas RL78 (ver. 1.30.2)
C compiler	IAR Systems IAR Assembler for Renesas RL78 (ver. 1.30.2.50666) IAR C/C++ Compiler for Renesas RL78 (ver. 1.30.2.50666)
	Compiler options: The default settings ("level: low") for the integrated development environment are used.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Starter Kit for RL78/G14

(4) **RL78/G1C IICA Integrated Development Environment CubeSuite+**

Table 2.4 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/G14 group (program ROM: 32 KB, RAM: 5.5 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics CubeSuite+ V2.01.00
C compiler	Renesas Electronics CubeSuite+ RL78, RL78K0R compiler CA78K0R, V1.70
	Compile option Default settings (-qx2) of integrated development environment used as compile options.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Electronics RL78/G1C Target Board QB-R5F10JGC-TB

(5) **RL78/G1C IICA Integrated Development Environment IAR Embedded Workbench**

Table 2.5 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/G14 group (program ROM: 32 KB, RAM: 5.5 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	IAR Systems IAR Embedded Workbench for Renesas RL78 (ver. 1.30.5)
C compiler	IAR Systems IAR Assembler for Renesas RL78 (Ver.1.30.4.50715) IAR C/C++ Compiler for Renesas RL78（Ver.1.30.5.50715）
	Compiler options: The default settings ("level: low") for the integrated development environment are used.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Electronics RL78/G1C Target Board QB-R5F10JGC-TB

(6) **RL78/L12 IICA Integrated Development Environment CubeSuite+**

Table 2.6 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/L12 group (program ROM: 32 KB, RAM: 1.5 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics CubeSuite+ V2.01.00
C compiler	Renesas Electronics CubeSuite+ RL78, RL78K0R compiler CA78K0R, V1.70
	Compile option Default settings (-qx2) of integrated development environment used as compile options.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Starter Kit for RL78/L12

(7) **RL78/L12 IICA Integrated Development Environment IAR Embedded Workbench**

Table 2.7 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/G14 group (program ROM: 32 KB, RAM: 1.5 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	IAR Systems IAR Embedded Workbench for Renesas RL78 (ver. 1.30.5)
C compiler	IAR Systems IAR Assembler for Renesas RL78 (Ver.1.30.4.50715) IAR C/C++ Compiler for Renesas RL78（Ver.1.30.5.50715）
	Compiler options: The default settings ("level: low") for the integrated development environment are used.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Starter Kit for RL78/L12

(8) **RL78/L13 IICA Integrated Development Environment CubeSuite+**

Table 2.8 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/L13 group (program ROM: 128 KB, RAM: 8 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics CubeSuite+ V2.01.00
C compiler	Renesas Electronics CubeSuite+ RL78, RL78K0R compiler CA78K0R, V1.70
	Compile option Default settings (-qx2) of integrated development environment used as compile options.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Starter Kit for RL78/L13

(9) **RL78/L13 IICA Integrated Development Environment IAR Embedded Workbench**

Table 2.9 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/L13 group (program ROM: 128 KB, RAM: 8 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	IAR Systems IAR Embedded Workbench for Renesas RL78 (ver. 1.30.5)
C compiler	IAR Systems IAR Assembler for Renesas RL78 (Ver.1.30.4.50715) IAR C/C++ Compiler for Renesas RL78（Ver.1.30.5.50715）
	Compiler options: The default settings ("level: low") for the integrated development environment are used.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Starter Kit for RL78/L13

(10) **RL78/L1C IICA Integrated Development Environment CubeSuite+**

Table 2.10 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/L1C group (program ROM: 256 KB, RAM: 16 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics CubeSuite+ V2.01.00
C compiler	Renesas Electronics CubeSuite+ RL78, RL78K0R compiler CA78K0R, V1.70
	Compile option Default settings (-qx2) of integrated development environment used as compile options.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Starter Kit for RL78/L1C

(11) **RL78/L1C IICA Integrated Development Environment IAR Embedded Workbench**

Table 2.11 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RL78/L1C group (program ROM: 256 KB, RAM: 16 KB)
Operating frequency	Main system clock: 24 MHz Peripheral hardware clock: 24 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	IAR Systems IAR Embedded Workbench for Renesas RL78 (ver. 1.30.5)
C compiler	IAR Systems IAR Assembler for Renesas RL78 (Ver.1.30.4.50715) IAR C/C++ Compiler for Renesas RL78（Ver.1.30.5.50715）
	Compiler options: The default settings ("level: low") for the integrated development environment are used.
Sample code version	Ver. 1.01
Software used	RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ), ver. 1.02
Board used	Renesas Starter Kit for RL78/L1C

2.2 RX

(1) **RX62N RIIC**

Table 2.12 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RX62N group (program ROM: 512 KB, RAM: 64 KB)
Operating frequency	ICLK: 96 MHz PCLK: 48 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics High-performance embedded Workshop Version 4.09.01.007
C compiler	Renesas Electronics RX Family C/C++ Compiler Package (Toolchain 1.2.1.0)
	Compile option The integrated development environment default settings*1 are used. Note: 1. Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version	Ver. 1.00
Software used	RX600, RX200 Series I^{2}C Bus Single Master Control Software using RIIA Serial Interface (R01AN1254EJ), ver. 1.13
Board used	Renesas Starter Kit for RX62N

(2) **RX63N RIIC**

Table 2.13 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RX63N group (program ROM: 1 MB, RAM: 128 KB)
Operating frequency	ICLK: 96 MHz PCLK: 48 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics High-performance embedded Workshop Version 4.09.01.007
C compiler	Renesas Electronics RX Family C/C++ Compiler Package (Toolchain 1.2.1.0)
	Compile option The integrated development environment default settings*1 are used. Note: 1. Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version	Ver. 1.00
Software used	RX600, RX200 Series I^{2}C Bus Single Master Control Software using RIIC Serial Interface (R01AN1254EJ), ver. 1.13
Board used	Renesas Starter Kit for RX63N

(3) **RX63T RIIC**

Table 2.14 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RX63T group (program ROM: 512KB, RAM: 48 KB)
Operating frequency	ICLK: 96 MHz PCLK: 48 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics CubeSuite+ V2.00.00
C compiler	Renesas Electronics RX Family C/C++ Compiler Package (Toolchain 2.00.000)
	Compile option The integrated development environment default settings*[1] are used. Note: 1. Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version	Ver. 1.00
Software used	RX600, RX200 Series I^{2}C Bus Single Master Control Software using RIIC Serial Interface (R01AN1254EJ), ver. 1.13
Board used	Renesas Starter Kit for RX63T

(4) **RX210 RIIC**

Table 2.15 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RX210 group (program ROM: 512KB, RAM: 48KB)
Operating frequency	ICLK: 50 MHz PCLK: 25 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics High-performance embedded Workshop Version 4.09.01.007
C compiler	Renesas Electronics RX Family C/C++ Compiler Package (Toolchain 1.2.1.0)
	Compile option The integrated development environment default settings*[1] are used. Note: 1. Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version	Ver. 1.00
Software used	RX600, RX200 Series I^{2}C Bus Single Master Control Software using RIIC Serial Interface (R01AN1254EJ), ver. 1.13
Board used	Renesas Starter Kit for RX210

(5) **RX21A RIIC**

Table 2.16 Operation Confirmation Conditions

Item	Contents
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I^{2}C Serial EEPROM
MCU used for evaluation	RX21A group (program ROM: 512KB, RAM: 64KB)
Operating frequency	ICLK: 50 MHz PCLK: 25 MHz Transfer clock: 400 kHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics High-performance embedded Workshop Version 4.09.01.007
C compiler	Renesas Electronics RX Family C/C++ Compiler Package (Toolchain 1.2.1.0)
	Compile option The integrated development environment default settings*1 are used. Note: 1. Optimization level: 2, optimization method: Size priority
Endian order	Big endian / Little endian
Sample code version	Ver. 1.00
Software used	RX600, RX200 Series I^{2}C Bus Single Master Control Software using RIIC Serial Interface (R01AN1254EJ), ver. 1.13
Board used	Renesas Starter Kit for RX210

3. Reference Application Note

For additional information associated with this document, refer to the following application note.

- RL78/G14, RL78/G1C, RL78/L12, RL78/L13, RL78/L1C Group I^{2}C Bus Single Master Control Software using IICA Serial Interface (R01AN1074EJ)
- RX600, RX200 Series I^{2}C Bus Single Master Control Software using RIIC Serial Interface (R01AN1254EJ)

4. Hardware

4.1 Pins Used

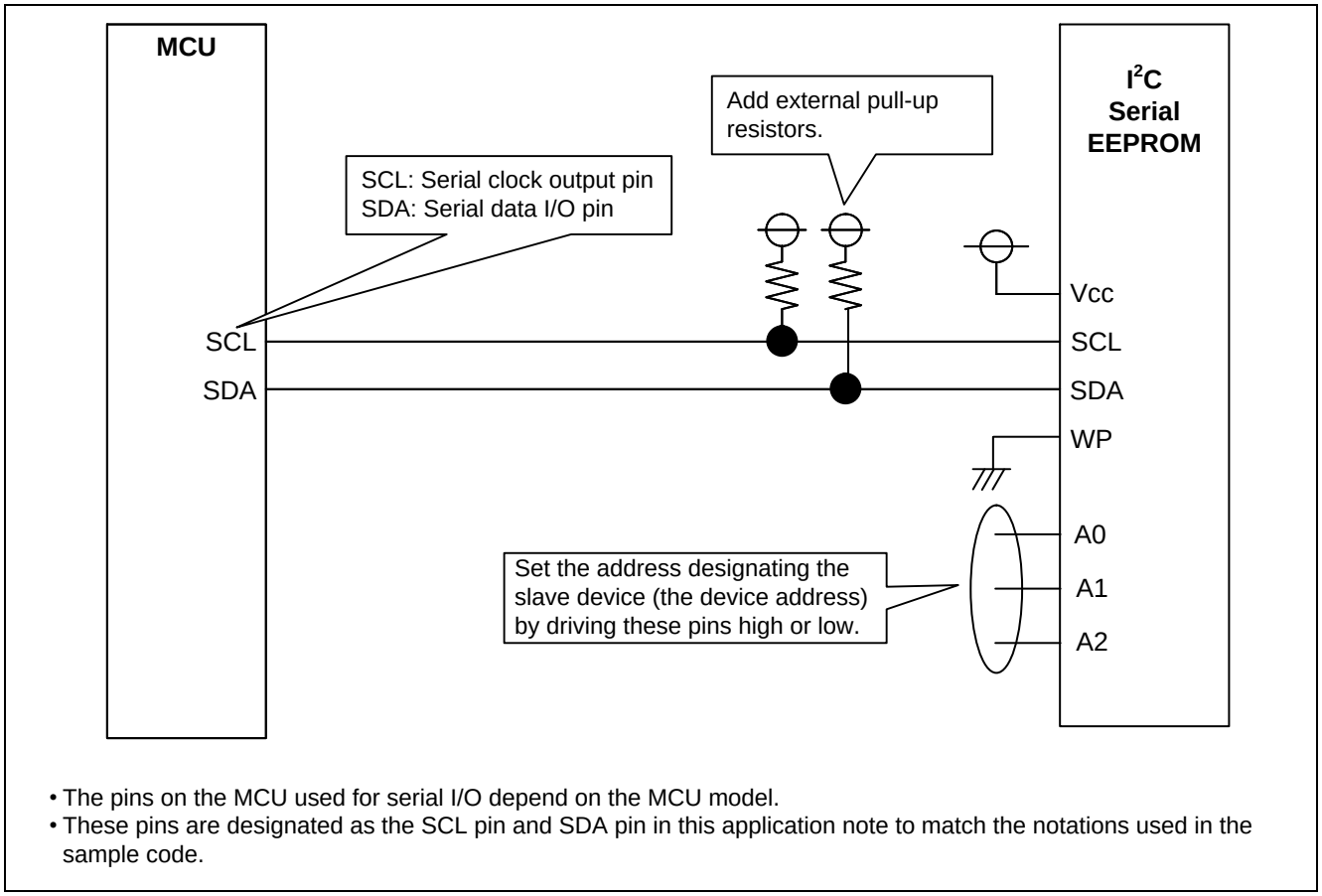
Table 4.1 lists the Pins Used and Their Functions.

Table 4.1 Pins Used and Their Functions

Pin Name	I/O	Description
SCL (SCL in Figure 4.1)	**Output**	Serial clock output
SDA (SDA in Figure 4.1)	**I/O**	Serial data I/O

4.2 Reference Circuit

Figure 4.1 is a connection diagram. Since the output is N-ch open drain, the serial clock line and serial data bus line require external pull-up resistors. Select resistors that are appropriate for the system. Also consider adding damping resistors to the signal lines to ensure matching circuit characteristics.



- The pins on the MCU used for serial I/O depend on the MCU model.
- These pins are designated as the SCL pin and SDA pin in this application note to match the notations used in the sample code.

Figure 4.1 MCU and I^2C Serial EEPROM Connection Example

4.3 Controlling Multiple Slave Devices

The sample code supports use of multiple channels. In addition, multiple slave devices with different type name can be connected to a single channel bus and controlled. However, communication with other devices is not possible during the period from when the start condition occurs to when the stop condition occurs.

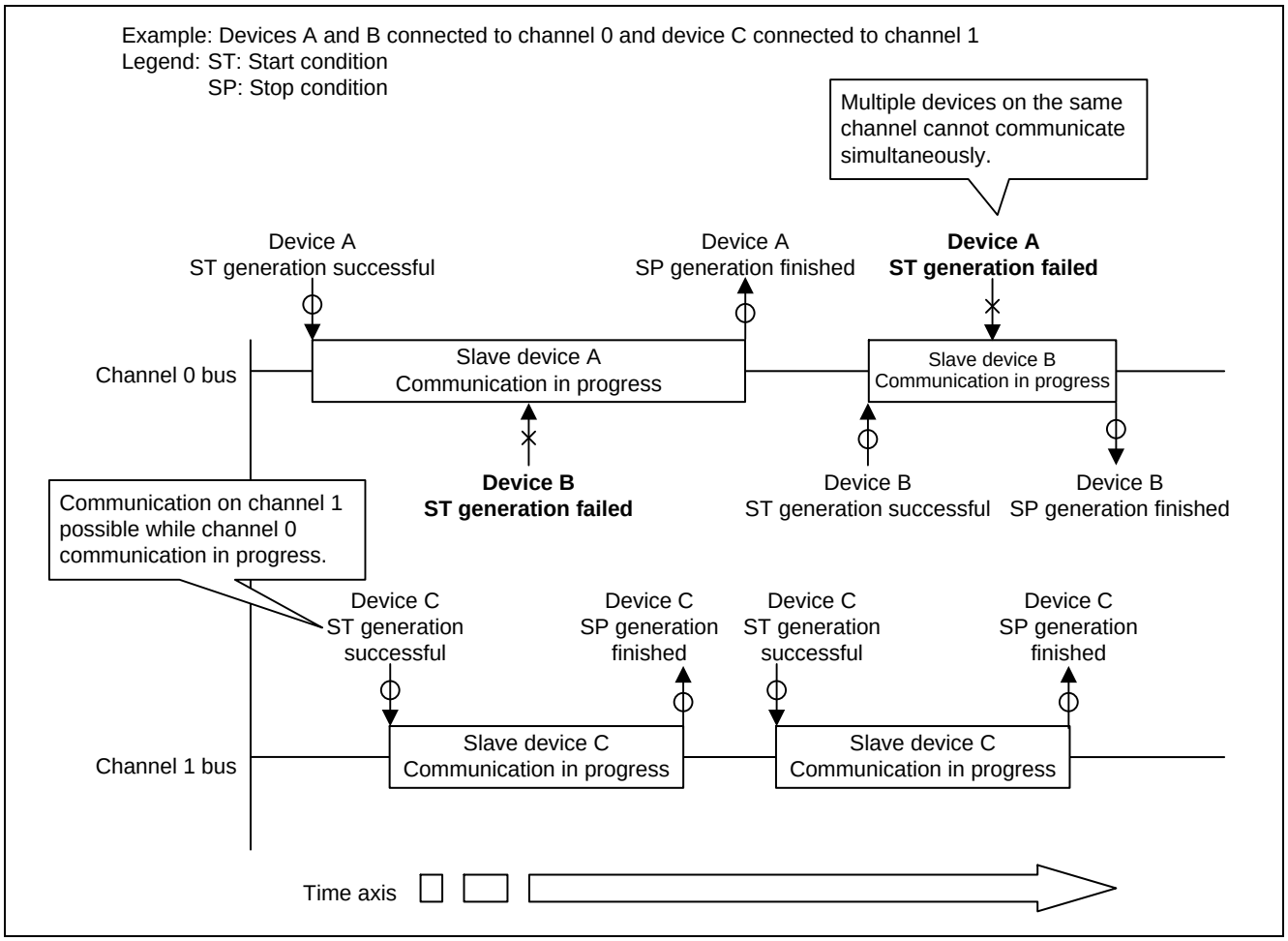


Figure 4.2 Example of Control of Multiple Slave Devices

4.4 Maximum Transfer Speed

The maximum transfer speed setting is 400 kHz.

However, when both standard mode and fast mode devices are connected to the same channel, the standard mode maximum setting of 100 kHz must be observed.

The maximum transfer speeds of mixed bus systems are listed below.

Table 4.2 Maximum Transfer Speeds of Mixed Bus Systems

Communication Device	Mixed Devices	
	Fast Mode	Standard Mode
Fast mode	0 to 400 kHz	0 to 100 kHz
Standard mode	0 to 100 kHz	0 to 100 kHz

5. Software

5.1 Operation Configuration

The software for controlling the slave devices is the upper layer, and the software that implements I^2C single master basic protocol control is the lower layer. The protocols provided by the upper and lower layers are used in combination to control the slave devices.

The sample code is positioned as the upper layer software for controlling serial EEPROM as slave devices.

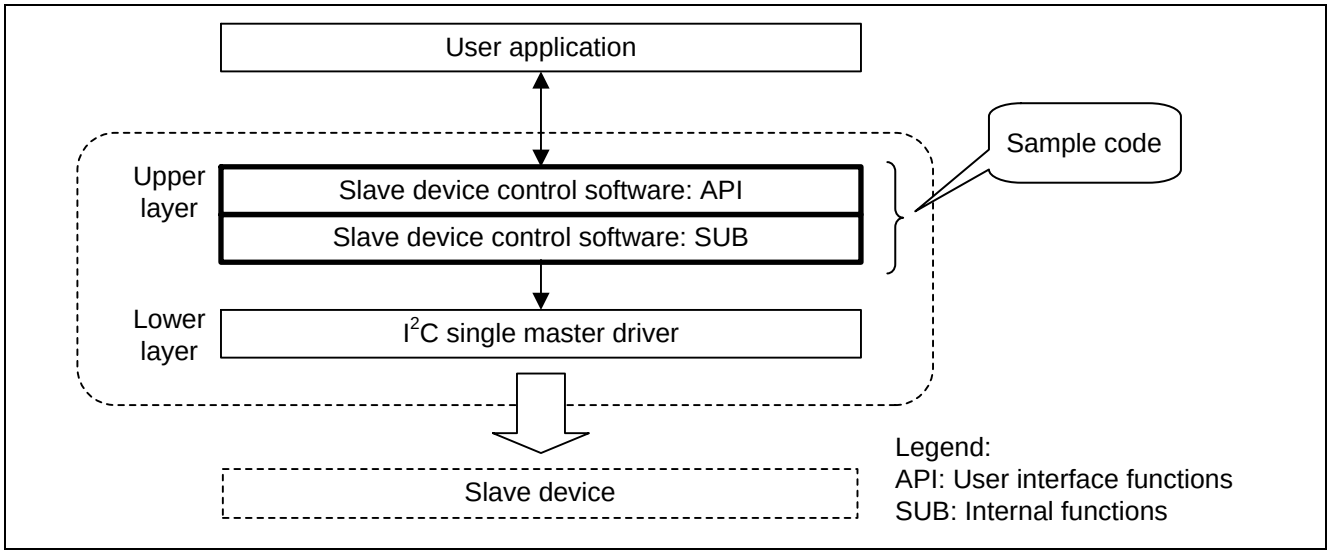


Figure 5.1 Software Configuration

5.2 Operation Overview

This section describes the method by which the on-chip I^2C bus control function of the MCU is used as the master of the single master I^2C bus to control EEPROM devices as slave devices.

5.2.1 Address Specification

The methods of specifying the *address that designates the slave device* and the *EEPROM internal address* differ depending on the capacity of the I^2C serial EEPROM used. The address specification methodology is described below.

(1) **Address that Designates the Slave Device**

After the start condition occurs, eight bits of data (device address word) are sent to the EEPROM. The EEPROM requires that the upper four bits have a fixed value of 1010. The next three bits (device address code) differ depending on the device used. See Table 5.2 for details.

Table 5.1 EEPROM Device Address Word

Device Code (Fixed)				Device Address Code			R/W Code
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	1	0	A2	A1	A0	R/W

(2) **Address Specification According to EEPROM Capacity**

Table 5.2 lists the specifications of the EEPROM devices covered by this application note. The values preceded by "A" in the device address code columns indicate addresses that designate the slave device, and values preceded by "a" indicate EEPROM internal addresses. Devices for which values of a16 or a10 to a8 are indicated in the device address code column require that the EEPROM internal address be indicated in the device address code.

For example, for the 8 Kb EEPROM device listed, bit 3 (A2) of the device address code indicates the connected EEPROM device. Bit 2 (a9) and bit 1 (a8) indicate the upper bits of the EEPROM internal address. So if the EEPROM internal address is "11 1001 0101h", the value of the top two bits (a9 and a8) is "11" and bits 2 and 1 of the device address code are set to this value.

Table 5.2 List of EEPROM Address Specifications According to EEPROM Capacity*[1]

Type Name	Capacity (Bits)	EEPROM Internal Address Length	Max. Connect-able Slave Devices	Device Address Code			EEPROM Internal Address	
				Bit 3	Bit 2	Bit 1	1st Byte	2nd Byte
HN58W241000I	1 M	17 bits (a16 to a0)	4	A2	A1	a16	a15 to a8	a7 to a0
R1EX24512B	512 K	16 bits (a15 to a0)	8	A2	A1	A0	a15 to a8	a7 to a0
R1EX24512A HN58X24512	512 K	16 bits (a15 to a0)	4	0 *[2]	A1	A0	a15 to a8	a7 to a0
R1EX24256B R1EX24256A HN58X24256	256 K	15 bits (a14 to a0)	8	A2	A1	A0	a14 to a8	a7 to a0
R1EX24128B R1EX24128A HN58X24128	128 K	14 bits (a13 to a0)	8	A2	A1	A0	a13 to a8	a7 to a0
R1EV24064A R1EX24064A HN58X2464	64 K	13 bits (a12 to a0)	8	A2	A1	A0	a12 to a8	a7 to a0
R1EX24032A HN58X2432	32 K	12 bits (a11 to a0)	8	A2	A1	A0	a11 to a8	a7 to a0
R1EX24016A HN58X2416	16 K	11 bits (a10 to a0)	1	a10	a9	a8	a7 to a0	
R1EX24008A HN58X2408	8 K	10 bits (a9 to a0)	2	A2	a9	a8	a7 to a0	
R1EV24004A R1EX24004A HN58X2404	4 K	9 bits (a8 to a0)	4	A2	A1	a8	a7 to a0	
R1EV24002A R1EX24002A HN58X2402	2 K	8 bits (a7 to a0)	8	A2	A1	A0	a7 to a0	

Notes: 1. This list of EEPROM products is current as of revision 1.00 of this document. For newer devices, refer to the respective datasheets.
2. Bit value doesn't matter.

5.2.2 Write Operation

This description of the sample code assumes for write operation that transmission of data from the master (MCU) to the slave (EEPROM) is defined as “writing” and that internal rewriting of the EEPROM with the data written to it is defined as “rewriting.”
The sample code supports byte write and page write operation.*

Note: Page write performs a batch rewrite of a user-defined byte count, relative to the page size specified for the EEPROM device used.

(1) **Data Write (WP = L)**

The I^2C single master control software’s master transmit mode (pattern 1) is used. First, a start condition (ST) is generated, next the address of the slave device is transmitted. In this case, the eighth bit is the transfer direction specification bit, and it has a value of “0” (write) during data transmission. Next, the EEPROM internal address is transmitted, and then data transmission starts.

The number of data bytes that can be rewritten within the page size are transmitted consecutively, then a stop condition is generated and the bus is released.

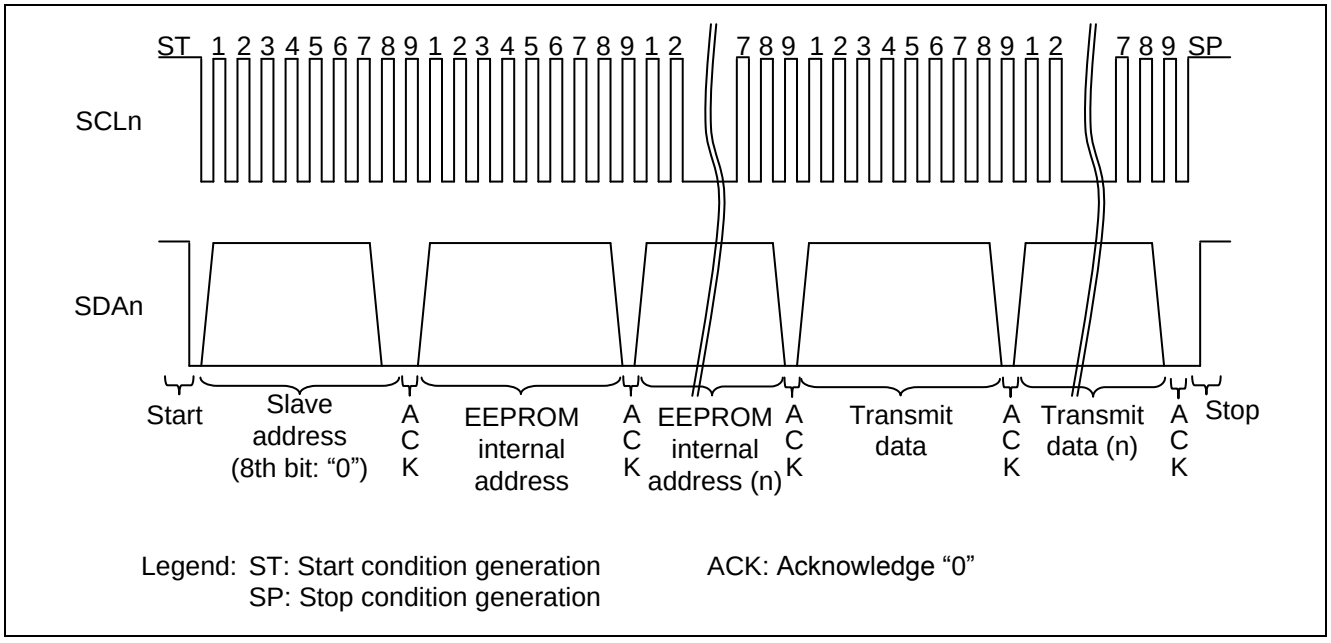


Figure 5.2 Data Write Operation

(2) **Data Write (WP = H)**

It is not possible to perform a data rewrite of the EEPROM if the write protect pin (WP) is in the high state when the write operation occurs.

The start condition → slave address transmission → EEPROM internal address transmission sequence operates successfully because an acknowledge “0” value is received every ninth bit. However, an acknowledge “1” value is received after transmit data, resulting in a NACK error. After the NACK error, a stop condition is generated and the bus is released.

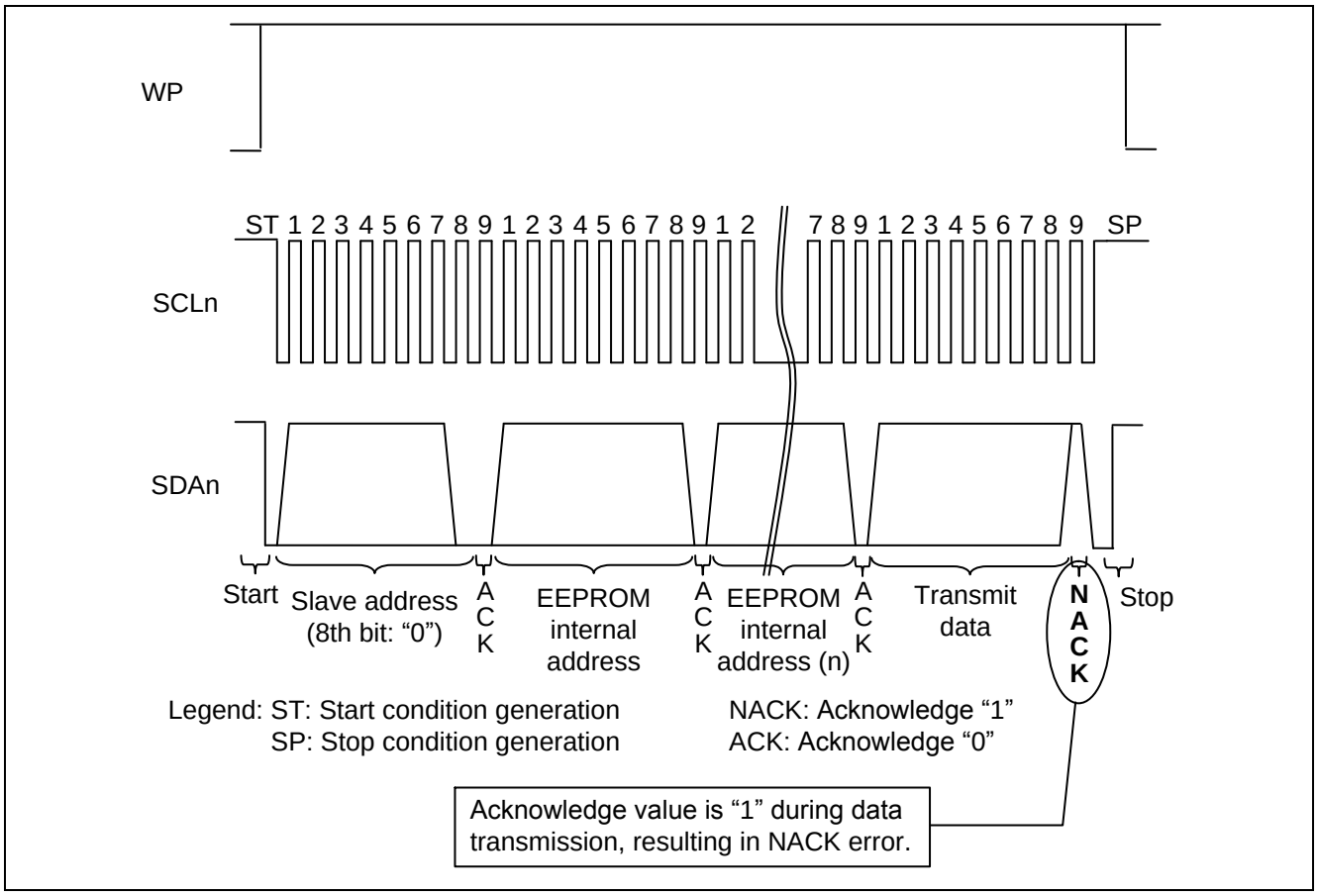


Figure 5.3 Write Operation with Write Protect Enabled (WP = H)

(3) **Acknowledge Polling**

After the data write finishes and the stop condition is received, the EEPROM commences rewrite operation. Acknowledge polling is a function used to determine whether the EEPROM rewrite has finished.

The sample code uses the I^2C single master control software's master transmit mode (pattern 3). First, a start condition (ST) is generated, next the address of the slave device is transmitted. In this case, the eighth bit is the transfer direction specification bit, and it has a value of "0" (write). The ninth bit, the acknowledge bit, is used to determine whether the rewrite has finished. An acknowledge value of "1" (NACK) indicates that the rewrite is in progress, and an acknowledge value of "0" (ACK) indicates that the rewrite is finished. Finally, a stop condition is generated and the bus is released.

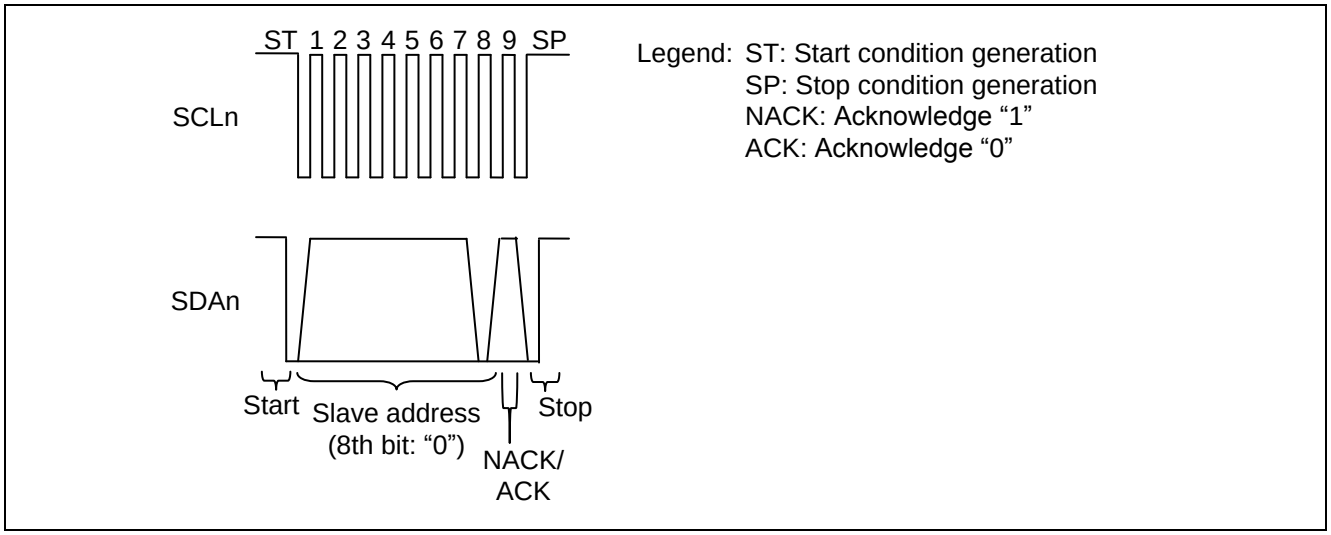


Figure 5.4 Acknowledge Polling (Rewrite Finished Determination)

(4) **Block Rewrite**

If the total number of data bytes to be rewritten is specified, the sample code calculates in software the size of the rewrite and performs a block rewrite. Block rewrite operation is described below.

The number of bytes that can be rewritten at one time by the EEPROM is one page. When the last address within a page is reached, the address is "rolled over" to the start address of the page. For this reason, to rewrite a quantity of data that extends beyond a page boundary (block rewrite), it is necessary to break the data up onto multiple portions that will each fit within a single page. Table 5.3 lists the page sizes of EEPROM devices of various capacities. The page size differs depending on the EEPROM specifications. Make settings appropriate to match the EEPROM devices used.

Table 5.3 List of Page Sizes of EEPROM Devices of Various Capacities*

Type Name	Page Size
HN58W241000I	256 bytes
R1EX24512B, R1EX24512A, HN58X24512	128 bytes
R1EX24256B, R1EX24128B, R1EX24256A, R1EX24128A, HN58X24256 HN58X24128	64 bytes
R1EV24064A, R1EX24064A, R1EX24032A, HN58X2464, HN58X2432, HN58X2416, HN58X2408	32 bytes
R1EX24016A, R1EX24008A, R1EV24004A, R1EX24004A, R1EX24002A	16 bytes
R1EV24002A, HN58X2402, HN58X2404	8 bytes

Note: This list of EEPROM products is current as of revision 1.00 of this document. For newer devices, refer to the respective datasheets.

5.2.3 Read Operation

This description of the sample code assumes for read operation that reception of data by the master (MCU) from the slave (EEPROM) is defined as “reading.”

The sample code supports random read and sequential read operation.

(1) **Data Read**

The I^2C single master control software’s master composite mode is used. First, a start condition (ST) is generated, next the address of the slave device is transmitted. In this case, the eighth bit is the transfer direction specification bit, and it has a value of “0” (write) during data transmission. Next, the EEPROM internal address is transmitted. Next, a restart condition (RST) is generated, and the slave address is transmitted. In this case, the eighth bit is the transfer direction specification bit, and it has a value of “1” (read). Then data reception starts.

In random read operation, the MCU transmits an acknowledge value of “1” to the EEPROM after receiving eight bits of data from the EEPROM, then a stop condition (SP) is generated and the bus is released.

In sequential read operation, acknowledge values of “0” are transmitted after data reception while consecutive reception is in progress. This causes the EEPROM internal address to be incremented so the next unit of data can be received. After the final unit of data is received, the MCU transmits an acknowledge value of “1” to the EEPROM, a stop condition (SP) is generated, and the bus is released.

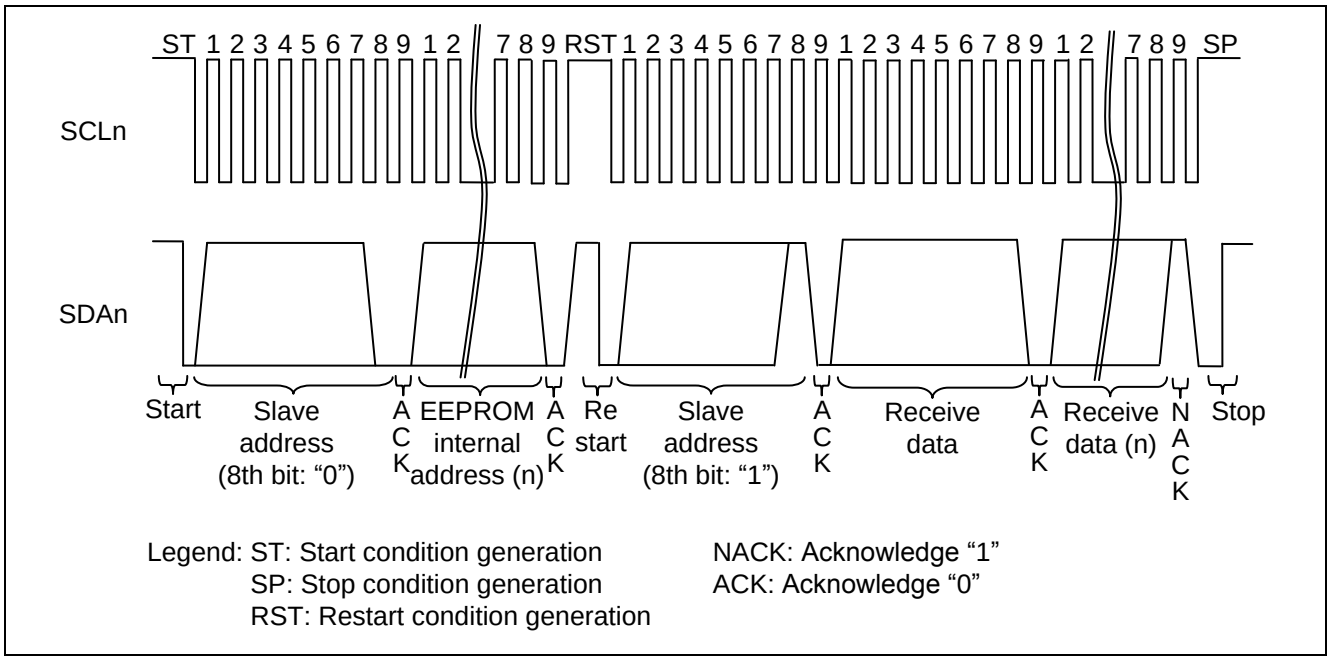


Figure 5.5 Data Read Operation

5.3 Software Operation

The specifications of the sample code were designed with OS control*[1] in mind. The supported processing methods are described below.

(1) **Normal Control (No OS)**

Communication is started by calling the start function. After this, processing to proceed with I^2C communication is performed by means of the EEPROM advance function, which is called by the user. The EEPROM advance function determines whether or not to proceed with I^2C communication by whether or not an I^2C interrupt is generated. The EEPROM advance function is not called by the interrupt handler. Rather, the software specification supports multiple calls on I^2C channels by using the main processing routine to call the EEPROM advance function.

In control not involving the OS, an event flag (g_iic_Event[])*[2] is set when an interrupt is generated. The EEPROM advance function recognizes the flag and executes communication.

Whether or not communication is in progress can be checked by means of the return value of the EEPROM advance function.

(2) **Normal Control (OS Present)**

The operation of this control mode has not been verified, so careful evaluation should be performed and modifications applied if necessary.

In control involving the OS, an event is indicated by an OS system call rather than an event flag.

When the EEPROM advance function is called after the start function was called, the system enters the system call standby state until an event occurs. An OS system call is generated when an interrupt occurs, and the EEPROM advance function executes the task (processing to proceed with I^2C communication).

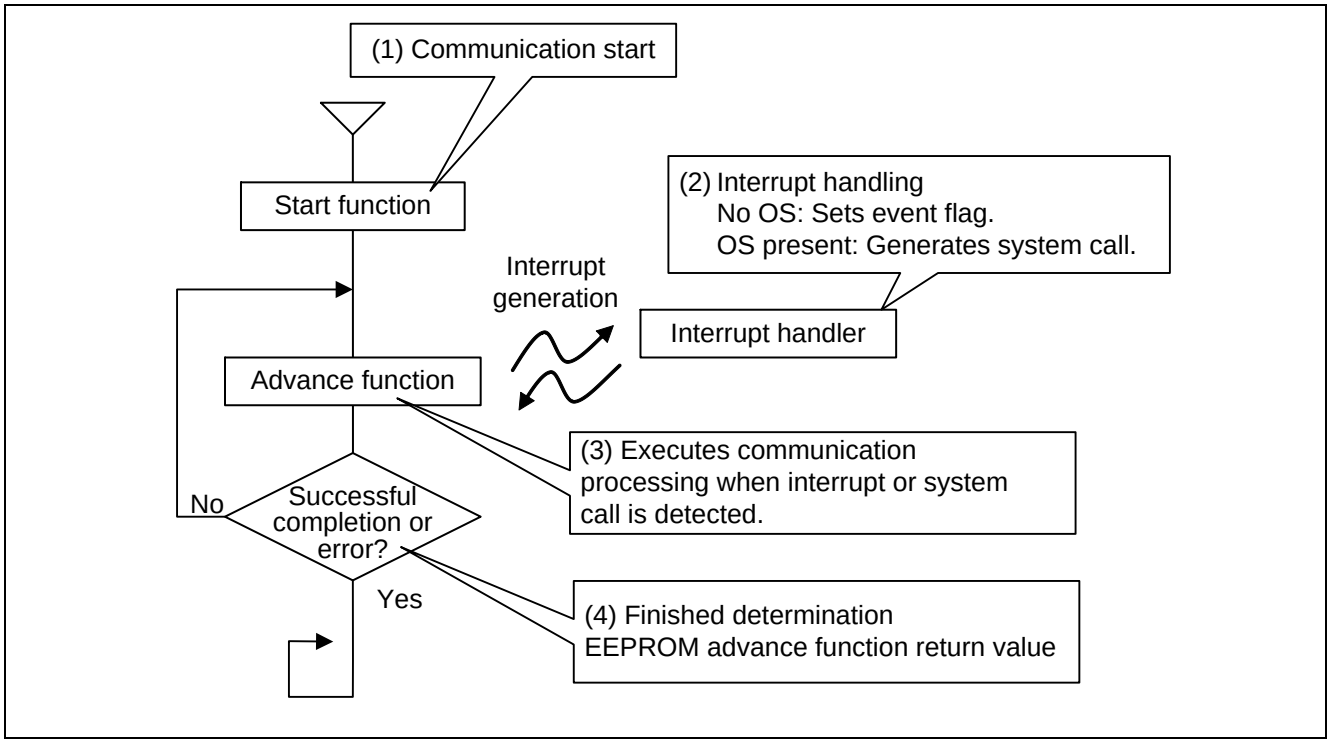


Figure 5.6 Software Operation Outline (No OS/ OS Present Control)

Notes: 1. The OS control capabilities the sample code assume µITRON 4.0.
2. For details, refer to the documentation of the I^2C single master control software.

5.4 Software Operation Sequence

(1) Normal Operation (No OS/OS Present)

The normal operation sequence (No OS/OS present) is shown below.

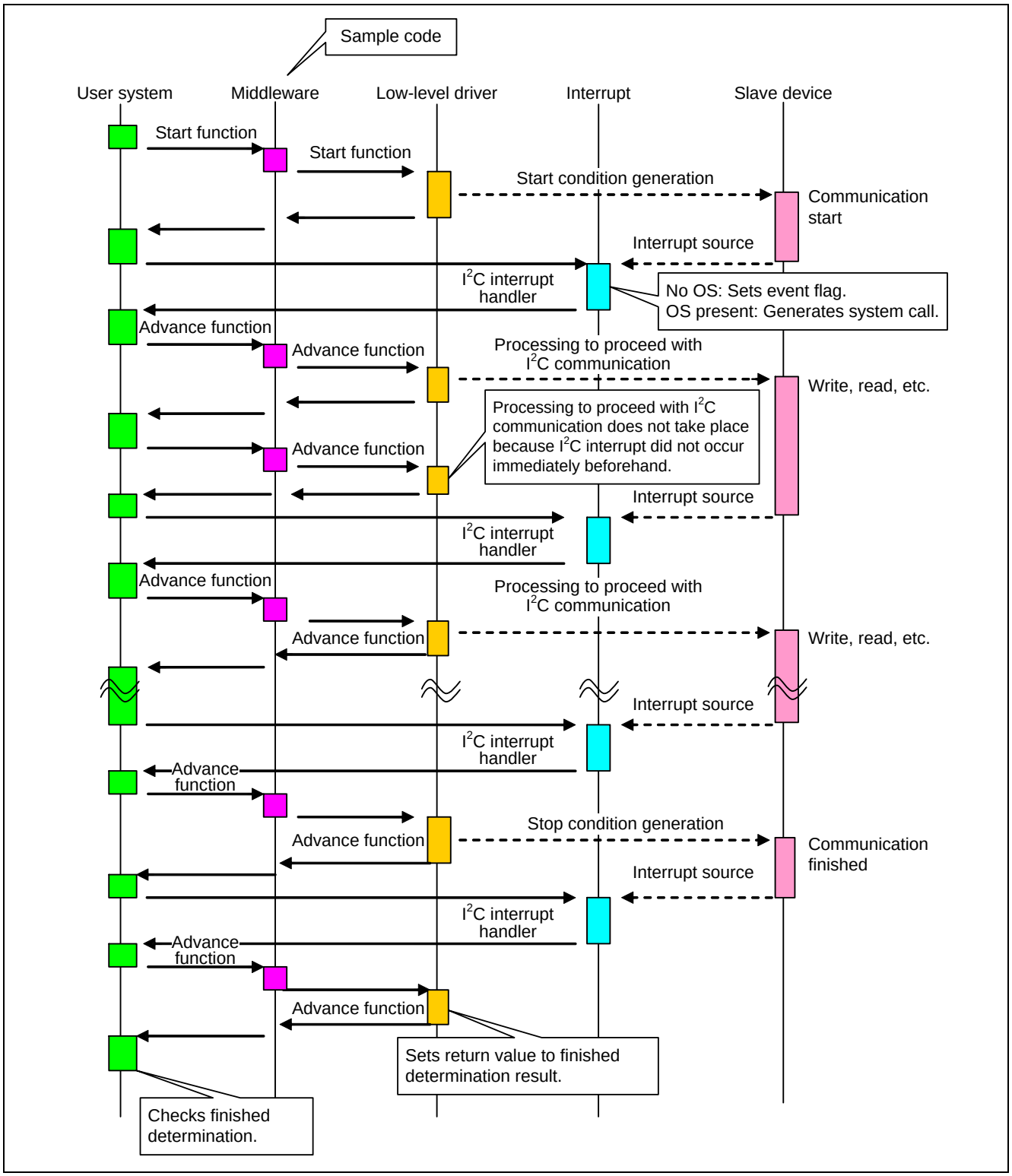


Figure 5.7 Normal Operation (No OS/OS Present) Sequence Diagram

5.5 Block Rewrite Implementation Method

Calling the write start function causes one page's worth of data to be written. To rewrite a quantity of data that extends beyond a page boundary, it is necessary to call the write start function again after finishing communication by means of the EEPROM advance function.

The sample code checks to determine if there will be data spanning a page boundary (referred to below as "leftover data") after finishing communication with the EEPROM advance function. If there will be leftover data, a return value indicating "leftover data present" is returned. A block rewrite can therefore be accomplished by repeatedly calling the write start function until a "no leftover data" return value is received.

5.6 Operation Flowcharts

5.6.1 Write Operation Flowchart

(1) **With Acknowledge Polling**

Figure 5.8 is a flowchart of write operation using acknowledge polling.

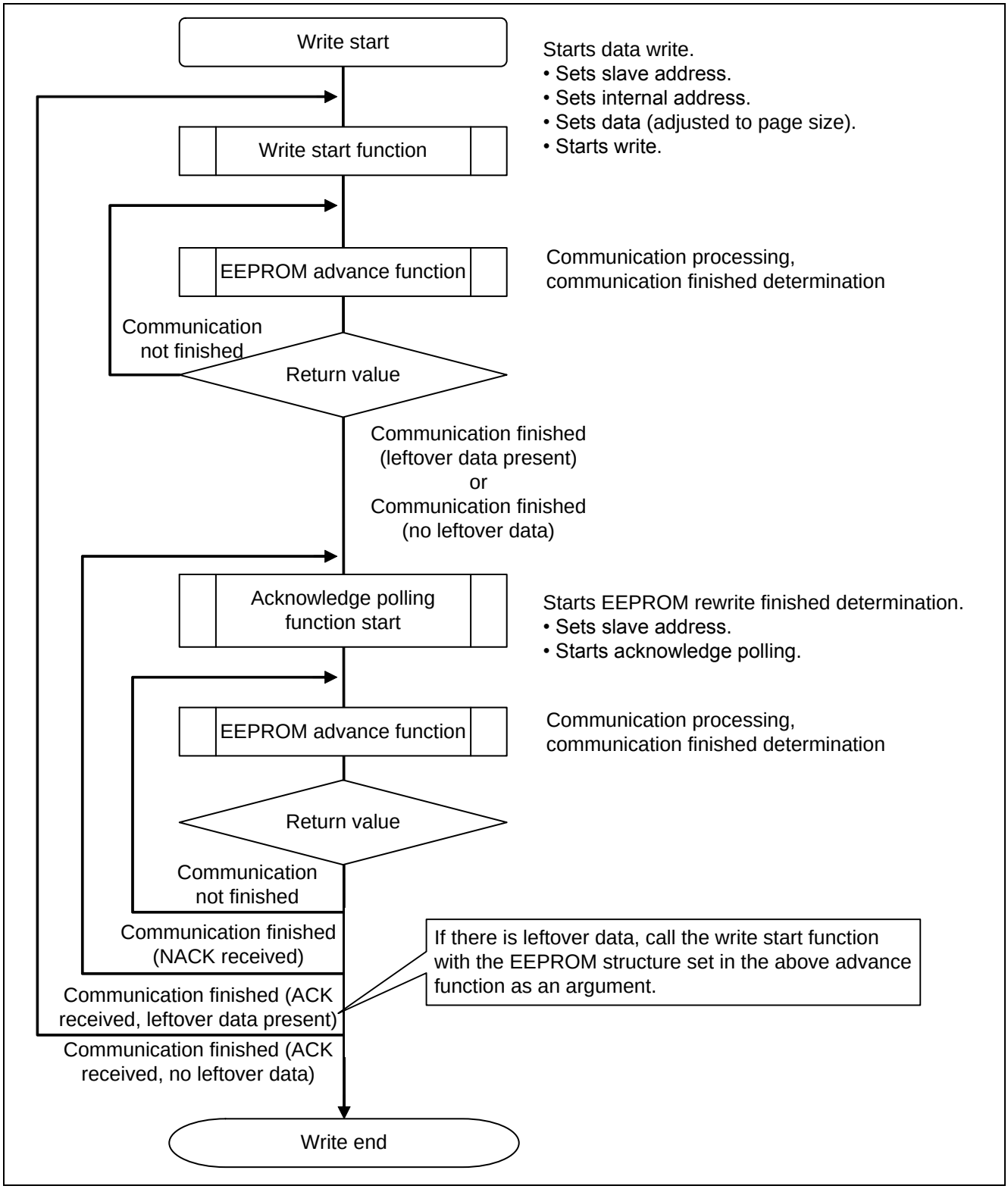


Figure 5.8 Write Operation Flowchart (Acknowledge Polling Used)

(2) **Without Acknowledge Polling**

Figure 5.9 is a flowchart of write operation without using acknowledge polling.

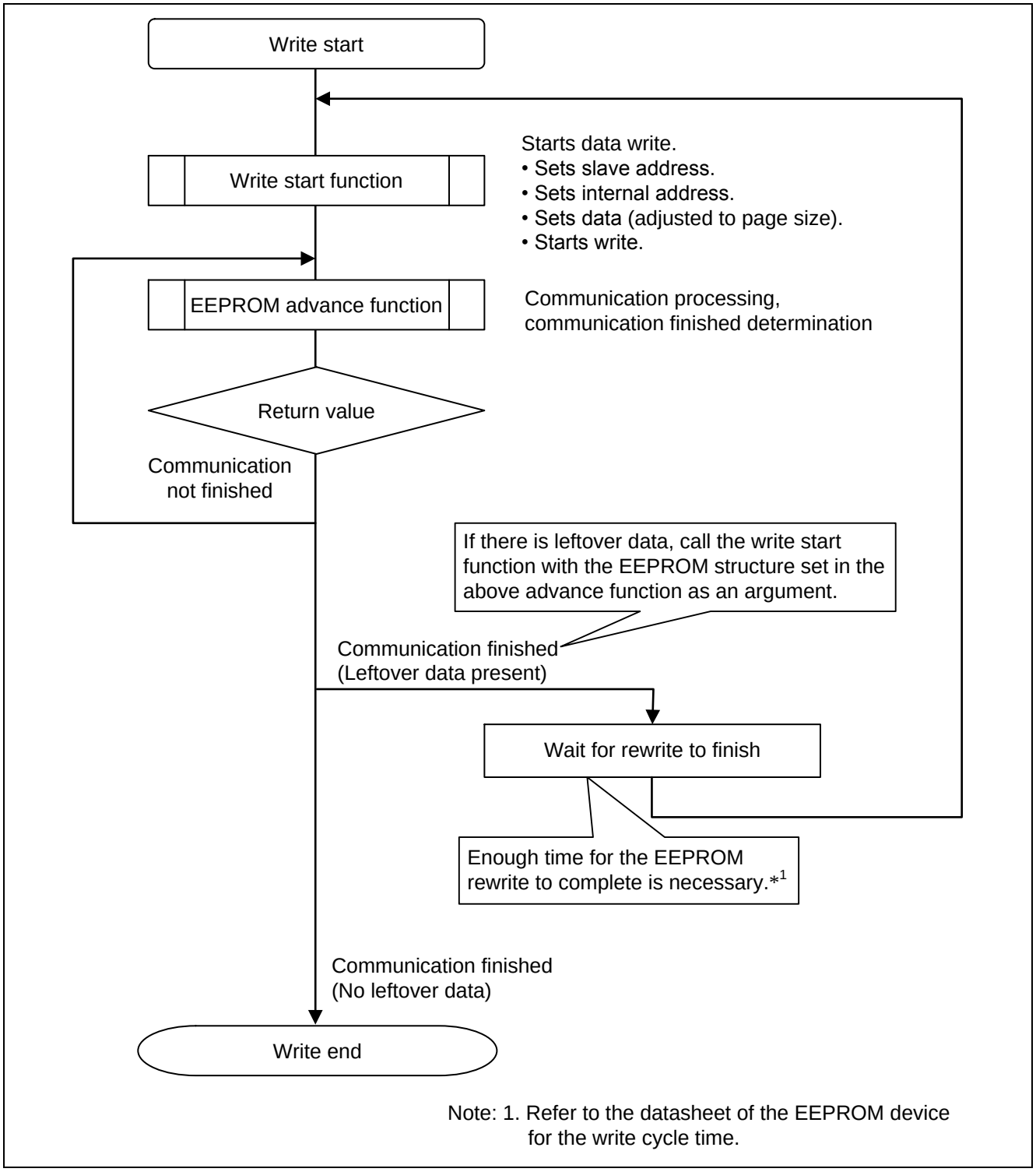


Figure 5.9 Write Operation Flowchart (Acknowledge Polling Not Used)

5.6.2 Read Operation Flowchart

Figure 5.10 is a flowchart of read operation.

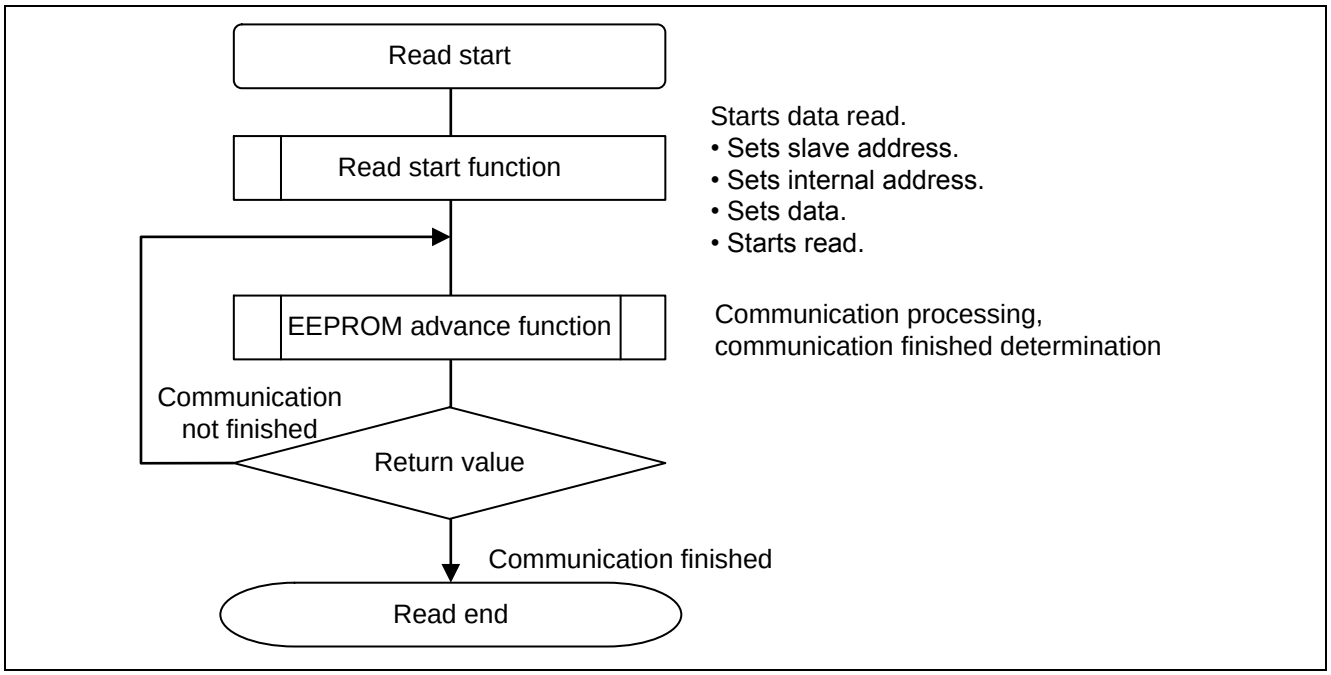


Figure 5.10 Read Operation Flowchart

5.7 Relationship of Data Buffers and Transmit/Receive Data

The sample code is a block device driver, and transmit/receive data pointers are set as arguments. The relationship of the data alignment of the data buffers in RAM and the transmit/receive order is described below. Regardless of the endian mode or serial communication function used, data is transmitted in the transmit data buffer alignment order, and data is written to the receive data buffer in the order received.

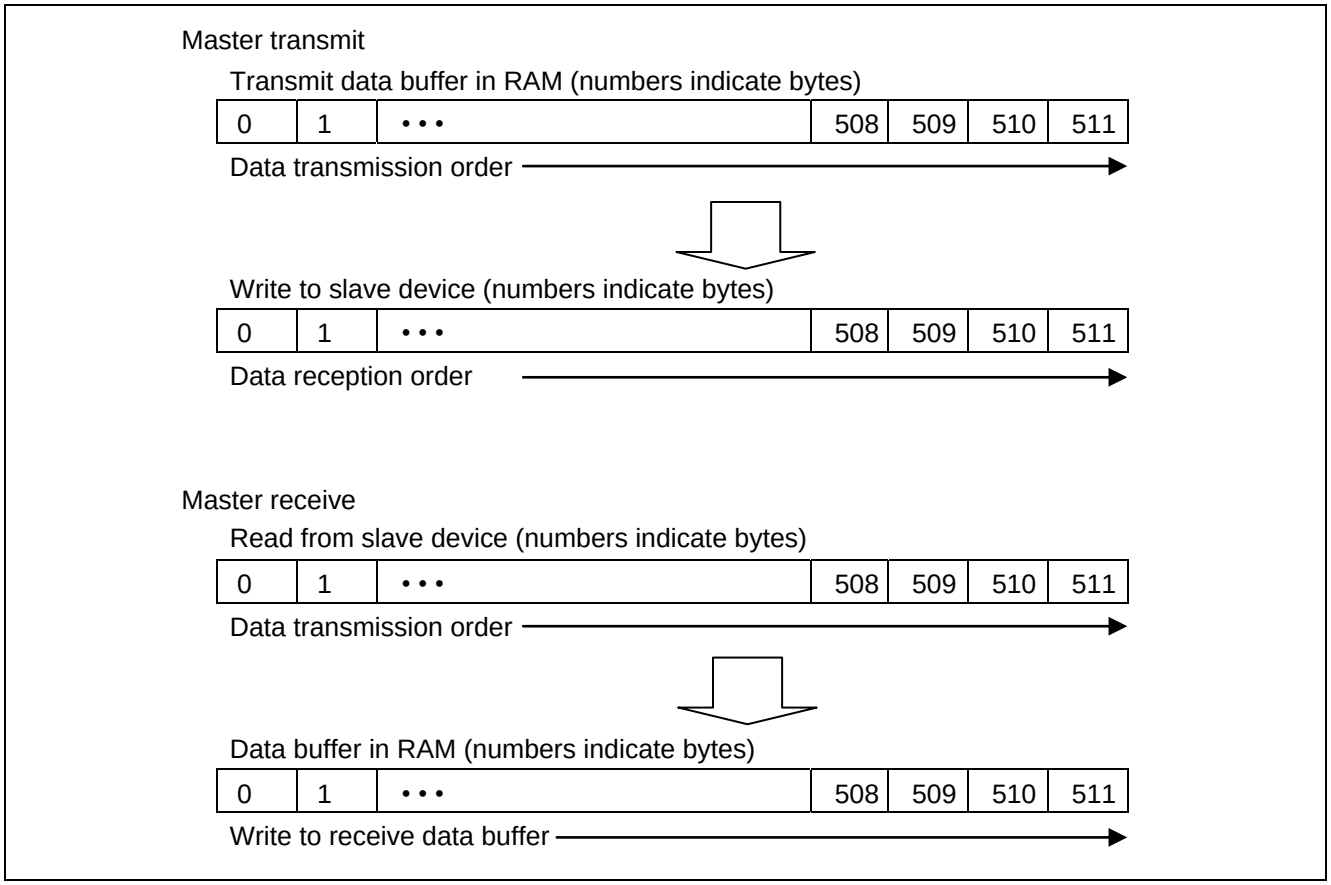


Figure 5.11 Storage of Transfer Data

5.8 Required Memory Sizes

The sizes of the required memory areas for each MCU of different instructions are given below. Investigate the instructions of MCU to be used and give by reference.

For information on the environment, see section 2, Verified Operating Conditions.

5.8.1 RL78

(1) **RL78/G14 IICA Integrated Development Environment CS+ for CA,CX (Compiler: CA78K0R)**

Table 5.4 Required Memory Sizes

Memory Used	Size	Remarks
ROM	2,047 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
RAM	2 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
Maximum usable user stack	90 bytes	
Maximum usable interrupt stack	—	

The required memory sizes differ according to the C compiler version and the compile options.

The ROM and RAM sizes listed do not include memory used by the lower layer I^2C single master control software.

The required memory sizes may differ from those listed above when a different MCU is used.

The maximum usable user stack size includes the stack used by the lower layer I^2C single master control software.

(2) **RL78/G14 IICA Integrated Development Environment CS+ for CC (Compiler: CC-RL)**

Table 5.5 Required Memory Sizes

Memory Used	Size	Remarks
ROM	1,329 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
RAM	2 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
Maximum usable user stack	70 bytes	
Maximum usable interrupt stack	—	

The required memory sizes differ according to the C compiler version and the compile options.

The ROM and RAM sizes listed do not include memory used by the lower layer I^2C single master control software.

The required memory sizes may differ from those listed above when a different MCU is used.

The maximum usable user stack size includes the stack used by the lower layer I^2C single master control software.

(3) **RL78/G14 IICA Integrated Development Environment IAR Embedded Workbench**

Table 5.6 Required Memory Sizes

Memory Used	Size	Remarks
ROM	3,892 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
RAM	4 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
Maximum usable user stack	272 bytes	
Maximum usable interrupt stack	—	

The required memory sizes differ according to the C compiler version and the compile options.

The ROM and RAM sizes listed do not include memory used by the lower layer I^2C single master control software.

The required memory sizes may differ from those listed above when a different MCU is used.

The maximum usable user stack size is the stack size for the entire project. It includes the stack used by the lower layer I^2C single master control software.

(4) **RL78/L13 IICA Integrated Development Environment CubeSuite+**

Table 5.7 Required Memory Sizes

Memory Used	Size	Remarks
ROM	1,963 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
RAM	4 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
Maximum usable user stack	78 bytes	
Maximum usable interrupt stack	—	

The required memory sizes differ according to the C compiler version and the compile options.

The ROM and RAM sizes listed do not include memory used by the lower layer I^2C single master control software.

The required memory sizes may differ from those listed above when a different MCU is used.

The maximum usable user stack size includes the stack used by the lower layer I^2C single master control software.

(5) **RL78/L13 IICA Integrated Development Environment IAR Embedded Workbench**

Table 5.8 Required Memory Sizes

Memory Used	Size	Remarks
ROM	3,482 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
RAM	4 bytes	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
Maximum usable user stack	146 bytes	
Maximum usable interrupt stack	—	

The required memory sizes differ according to the C compiler version and the compile options.

The ROM and RAM sizes listed do not include memory used by the lower layer I^2C single master control software.

The required memory sizes may differ from those listed above when a different MCU is used.

The maximum usable user stack size is the stack size for the entire project. It includes the stack used by the lower layer I^2C single master control software.

5.8.2 RX

(1) **RX63N RIIC**

Table 5.9 Required Memory Sizes

Memory Used	Size	Remarks
ROM	1,155 bytes(Little Endian)	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
RAM	1 bytes(Little Endian)	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
Maximum usable user stack	136 bytes	
Maximum usable interrupt stack	—	

The required memory sizes differ according to the C compiler version and the compile options.

The ROM and RAM sizes listed do not include memory used by the lower layer I^2C single master control software.

The required memory sizes may differ from those listed above when a different MCU is used.

The maximum usable user stack size includes the stack used by the lower layer I^2C single master control software.

(2) **RX210 RIIC**

Table 5.10 Required Memory Sizes

Memory Used	Size	Remarks
ROM	1,126 bytes(Little Endian)	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
RAM	1 bytes(Little Endian)	r_iic_eepmdl_api.c r_iic_eepmdl_sub.c
Maximum usable user stack	148 bytes	
Maximum usable interrupt stack	—	

The required memory sizes differ according to the C compiler version and the compile options.

The ROM and RAM sizes listed do not include memory used by the lower layer I^2C single master control software.

The required memory sizes may differ from those listed above when a different MCU is used.

The maximum usable user stack size includes the stack used by the lower layer I^2C single master control software.

5.9 File Structure

Table 5.11 lists the files used by the sample code. Note that files that are generated automatically by the integrated development environment are not listed.

Table 5.11 File Structure

\an_r01an1075ej0103_mcu_seep		<DIR>	Sample code folder
	r01an1075ej0103_mcu.pdf		Application note
	\ source	<DIR>	Program storage folder
		\ r_iic_eepmdl <DIR>	Serial EEPROM control software folder
		r_iic_eepmdl_api.c	API source file
		r_iic_eepmdl_api.h	API header file
		r_iic_eepmdl_sub.c	Internal function source file
		r_iic_eepmdl_sub.h	Internal function header file
		\ sample <DIR>	Operation verification program storage folder
		testmain.c	Sample source file for operation verification with 1 device
		testmain.h	Sample header file for operation verification with 1 device
		testmain2.c	Sample source file for operation verification with 2 device
		testmain2.h	Sample header file for operation verification with 2 device

5.10 Constants

5.10.1 Definitions

Definitions used in the sample code are listed below.

Table 5.12 Macro Definitions (Return Values)

Constant Name	Setting Value	Description
R_IIC_EEP_NO_INIT	(error_t)(0)	Uninitialized state
R_IIC_EEP_IDLE	(error_t)(1)	Idle state
R_IIC_EEP_COMMUNICATION	(error_t)(4)	Communication in progress: Write operation, acknowledge polling, or read operation communication in progress
R_IIC_EEP_LOCK_FUNC	(error_t)(5)	API processing in progress This occurs in the following case: • When another API is called while an API is processing
R_IIC_EEP_BUS_BUSY	(error_t)(6)	Bus busy This occurs in the following cases: • When the EEPROM initialization function or a start function is called while communication is in progress • When a start function or the EEPROM advance function is called while communication is in progress by another device on the same channel
R_IIC_EEP_FINISH_WRITE	(error_t)(21)	Idle state: Write finished Writing of all data finished
R_IIC_EEP_FINISH_WRITE_AGN	(error_t)(22)	Idle state: Write finished Page write finished, leftover data present
R_IIC_EEP_FINISH_ACKPOL	(error_t)(23)	Idle state: Acknowledge polling finished Rewriting of all data finished
R_IIC_EEP_FINISH_ACKPOL_AGN	(error_t)(24)	Idle state: Acknowledge polling finished Page rewrite finished, leftover data present
R_IIC_EEP_FINISH_ACKPOL_NACK	(error_t)(25)	Idle state: Acknowledge polling finished Page rewrite incomplete (NACK received)
R_IIC_EEP_FINISH_READ	(error_t)(26)	Idle state: Read finished Reading of all data finished
R_IIC_EPP_ERR_PARAM	(error_t)(-1)	Parameter error
R_IIC_EPP_ERR_AL	(error_t)(-2)	Arbitration lost error
R_IIC_EPP_ERR_NON_REPLY	(error_t)(-3)	No reply error
R_IIC_EPP_ERR_SDA_LOW_HOLD	(error_t)(-4)	SDA low-hold error when EEPROM recovery processing function called
R_IIC_EPP_ERR_OTHER	(error_t)(-5)	Other error
R_IIC_EEP_ERR_NACK	(error_t)(-21)	NACK received during write communication

Table 5.13 Macro Definitions (Change Prohibited)

Constant Name	Setting Value	Description
R_IIC_EEP_DEVCODE	(uint8_t)(0xa0)	EEPROM fixed device code
R_IIC_EEP_FALSE	(uint8_t)(0x00)	Flag “ON”
R_IIC_EEP_TRUE	(uint8_t)(0x01)	Flag “OFF”

Table 5.14 Macro Definitions (User Changeable)

Constant Name	Setting Value	Description
SCL_CLK_CNT	(uint8_t)(0x09)	SCL pseudo clock generation counter Defines the number of pseudo clock cycles that are generated and sent to SCL when the EEPROM recovery function is called. The counter value is set to 9 because a communication unit of nine clock cycles is common.

5.11 Structures and Unions

5.11.1 EEPROM Communication Information Structure

The EEPROM communication information structure used in the sample code is shown below. This structure contains the information required for communicating with the EEPROM. It must be set for each slave device used.

```
typedef struct
{
  r_iic_eepmdl_mode_t        EepMode;     /* Mode of EEPROM                    */
  r_iic_eepmdl_eepsize_t     EepSize;     /* Size of EEPROM                    */
  r_iic_eepmdl_pagesize_t    PageSize;    /* Size of write page                */
  uint32_t                   EepIntAdr;   /* Internal address of EEPROM        */
  uint32_t                   EepRWCnt;    /* R/W data counter                  */
  r_iic_drv_info_t           RIic_Info;   /* IIC information                   */
  uint8_t                    DevAdr;      /* Address to appoint a slave device */
  uint8_t                    rsv1;
  uint8_t                    rsv2;
  uint8_t                    rsv3;
} r_iic_eepmdl_info_t;
```

Figure 5.12 EEPROM Communication Information Structure

(1) **Description of Members**

Table 5.15 describes structure **r_iic_eepmdl_info_t**.

Table 5.15 List of Members of Structure r_iic_eepmdl_info_t

Structure Member	Allowable Setting Range	Description
EepMode	(Setting prohibited)	Operating mode There are four modes: Non-communication state, write in progress, acknowledge polling in progress, and read in progress.
EepSize	—	EEPROM capacity Capacities from 2 kbits to 1 Mbit are supported. Refer to the values defined in enumerated type **r_iic_eepmdl_romsize_t** (Table 5.17), and set EepSize to one of these defined values.
PageSize	—	EEPROM page size*1 *2 Sizes from 8 bytes to 256 bytes are supported. Refer to the values defined in enumerated type **r_iic_eepmdl_pagesize_t** (Table 5.18), and set PageSize to one of these defined values.
EepIntAdr	0000 0000h to FFFF FFFFh	EEPROM internal address (an)*1 Specify the start address for writing to or reading from the EEPROM. Use a setting value that is contained within the maximum of the EEPROM.
EepRWCnt	0000 0000h to FFFF FFFFh	Data counter (byte count)*1 Writing: transmit data counter (Sets the total number of data bytes to be transmitted.) Reading: receive data counter (Sets the total number of data bytes to be received.)
RIic_Info	—	I^2C single master control software I^2C communication information structure See Table 5.16 for details.
DevAdr	00h to 07h	EEPROM device address code (An)*3 As the EEPROM device address code, specify the device address code indicated by the physical connections of EEPROM pins A0 to A2. Table: Correspondence between physical pin connections and DevAdr setting value **A2 / A1 / A0 / DevAdr** L / L / L / 0 L / L / H / 1 L / H / L / 2 L / H / H / 3 H / L / L / 4 H / L / H / 5 H / H / L / 6 H / H / H / 7
rsv1 rsv2 rsv3	(Setting has no effect.)	Used for alignment.

Notes: 1. Restrictions on values differ according to the EEPROM capacity. See 5.2.2 (4) for details.
2. Set the page size specified for the EEPROM device used. The sample code can perform communication if a value greater than the page size of the EEPROM device used is set. Note that this will result in roll-over during writes.
3. The maximum number of slave devices that can be connected differs depending on the EEPROM type name. See 5.2.1 (2) for details.

Table 5.16 List of Members of I²C Communication Information Structure of I²C Single Master Control Software

Structure Member	Allowable Setting Range	Description
*pSlvAdr	—	Slave address storage buffer pointer This is the storage source location for the data specifying the slave address. Specify the address of the data storage source location. Reserve one byte.
*pData1st	—	EEPROM internal address storage buffer pointer This is the storage source location for the data specifying the EEPROM internal address. Specify the address of the data storage source location. Writing or reading: Reserve one byte if the EEPROM capacity is 16 Kbits or less. Reserve two bytes if the EEPROM capacity is 32 Kbits or more. Acknowledge polling: Setting not necessary. The setting value is ignored.
*pData2nd	—	Data storage buffer pointer Writing: Storage source location for the data written to the EEPROM Reading: Storage destination for the data read from the EEPROM Acknowledge polling: Setting not necessary. The setting value is ignored.
*pDevStatus	—	Device state flag pointer Refer to the documentation of the I²C single master control software for usage instructions.
Cnt1st	(Setting prohibited)	EEPROM internal address counter This is a data counter that specifies the EEPROM internal address. It is set by the sample code, so setting by the user is prohibited.
Cnt2nd	(Setting prohibited)	Data counter Writing: Number of data bytes written to one page Reading: Total number of data bytes to be read This is set by the sample code, so setting by the user is prohibited.
CallBackFunc	—	Call-back function Refer to the documentation of the I²C single master control software for usage instructions.
ChNo	00h to FFh	I²C bus control function channel number Set ChNo to the channel number of the bus used.
rsv1 rsv2 rsv3	(Setting has no effect.)	Used for alignment.

5.12 Enumerated Types

The enumerated type definitions used in the sample code are listed below.

Table 5.17 EEPROM Capacity List (enum r_iic_eepmdl_eepsize_t)

Definition	Description
R_IIC_EEP_EEPSIZE_002K	EEPROM capacity 2 kbits
R_IIC_EEP_EEPSIZE_004K	EEPROM capacity 4 kbits
R_IIC_EEP_EEPSIZE_008K	EEPROM capacity 8 kbits
R_IIC_EEP_EEPSIZE_016K	EEPROM capacity 16 kbits
R_IIC_EEP_EEPSIZE_032K	EEPROM capacity 32 kbits
R_IIC_EEP_EEPSIZE_064K	EEPROM capacity 64 kbits
R_IIC_EEP_EEPSIZE_128K	EEPROM capacity 128 kbits
R_IIC_EEP_EEPSIZE_256K	EEPROM capacity 256 kbits
R_IIC_EEP_EEPSIZE_512K	EEPROM capacity 512 kbits
R_IIC_EEP_EEPSIZE_001M	EEPROM capacity 1 Mbit

Table 5.18 EEPROM Page Size List (enum r_iic_eepmdl_pagesize_t)

Definition	Description
R_IIC_EEP_PAGESIZE_8B	EEPROM page size 8 bytes
R_IIC_EEP_PAGESIZE_16B	EEPROM page size 16 bytes
R_IIC_EEP_PAGESIZE_32B	EEPROM page size 32 bytes
R_IIC_EEP_PAGESIZE_64B	EEPROM page size 64 bytes
R_IIC_EEP_PAGESIZE_128B	EEPROM page size 128 bytes
R_IIC_EEP_PAGESIZE_256B	EEPROM page size 256 bytes

Table 5.19 EEPROM Operating Mode (enum r_iic_eepmdl_mode_t)

Definition	Description
R_IIC_EEP_MODE_NONE	Non-communication state
R_IIC_EEP_MODE_WRITE	Write in progress
R_IIC_EEP_MODE_ACKPOL	Acknowledge polling in progress
R_IIC_EEP_MODE_READ	Read in progress

5.13 Variables

Table 5.20 lists the global variable.

Table 5.20 Global Variable

Type	Valuable	Description	Used by Function
bool	g_iic_EepMdl_Api[MAX_IIC_CH_NUM]*	EEPROM API flag This is used to prevent overlapping API calls by the sample code. It is set when API processing starts and cleared after it finishes.	R_IIC_EepMdl_Init() R_IIC_EepMdl_Write() R_IIC_EepMdl_AckPolling() R_IIC_EepMdl_Read() R_IIC_EepMdl_Advance() R_IIC_EepMdl_Recovery()

Note: * The value of MAX_IIC_CH_NUM is defined by the I^2C single master control software. It stores a value for “maximum number of channels that can be used simultaneously + 1.”

5.14 Functions

Table lists the Functions.

Table 5.21 Functions

Function Name	Outline
R_IIC_EepMdl_Init()	EEPROM initialization function
R_IIC_EepMdl_Write()	Write start function
R_IIC_EepMdl_AckPolling()	Acknowledge polling start function
R_IIC_EepMdl_Read()	Read start function
R_IIC_EepMdl_Advance()	EEPROM advance function
R_IIC_EepMdl_Recovery()	EEPROM recovery function

5.15 State Transition Diagram

Figure 5.13 is a diagram showing state transitions for each channel.

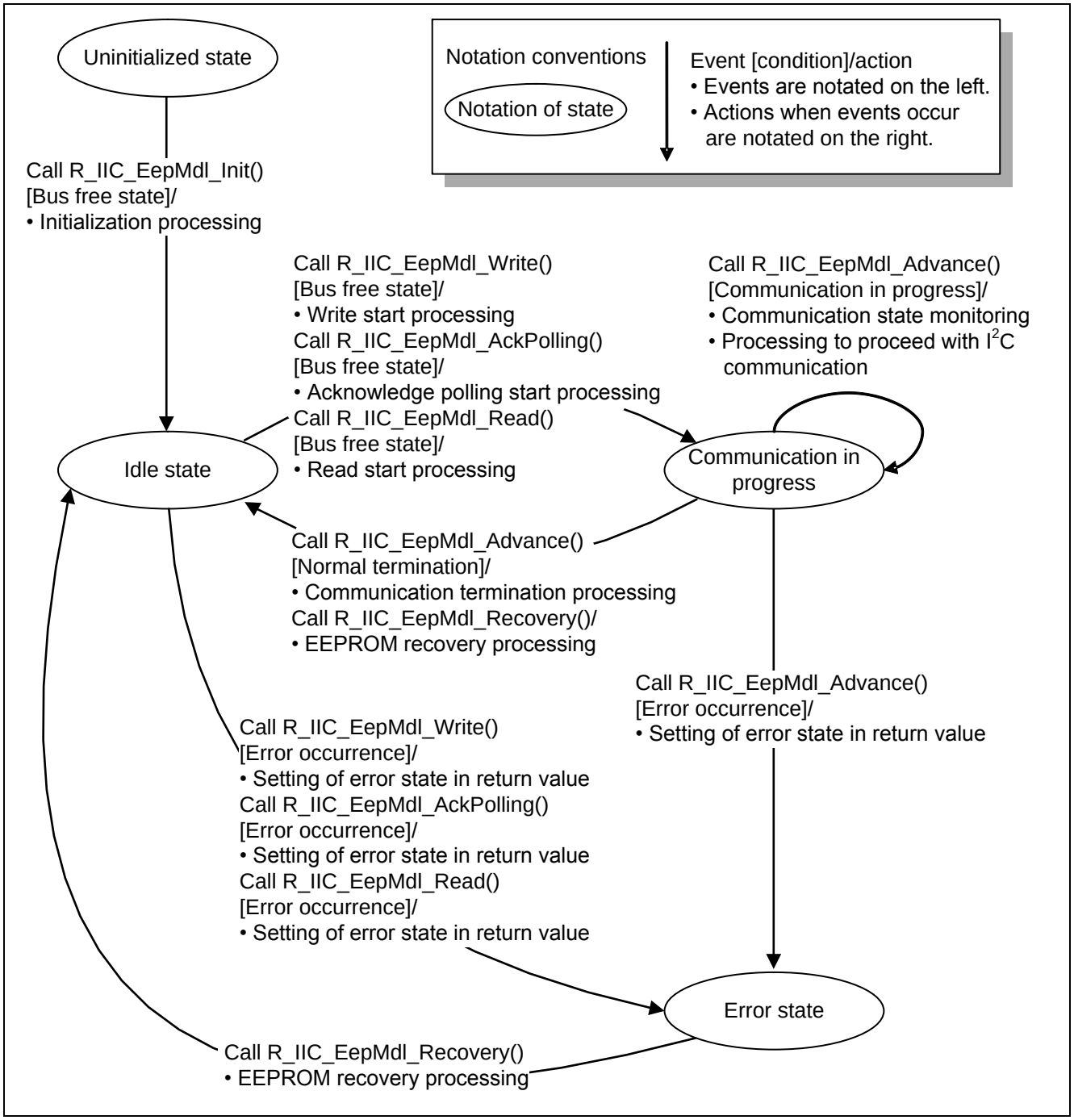


Figure 5.13 State Transition Diagram

5.16 Function Specifications

In the user application, reserve a communication information storage area for the device and, during control, set parameters and call functions as appropriate.

5.16.1 Common Function Processing

The sample code allows one function to be called at a time. If, while a function is being processed, another sample code function is called, the second function is terminated without processing. In this case R_IIC_EEP_LOCK_FUNC is returned as a return value.

The EEPROM API flag is provided to prevent more than one function from being called at the same time. This flag is set while a function is being processed. Each function first checks the flag and executes processing only if the flag has not beet set. Figure 5.14 is an outline flowchart illustrating this operation.

This processing is performed for the functions defined in section 5.14. The processing details of the API functions shown in Figure 5.14 are listed in sections 5.16.2 and subsequent below.

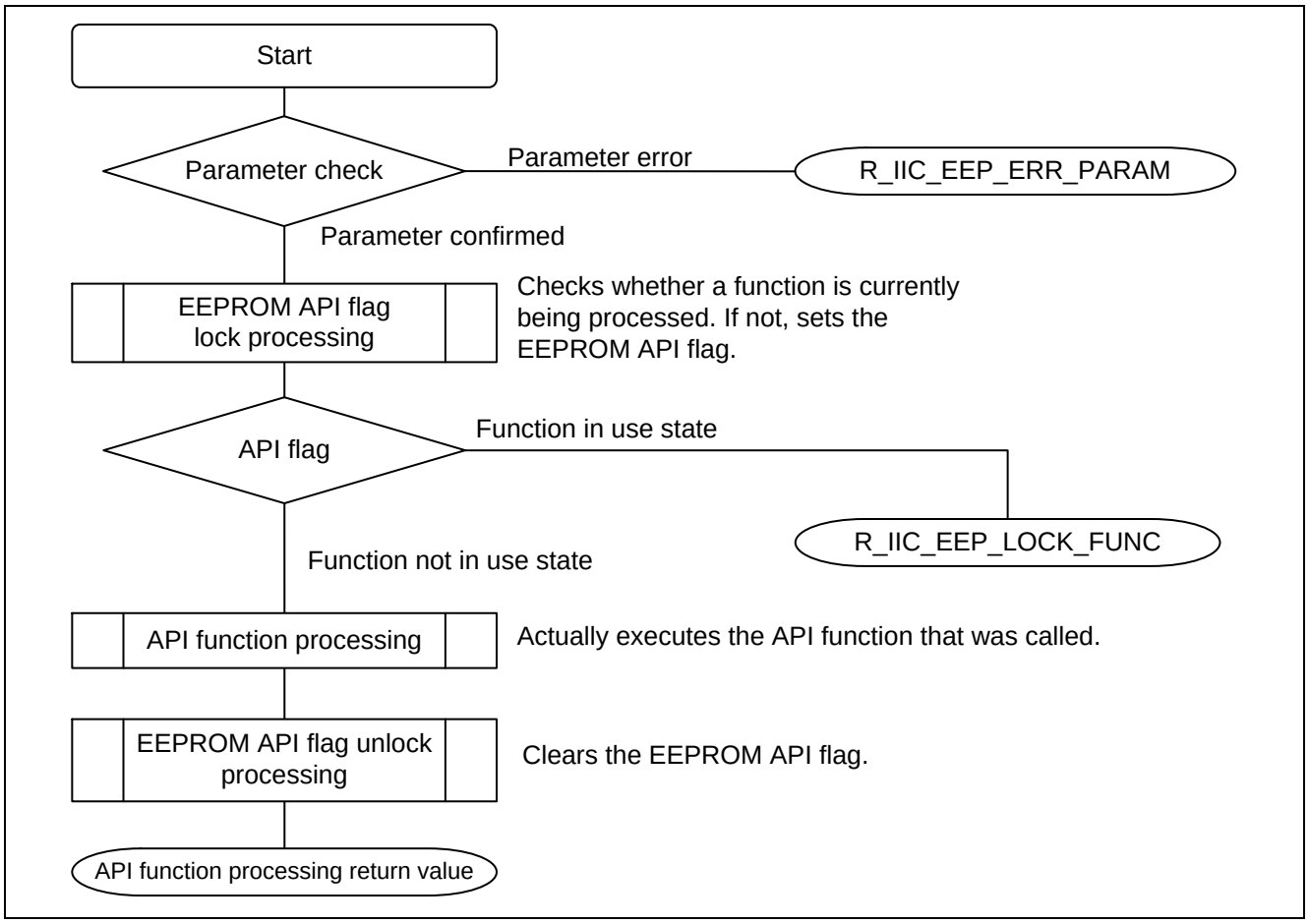


Figure 5.14 Outline Flowchart of Processing to Prevent Multiple Function Calls

5.16.2 EEPROM Initialization Function

R_IIC_EepMdl_Init

Outline EEPROM initialization function

Header r_iic_eepmdl_api.h, r_iic_eepmdl_sub.h

Declaration error_t R_IIC_EepMdl_Init(r_iic_eepmdl_info_t FAR *pEep_Info)

Description
- Makes initial settings for the target channel. After this, the device transitions to the idle state and communication is possible.
- The following settings are required to perform this processing.
  Member **RIic_Info.ChNo** of structure **r_iic_eepmdl_info_t**: Channel number used
  Channel flag (g_iic_ChStatus[]): Set to R_IIC_NO_INIT*[1]
  Device state flag (*(pEep.Info.RIic_Info.pDevStatus)): Set to R_IIC_NO_INIT*[1]

Arguments r_iic_eepmdl_info_t FAR *pEep_info ; EEPROM communication information structure pointer

Return Value

R_IIC_EEP_IDLE
Transitioned to idle state after initialization. No initialization performed if already initialized.
→ Communication is now possible by calling the start function.

R_IIC_EEP_LOCK_FUNC
No processing was performed because another API was being processed.
→ Call the function after processing of the other API finishes.

R_IIC_EEP_BUS_BUSY
Communication is in progress. Could not initialize.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_ERR_PARAM
Parameter error.
→ Check the setting value(s).

R_IIC_EEP_ERR_AL
Arbitration lost.
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_NON_REPLY
No reply error.*[2]
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_SDA_LOW_HOLD
SDA cannot be restored from low-hold state.
→ Check the system state to determine if the slave device is in the low-hold state, if the master device is outputting a low signal, etc.

R_IIC_EEP_ERR_OTHER
Other error occurred.
→ Check the following.
— Confirm that the EEPROM communication information structure settings are correct.
— Check to determine if an error occurred in OS control.

Remarks

- Call this function once for the control target device.
- If the device has already been initialized, no I^2C driver initialization processing takes place.
- To retry initialization after an error, etc., call the EEPROM recovery function.

Notes: 1. This is a global variable defined by the I^2C single master control software. It manages the bus state (uninitialized/idle/communication in progress/error). Set **R_IIC_NO_INIT** before calling the EEPROM initialization function. If it is not set before the EEPROM initialization function is called, initialization processing may not perform.

2. The no reply error definition differs depending on the MCU being controlled. See the description of the no reply error in the documentation of the I^2C single master control software for details.

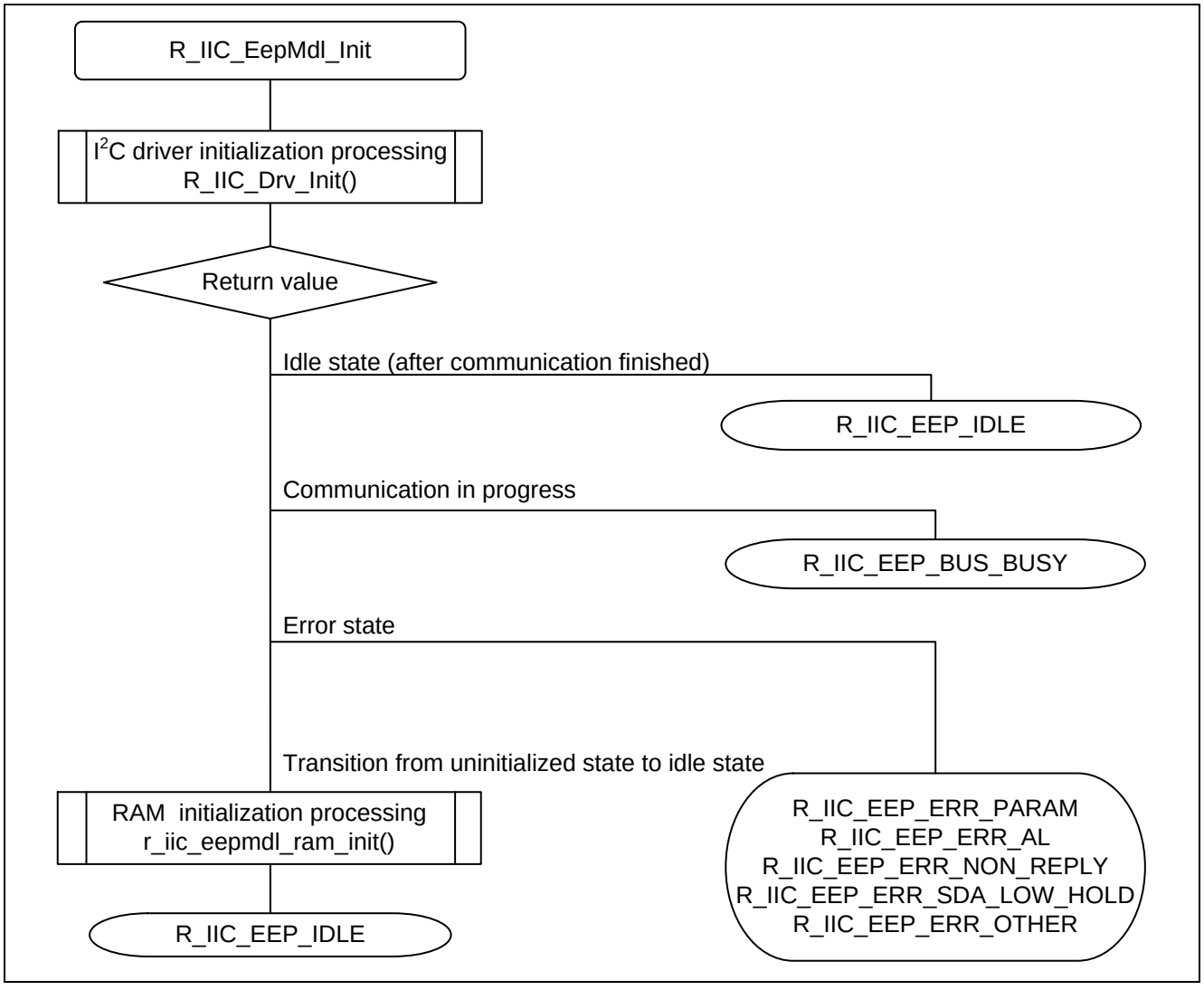


Figure 5.15 Outline of EEPROM Driver Initialization Function

5.16.3 Write Start Function

R_IIC_EepMdl_Write

Outline Write start function

Header r_iic_eepmdl_api.h, r_iic_eepmdl_sub.h

Declaration error_t R_IIC_EepMdl_Write(r_iic_eepmdl_info_t FAR *pEep_Info)

Description
- Starts data write to EEPROM.
- Use this function to write data from the master (MCU) to the slave (EEPROM).

Arguments r_iic_eepmdl_info_t FAR *pEep_Info ; EEPROM communication information storage pointer

Return Value

■ Return value (when R_IIC_EepMdl_Write() is called)

R_IIC_EEP_COMMUNICATION
Writing to EEPROM started.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_NO_INIT
Uninitialized state.
→ Call the EEPROM initialization function.

R_IIC_EEP_LOCK_FUNC
No processing was performed because another API was being processed.
→ Call the function after processing of the other API finishes.

R_IIC_EEP_BUS_BUSY
Communication is in progress. Processing of write to EEPROM could not start.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_ERR_PARAM
Parameter error.
→ Check the setting value(s).

R_IIC_EEP_ERR_AL
Arbitration lost.
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_NON_REPLY
No reply error.*[1]
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_SDA_LOW_HOLD
SDA cannot be restored from low-hold state.
→ Check the system state to determine if the slave device is in the low-hold state, if the master device is outputting a low signal, etc.

R_IIC_EEP_ERR_OTHER
Other error occurred.
→ Check the following.
— Confirm that the EEPROM communication information structure settings are correct.
— Check to determine if an error occurred in OS control.

■ Return value (when R_IIC_EepMdl_Advance() is called after R_IIC_EepMdl_Write())

R_IIC_EEP_COMMUNICATION
Communication is in progress.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_FINISH_WRITE
Writing of all data completed.
→ The acknowledge polling start function can be called to determine if rewriting to the EEPROM has finished.*[2]

R_IIC_EEP_FINISH_WRITE_AGN

Page write finished. There is leftover data.
→ Call the write start function after rewrite finishes. At this time do not change the member information of the EEPROM communication information structure.*3

See the description of the EEPROM advance function for information on the return values in case of error.

Remarks

- The I^2C single master control software's master transmit mode (pattern 1) is used.
- Settings must be made in **r_iic_eepmdl_info_t** to perform this processing. See 5.11.1 (1) for setting instructions.
- I^2C communication is not finished at the point of return from this function. It is necessary to call the EEPROM advance function to finish I^2C communication.
- Calling this function causes one page's worth of data to be written. If the end address of the write data is a larger value than the page boundary address, the write data is changed up to the page boundary.
- To rewrite the data beyond the page boundary, finish communication with the EEPROM advance function and then call the write start function again.

Notes: 1. The no reply error definition differs depending on the MCU being controlled. See the description of the no reply error in the documentation of the I^2C single master control software for details.
2. The write or read start function can be called without determining the rewrite status. To do this, call the start function after waiting for the EEPROM rewrite to finish. Refer to the datasheet of the EEPROM device for the write cycle time.
3. The leftover data write information is stored in the members of the EEPROM communication information structure, so communication cannot proceed correctly if the member values are changed.

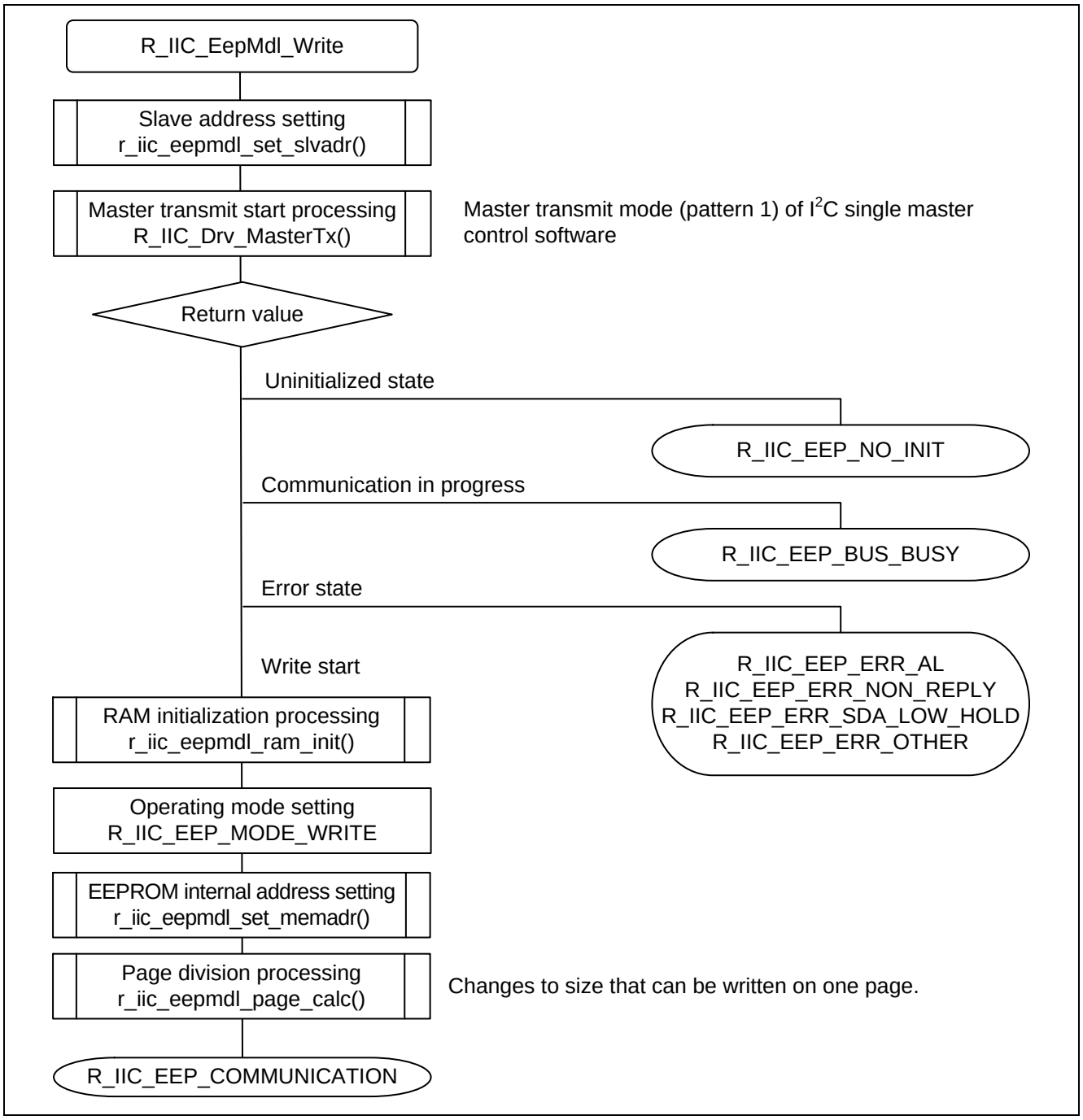


Figure 5.16 Outline of Write Start Function

5.16.4 Acknowledge Polling Start Function

R_IIC_EepMdl_AckPolling

Outline Acknowledge polling start function

Header r_iic_eepmdl_api.h, r_iic_eepmdl_sub.h

Declaration error_t R_IIC_EepMdl_AckPolling(r_iic_eepmdl_info_t FAR *pEep_Info)

Description
- Starts acknowledge polling (determination of data rewrite completion).
- After a data write from the master (MCU) to the slave (EEPROM) finishes, use this function to determine whether or not the EEPROM data rewrite has finished.

Arguments r_iic_eepmdl_info_t FAR *pEep_Info ; EEPROM communication information storage pointer

Return Value

■ Return value (when R_IIC_EepMdl_AckPolling() is called)

R_IIC_EEP_COMMUNICATION
Acknowledge polling started.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_NO_INIT
Uninitialized state.
→ Call the EEPROM initialization function.

R_IIC_EEP_LOCK_FUNC
No processing was performed because another API was being processed.
→ Call the function after processing of the other API finishes.

R_IIC_EEP_BUS_BUSY
Communication is in progress. Processing of Acknowledge polling could not start.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_ERR_PARAM
Parameter error.
→ Check the setting value(s).

R_IIC_EEP_ERR_AL
Arbitration lost.
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_NON_REPLY
No reply error.*[1]
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_SDA_LOW_HOLD
SDA cannot be restored from low-hold state.
→ Check the system state to determine if the slave device is in the low-hold state, if the master device is outputting a low signal, etc.

R_IIC_EEP_ERR_OTHER
Other error occurred.
→ Check the following.
— Confirm that the EEPROM communication information structure settings are correct.
— Check to determine if an error occurred in OS control.

■ Return value
(when R_IIC_EepMdl_Advance() is called after R_IIC_EepMdl_AckPolling)

R_IIC_EEP_COMMUNICATION
Communication is in progress.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_FINISH_ACKPOL
Rewriting of all data completed.
→ Communication is possible by calling the start function.

R_IIC_EEP_FINISH_ACKPOL_AGN

Page rewrite finished. There is leftover data.
→ Call the write start function. At this time do not change the member information of the EEPROM communication information structure.*2

R_IIC_EEP_FINISH_ACKPOL_NACK
Page rewrite did not finish (NACK received after transmission of slave address).
→ The acknowledge polling start function can be called to determine if rewriting to the EEPROM has finished.*3

See the description of the EEPROM advance function for information on the return values in case of error.

Remarks

- The I2C single master control software's master transmit mode (pattern 3) is used.
- Settings must be made in **r_iic_eepmdl_info_t** to perform this processing. See 5.11.1 (1) for setting instructions.
- I2C communication is not finished at the point of return from this function. It is necessary to call the EEPROM advance function to finish I2C communication.
- To rewrite the data beyond the page boundary, finish communication with the EEPROM advance function and then call the write start function again.

Notes: 1. The no reply error definition differs depending on the MCU being controlled. See the description of the no reply error in the documentation of the I2C single master control software for details.
2. The leftover data write information is stored in the members of the EEPROM communication information structure, so communication cannot proceed correctly if the member values are changed.
3. The write or read start function can be called without determining the rewrite status. To do this, call the start function after waiting for the EEPROM rewrite to finish. Refer to the datasheet of the EEPROM device for the write cycle time.

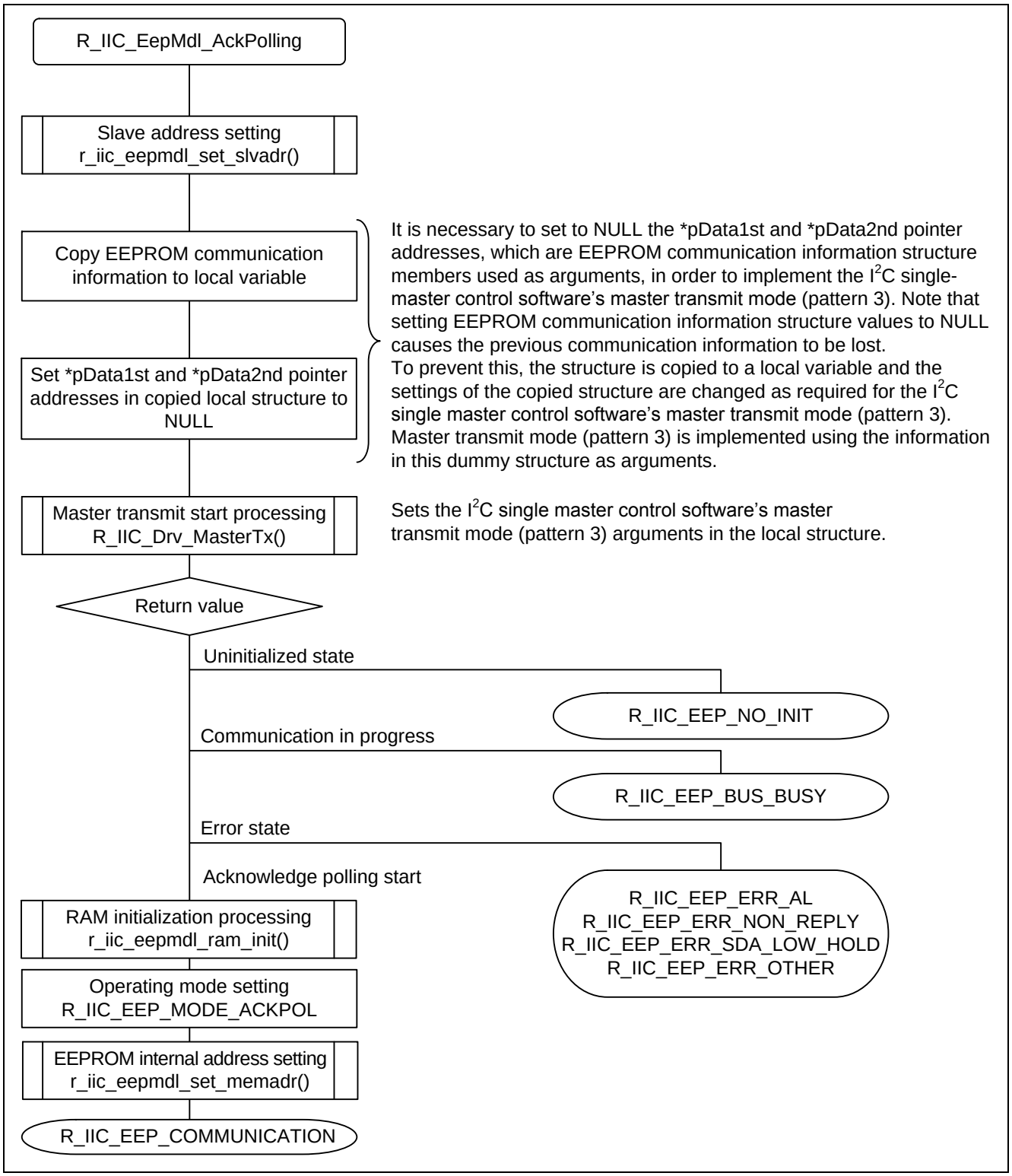


Figure 5.17 Outline of Acknowledge Polling Start Function

5.16.5 Read Start Function

R_IIC_EepMdl_Read

Outline Read start function

Header r_iic_eepmdl_api.h, r_iic_eepmdl_sub.h

Declaration error_t R_IIC_EepMdl_Read(r_iic_eepmdl_info_t FAR *pEep_Info)

Description
- Starts data read from the EEPROM.
- Use this function to enable the master (MCU) to read data from the slave (EEPROM).

Arguments r_iic_eepmdl_info_t FAR *pEep_Info ; EEPROM communication information storage pointer

Return Value

■ Return value (when R_IIC_EepMdl_Read() is called)

R_IIC_EEP_COMMUNICATION
Data read from EEPROM started.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_NO_INIT
Uninitialized state.
→ Call the EEPROM initialization function.

R_IIC_EEP_LOCK_FUNC
No processing was performed because another API was being processed.
→ Call the function after processing of the other API finishes.

R_IIC_EEP_BUS_BUSY
Communication is in progress. Processing of read from EEPROM could not start.
→ Call the EEPROM advance function to terminate communication.

R_IIC_EEP_ERR_PARAM
Parameter error.
→ Check the setting value(s).

R_IIC_EEP_ERR_AL
Arbitration lost.
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_NON_REPLY
No reply error.*[1]
→ The EEPROM recovery function can be called to perform recovery processing.

R_IIC_EEP_ERR_SDA_LOW_HOLD
SDA cannot be restored from low-hold state.
→ Check the system state to determine if the slave device is in the low-hold state, if the master device is outputting a low signal, etc.

R_IIC_EEP_ERR_OTHER
Other error occurred.
→ Check the following.
— Confirm that the EEPROM communication information structure settings are correct.
— Check to determine if an error occurred in OS control.

■ Return value (when R_IIC_EepMdl_Advance() is called after R_IIC_EepMdl_Read)

R_IIC_EEP_COMMUNICATION
Communication is in progress.
→ Call the EEPROM advance function to terminate communication.

R_IIC_ EEP_FINISH_READ
Reading of all data has finished.
→ Communication is possible by calling the start function.

See the description of the EEPROM advance function for information on the return values in case of error.

Remarks
- This function uses the I^2C single master control software's master composite mode.

- Settings must be made in **r_iic_eepmdl_info_t** to perform this processing. See 5.11.1 (1) for setting instructions.
- I^2C communication is not finished at the point of return from this function. It is necessary to call the EEPROM advance function to finish I^2C communication.

Note: 1. The no reply error definition differs depending on the MCU being controlled. See the description of the no reply error in the documentation of the I^2C single master control software for details.

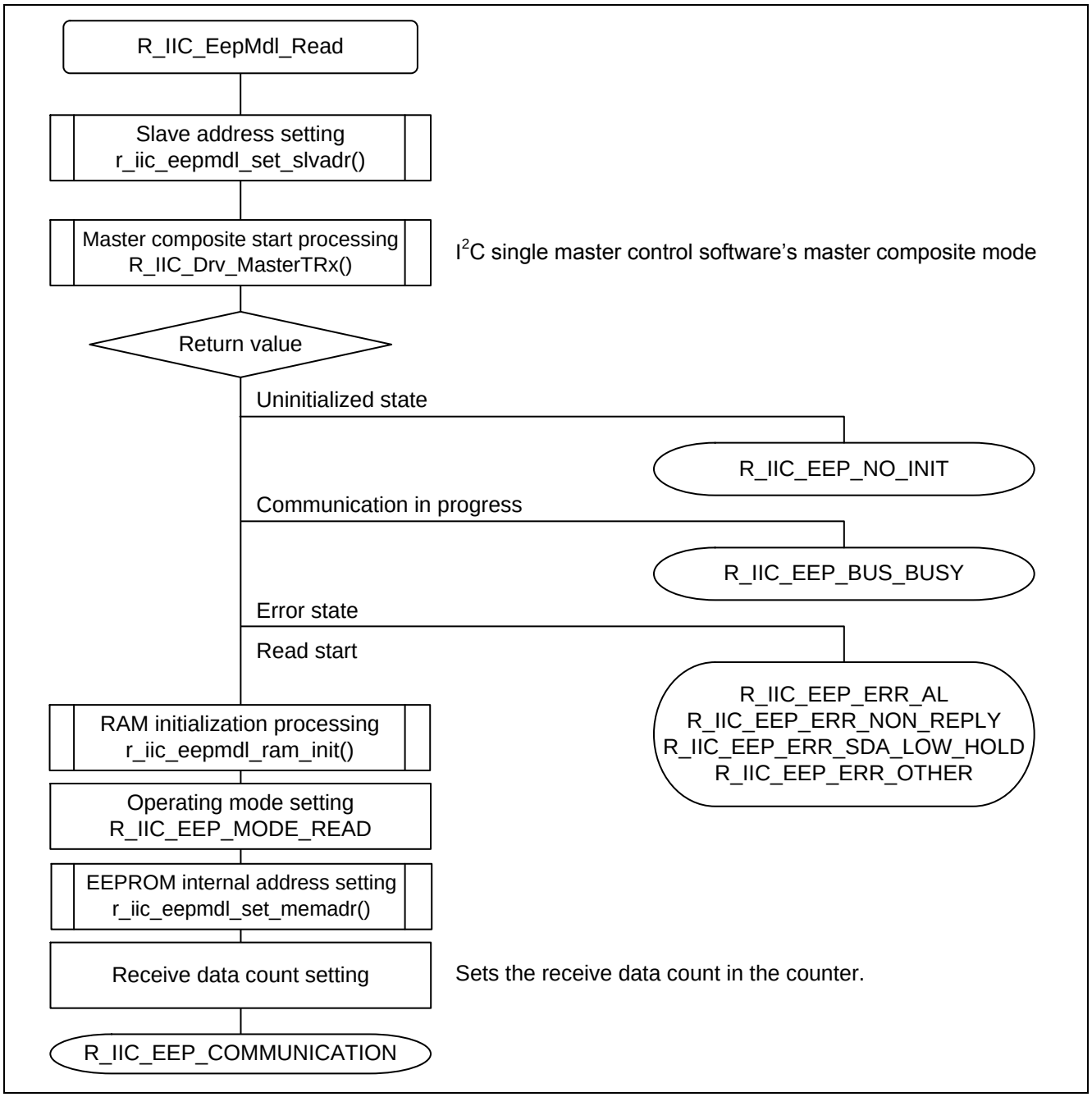


Figure 5.18 Outline of Read Start Function

5.16.6 EEPROM Advance Function

R_IIC_EepMdl_Advance

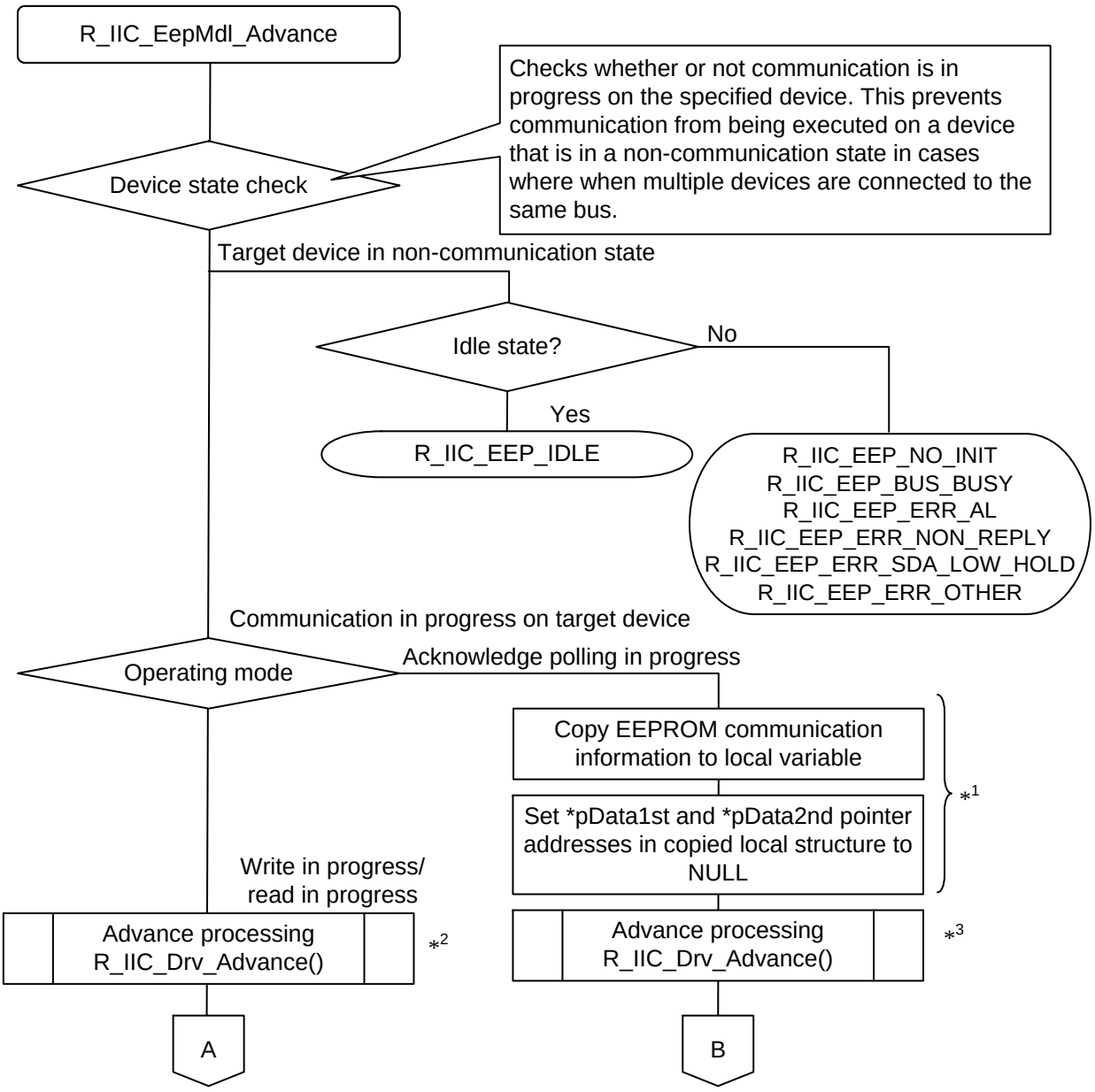
Outline	EEPROM advance function
Header	r_iic_eepmdl_api.h, r_iic_eepmdl_sub.h
Declaration	error_t R_IIC_EepMdl_Advance(r_iic_eepmdl_info_t FAR *pEep_Info)
Description	• Monitors communication and executes processing to enable communication to proceed. Returns the communication state as a return value.
Arguments	r_iic_eepmdl_info_t FAR *pEep_Info ; EEPROM communication information structure pointer
Return Value	R_IIC_EEP_COMMUNICATION Communication is in progress. → Call the EEPROM advance function to terminate communication. R_IIC_EEP_FINISH_WRITE Writing of all data completed. → The acknowledge polling start function can be called to determine if rewriting to the EEPROM has finished.*[1] R_IIC_EEP_FINISH_WRITE_AGN Page write finished. There is leftover data. → Call the write start function after rewrite finishes. At this time do not change the member information of the EEPROM communication information structure.*[2] R_IIC_EEP_FINISH_ACKPOL Rewriting of all data completed. → Communication is possible by calling the start function. R_IIC_EEP_FINISH_ACKPOL_AGN Page rewrite finished. There is leftover data. → Call the write start function. At this time do not change the member information of the EEPROM communication information structure.*[2] R_IIC_EEP_FINISH_ACKPOL_NACK Page rewrite did not finish (NACK received after transmission of slave address). → The acknowledge polling start function can be called to determine if rewriting to the EEPROM has finished.*[1] R_IIC_EEP_FINISH_READ Reading of all data completed. → Communication is possible by calling the start function. R_IIC_EEP_LOCK_FUNC No processing was performed because another API was being processed. → Call the function after processing of the other API finishes. R_IIC_EEP_BUS_BUSY Communication by another device is in progress on the same channel, so processing could not be performed. → Terminate communication by the other device. R_IIC_EEP_NO_INIT Uninitialized state. → Call the EEPROM initialization function. R_IIC_EEP_IDLE Idle state. → Communication is possible by calling the start function. R_IIC_EEP_ERR_PARAM Parameter error. → Check the setting value(s).

R_IIC_EEP_ERR_AL
Arbitration lost.
→ The EEPROM recovery function can be called to perform recovery processing.
R_IIC_EEP_ERR_NON_REPLY
No reply error.*3
→ The EEPROM recovery function can be called to perform recovery processing.
R_IIC_EEP_ERR_SDA_LOW_HOLD
SDA cannot be restored from low-hold state.
→ Check the system state to determine if the slave device is in the low-hold state, if the master device is outputting a low signal, etc.
R_IIC_EEP_ERR_OTHER
Other error occurred.
→ Check the following.
— Confirm that the EEPROM communication information structure settings are correct.
— Check to determine if an error occurred in OS control.
R_IIC_EEP_ERR_NACK
NACK received while write communication in progress.
→ Check the following.
— If the write protect pin (WP) is high, drive it low and then call the write start function. If the expected write data count does not match the data actually written, the EEPROM recovery function can be called to perform recovery processing.

Remarks

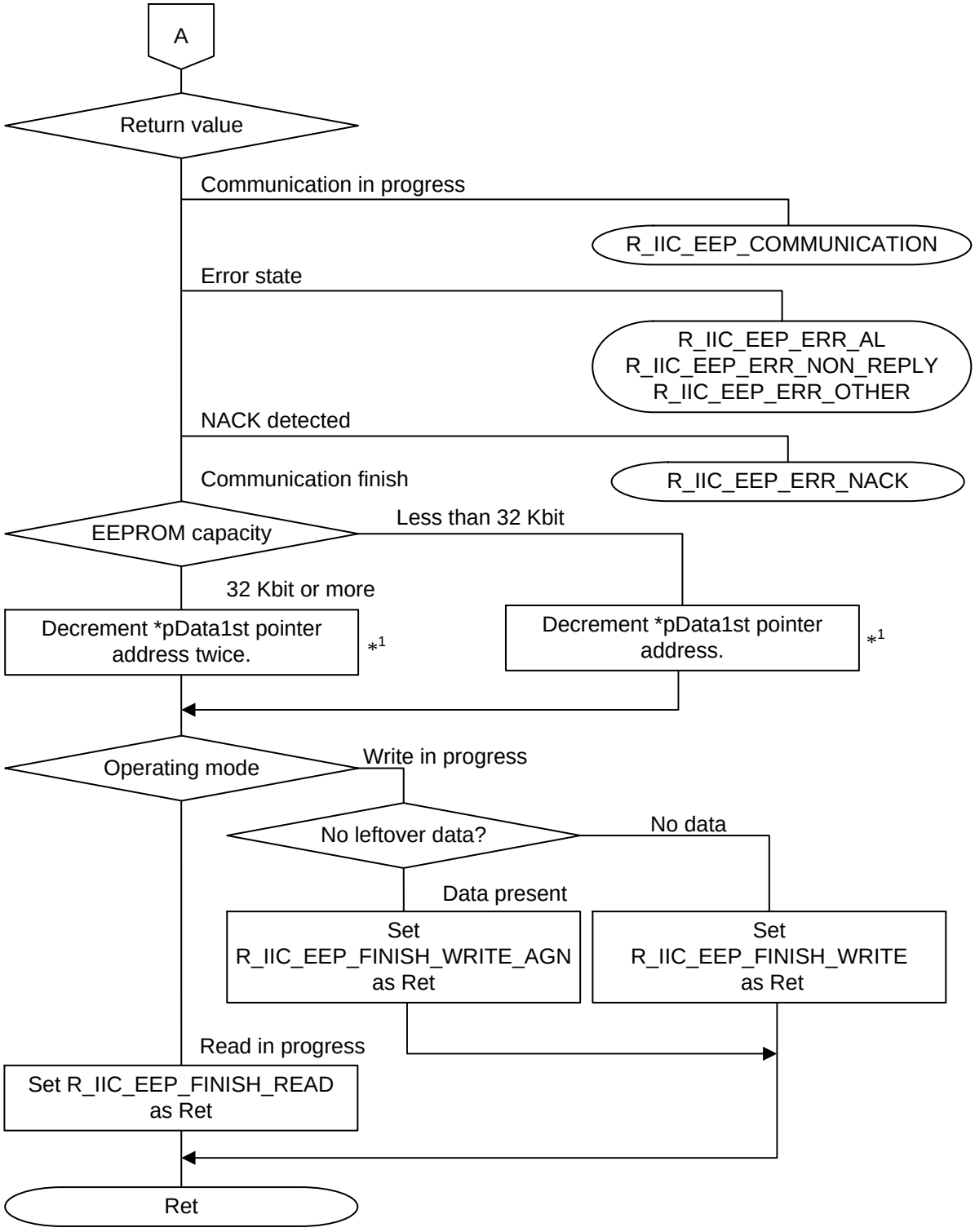
- After each start function is called, call the EEPROM advance function to finish communication.
- The EEPROM advance function's return value when communication finishes differs depending on the start function called. Check the return value against the description in this section or the EEPROM advance function return value description in the section on the specific start function.
- If an error occurs during communication, reset the data and call the start function again following recovery processing.

Notes: 1. The write or read start function can be called without determining the rewrite status. To do this, call the start function after waiting for the EEPROM rewrite to finish. Refer to the datasheet of the EEPROM device for the write cycle time.
2. The leftover data write information is stored in the members of the EEPROM communication information structure, so communication cannot proceed correctly if the member values are changed.
3. The no reply error definition differs depending on the MCU being controlled. See the description of the no reply error in the documentation of the I^2C single master control software for details.



Notes: 1. It is necessary to set to NULL the *pData1st and *pData2nd pointer addresses, which are EEPROM communication information structure members used as arguments, in order to implement the I^{2}C single master control software's master transmit mode (pattern 3). Note that setting EEPROM communication information structure values to NULL causes the previous communication information to be lost. To prevent this, the structure is copied to a local variable and the settings of the copied structure are changed as required for the I^{2}C single master control software's master transmit mode (pattern 3). Master transmit mode (pattern 3) is implemented using the information in this dummy structure as arguments.
2. Sets the EEPROM communication information structure information as arguments.
3. Sets the local structure information as arguments.

Figure 5.19 Outline of EEPROM Advance Function (1/3)



Note 1. *pData1st is a pointer to the EEPROM internal address storage buffer. The pointer address is incremented when communication finishes, so it must be returned to the position before communication started. Note that the value incremented differs according to the EEPROM capacity. If the capacity is 32 Kbits or greater a 2-byte internal address is transmitted, so the pointer address must be decremented twice. If the capacity is less than 32 Kbits, the pointer address is decremented once.

Figure 5.20 Outline of EEPROM Advance Function (2/3)

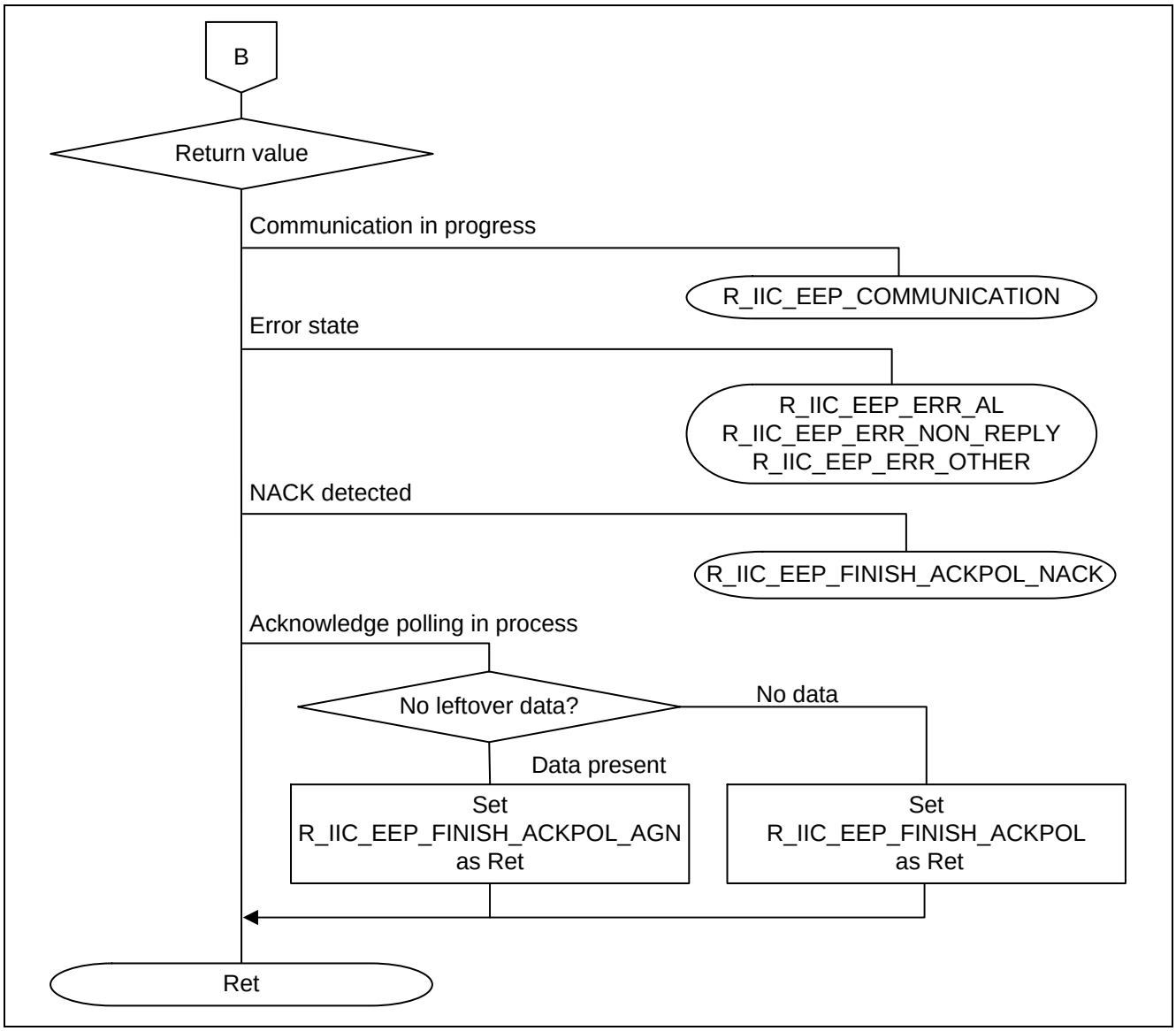


Figure 5.21 Outline of EEPROM Advance Function (3/3)

5.16.7 EEPROM Recovery Function

R_IIC_EepMdl_Recovery

Outline EEPROM recovery function

Header r_iic_eepmdl_api.h, r_iic_eepmdl_sub.h,

Declaration error_t R_IIC_EepMdl_Recovery(r_iic_eepmdl_info_t FAR *pEep_Info)

Description
- Can perform recovery processing after a communication error occurs.
- Call this function to force an initialization.
- After this function is called, a transition to the idle state occurs.

Arguments r_iic_eepmdl_info_t FAR *pEep_Info ; EEPROM communication information storage pointer

Return Value

R_IIC_EEP_IDLE
EEPROM recovery processing finished successfully and a transition to the idle state has occurred.
→ Communication is possible by calling the start function.

R_IIC_EEP_LOCK_FUNC
No processing was performed because another API was being processed.
→ Call the function after processing of the other API finishes.

R_IIC_EEP_ERR_PARAM
Parameter error.
→ Check the setting value(s).

R_IIC_EEP_ERR_AL
Arbitration lost.
→ Recovery processing can be performed by calling the EEPROM recovery function again.

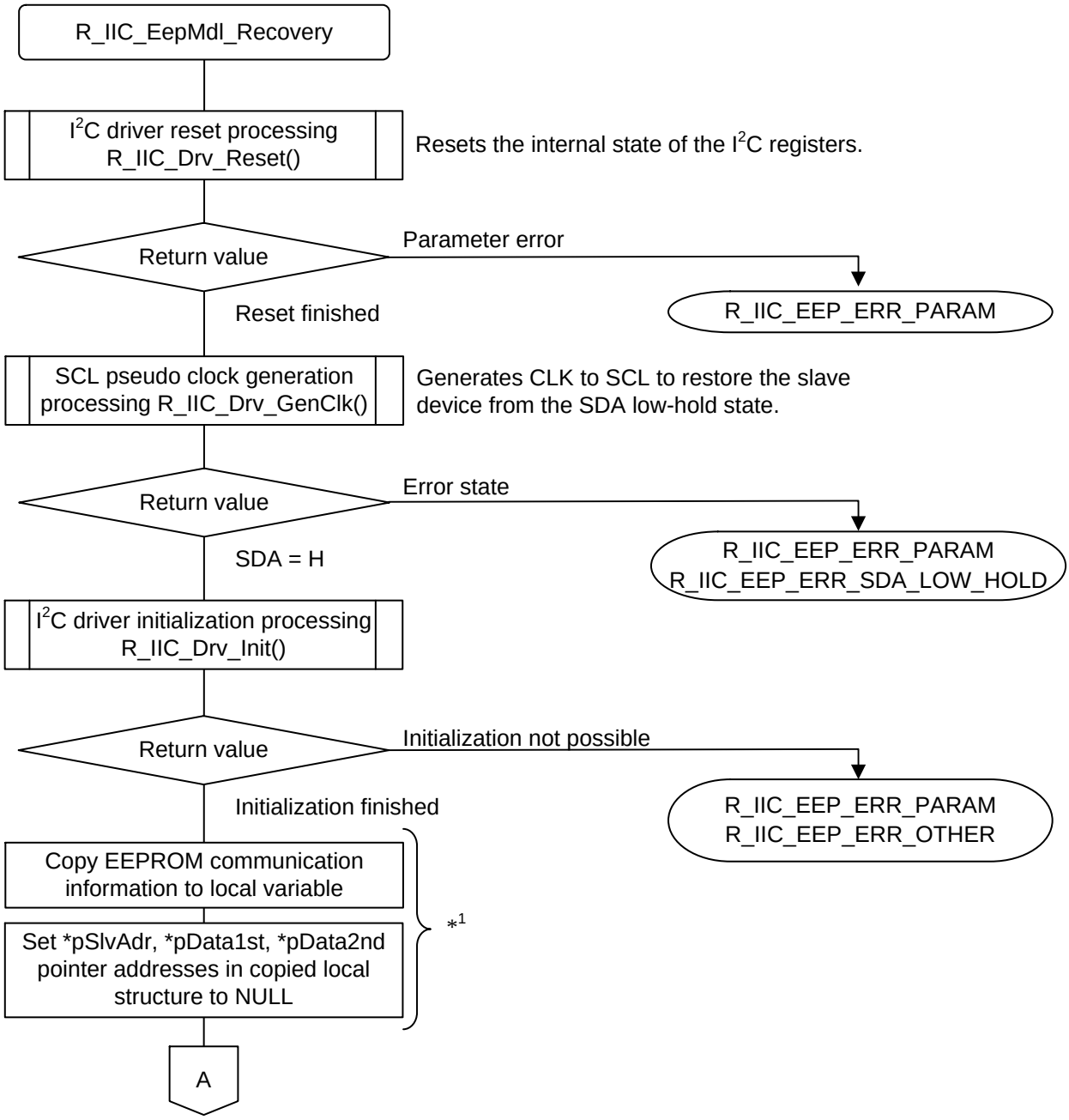
R_IIC_EEP_ERR_NON_REPLY
No reply error.*1
→ Recovery processing can be performed by calling the EEPROM recovery function again.

R_IIC_EEP_ERR_SDA_LOW_HOLD
EEPROM recovery processing was performed, but SDA did not recover from the low-hold state.
→ Check the system state to determine if the slave device is in the low-hold state, if the master device is outputting a low signal, etc.

R_IIC_EEP_ERR_OTHER
Other error occurred.
→ Check the following.
— Check the system state to determine if the slave device is not in the SCL low-hold state, if the master device is not outputting a low signal, etc.
— Confirm that the EEPROM communication information structure settings are correct.
— Check to determine if an error occurred in OS control.

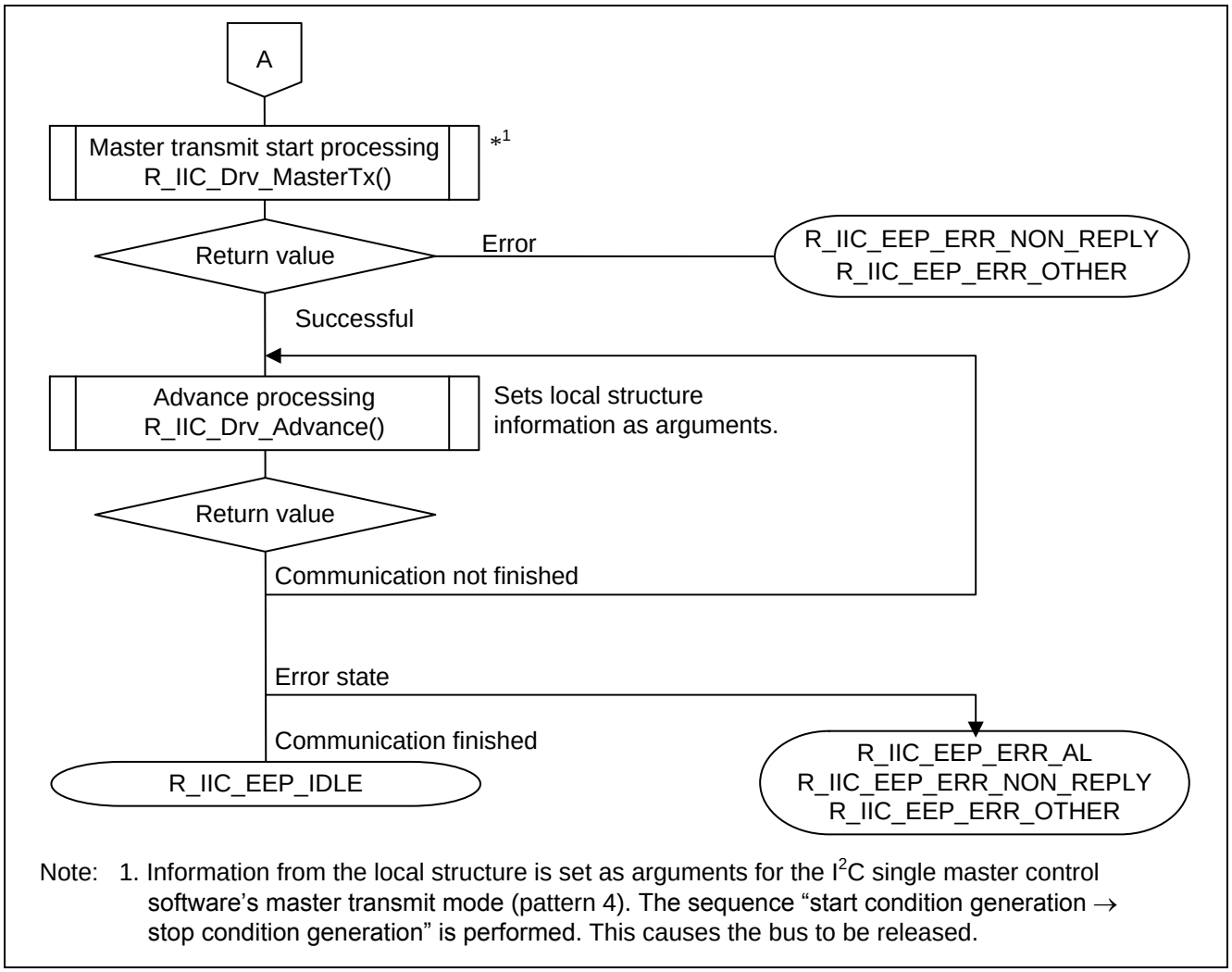
Remarks
- This function performs an I^2C internal reset.
- If SDA is in the low-hold state after the reset, a pseudo clock is generated and sent to SCL. The number of cycles generated can be set by the macro definition SCL_CLK_CNT. (See Table 5.14 in 5.10.1for information on macro definitions.)
- I^2C single master control software's master transmit mode (pattern 4) is used to generate a start condition and a stop condition, releasing the bus.
- If communication cannot be restored after executing this function, there may be a fault such as SDA being fixed to GND.

Note: 1. The no reply error definition differs depending on the MCU being controlled. See the description of the no reply error in the documentation of the I^2C single master control software for details.



Note: 1. It is necessary to set to NULL the *pSlvAdr, *pData1st, and *pData2nd pointer addresses, which are EEPROM communication information structure members used as arguments, in order to implement the I2C single master control software's master transmit mode (pattern 4). Note that setting EEPROM communication information structure values to NULL causes the previous communication information to be lost.
To prevent this, the structure is copied to a local variable and the settings of the copied structure are changed as required for the I2C single master control software's master transmit mode (pattern 4). Master transmit mode (pattern 4) is implemented using the information in this dummy structure as arguments.

Figure 5.22 Outline of EEPROM Recovery Function (1/2)



Note: 1. Information from the local structure is set as arguments for the I^2C single master control software's master transmit mode (pattern 4). The sequence "start condition generation → stop condition generation" is performed. This causes the bus to be released.

Figure 5.23 Outline of EEPROM Recovery Function (2/2)

6. Application Example

6.1 r_iic_eepmdl_api.h

Example settings when using the software are described below.

The portion of each file where the settings are entered are designated with the comment line /** SET **/.

(1) **Selection of MCU Used**

Specify the MCU to be used.

```
/*-------------------------------------------------------------------------*/
/*   Select to use MCU Type.                                               */
/*-------------------------------------------------------------------------*/
#define MCU_RL78
/* #define MCU_RX       */
```

(2) **Definition of RAM Area Accessed**

This setting defines the RAM area to be used when using the M16C.

The RAM is used to improve the efficiency of standard functions and some processing tasks.

This setting has no effect when an MCU other than the M16C is used.

In the example below, the FAR area is used.

```
/*-------------------------------------------------------------------------*/
/* If using a M16C, define the RAM area to be accessed by the user process.*/
/* Please choose one of definitions.                                       */
/* Efficient operations for standard functions and processes are applied.  */
/*-------------------------------------------------------------------------*/
#if defined(MCU_M16C)
#define R_IIC_FAR
/* #define R_IIC_NEAR */
#endif  /* #if defined(M16C) */
```

(3) **SCL Pseudo Clock Counter Value Definition**

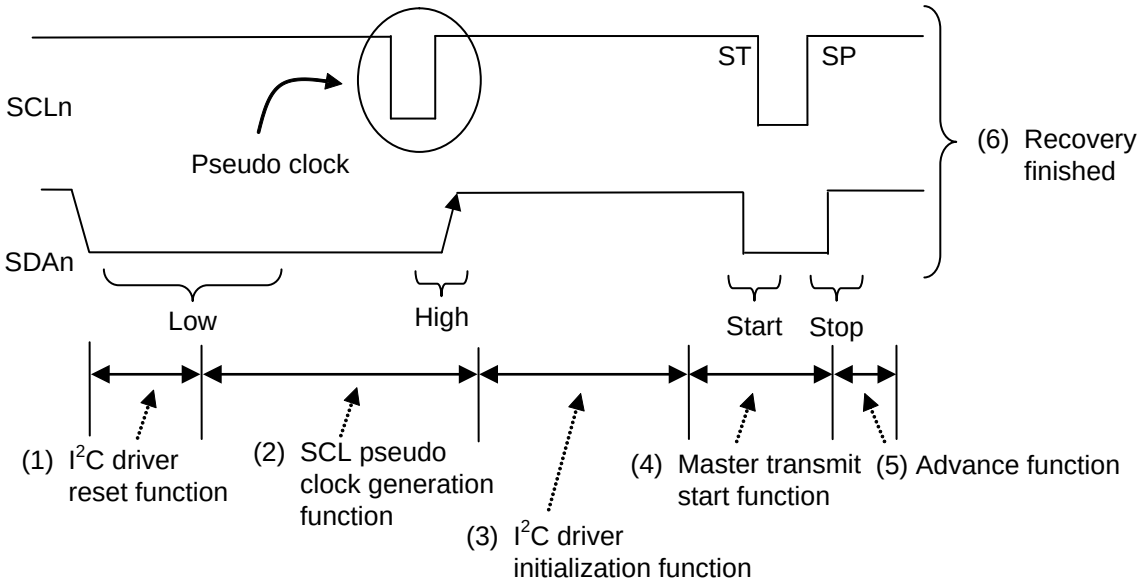
When SDA is in the low-hold state, the EEPROM recovery function generates a pseudo clock and sends it to SCL. Use this setting to define the number of clock cycles.

The counter value is set to 9 in the sample code because an I^2C communication unit of nine clock cycles is common.

```
/*-------------------------------------------------------------------------*/
/*   Counter                                                               */
/*-------------------------------------------------------------------------*/
#define  SCL_CLK_CNT (uint8_t)(9)  /* Clock counter to SCL when recovers processing */
```

6.2 EEPROM Recovery Function

Figure 6.1 illustrates the EEPROM recovery processing to restore communication when SDA or SCL are in the low-hold state because of an unexpected momentary power interruption or noise while communication is in progress. The device transitions to the idle state when processing finishes.



(1) Performs an I^2C internal reset.
(2) Generates a pseudo clock and sends it to SCL, releasing the slave device from the low-hold state.
(3) Initialization is performed for master transmit (pattern 4).
(4) Starts master transmit (pattern 4). A start condition is generated, then a stop condition is generated.
(5) Ends communication by using the advance function.
(6) The device enters the idle state. After this, communication is possible by calling a start function.

Figure 6.1 Outline of EEPROM Recovery Function Operation

7. Usage Notes

7.1 Notes on Incorporation

Make sure to include the following files when incorporating the sample code into a user program.

- r_iic_eepmdl_api.h
- r_iic_eepmdl_sub.h
- r_iic_drv_api.h
- r_iic_drv_sub.h
- r_iic_drv_sfr.h
- r_iic_drv_int.h
- r_iic_drv_os.h

7.2 Page Size Setting

It is necessary to specify the page size in the EEPROM communication information structure member **PageSize**. Use the page size stipulated for the EEPROM device used when making this setting. Depending on the EEPROM type name, devices with the same capacity may have different page sizes. The sample code will perform communication even if a setting different from the actual page size of the EEPROM device is used. Note that using a setting value greater than the page size stipulated for the EEPROM device will result in roll-over during writes.

7.3 Structure Handling when Calling Acknowledge Polling Start Function

The I^2C single master control software's master transmit pattern 3 is used for EEPROM acknowledge polling, and this requires setting the EEPROM communication information structure members *pData1st and *pData2nd, indicating pointer addresses, to NULL.

After a write operation finishes, it is necessary to set information from the EEPROM communication information structure of the communication currently in progress as arguments when calling the acknowledge polling start function or EEPROM advance function. If the *pData1st and *pData2nd structure members indicating pointer addresses are set to NULL while these functions are being processed, the communication information will be lost.

To prevent this, the acknowledge polling start function and EEPROM advance function each copy the structure with the argument information to a local variable and change the settings of the copied structure to the I^2C single master control software function arguments. This prevents the loss of communication information.

7.4 Structure Handling when Calling EEPROM Recovery Function

Like the acknowledge polling start function, the EEPROM recovery function uses the I^2C single master control software's master transmit pattern 4. This requires setting the EEPROM communication information structure members *pSlvAdr, *pData1st, and *pData2nd, indicating pointer addresses, to NULL. The same method described in section 7.3 is used by the EEPROM recovery function to perform processing with a copy of the structure.

7.5 Communication after Calling EEPROM Recovery Function

When restarting communication after calling the EEPROM recovery function, it is possible that the previous communication information may have been lost. Therefore, redo the communication procedure from the start.

7.6 Processing of EEPROM Advance Function within Interrupt Handler and OS Control

Processing of the EEPROM advance function within an interrupt handler and OS control* are unverified. When using these capabilities, careful evaluation should be performed and modifications applied if necessary.

Note: * The support for OS control in the sample code assumes µITRON 4.0.

7.7 Notes on Connection of Multiple Devices to Same Bus

The number of serial EEPROM devices that can be connected as slave devices differs depending on the capacity. Make sure to keep this in mind when connecting multiple slave devices to a single channel. See Table 5.2 for details on the number of slave devices that can be connected.

7.8 Considerations at Compile-time

Case compiled with the CC-RL compiler, Output the warning "W0520111: Statement is unreachable."

This is a warning message that does not run the break statement. It does not affect behavior. Ignore and no problem.

Website and Support

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

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

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.02	Oct 31, 2014	—	First edition issued
1.03	Mar 31, 2016	6	Changed the following title to section 2. (1) RL78/G14 IICA Integrated Development Environment CS+ for CA,CX (Compiler: CA78K0R) Added the following title to section 2. (2) RL78/G14 IICA Integrated Development Environment CS+ for CC (Compiler: CC-RL)
		31	Changed the following title to section 5.8.1 (1) RL78/G14 IICA Integrated Development Environment CS+ for CA,CX (Compiler: CA78K0R) Added the following title to section 5.8.1 (2) RL78/G14 IICA Integrated Development Environment CS+ for CC (Compiler: CC-RL)
		35	Section 5.9 File Structure Changed Application Note Number. Changed Folder names.
		65	Added section 7.8.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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 Microprocessing unit or Microcontroller unit products 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.

RENESAS

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, HALII 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