RL78/G14, RL78/G1C, RL76/L12, RL78/L13, RL78/L1C Group

R01AN1074EJ0104
Rev.1.04
Mar 31, 2016

I²C Bus Single Master Control Software Using IICA Serial Interface

Introduction

This application note describes I^{2}C bus single master control using the RL78/G14, RL78/G1C, RL76/L12, RL78/L13, RL78/L1C Group IICA serial interface, sample code that implements that control, and use of the sample code.

In this application note, the software used to control the slave device is referred to as the upper layer and the software that implements I^{2}C single master basic protocol control as the lower layer. Slave devices are controlled by combining the protocols provided by the upper and lower layers.

This sample code implements the lower layer used for I^{2}C single master control. The user should acquire or implement software corresponding to the upper level for slave device control.

Software in the upper-level layer for controlling the slave device is separately available, so please obtain this from the following URL as well. When the slave device control software is added, update of this application note may not be in time. Refer to the following URL for the combination information on the latest slave device control software.

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

Target Device

Microcontroller:

RL78/G1x series : RL78/G14, RL78/G1C group

RL78/L1x series : RL78/L12, RL78/L13, RL78/L1C group

Device used for verifying operation: Renesas Electronics R1EX24xxx Series I^{2}C Serial EEPROM.

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

Note that the term “RL78 Family microcontroller” is used in this document for ease of description since the target devices come from multiple groups.

Contents

1. Specifications 4

2. Operation Confirmation Conditions 6

3. Reference Application Note 12

4. Peripheral Functions 12

5. Hardware 13
5.1 Pins Used 13
5.2 Reference Circuit 13
5.3 Controlling Multiple Slave Devices 14
5.4 Maximum Transfer Speed 14

6. Software 15
6.1 Software Structure 15
6.2 Operation Overview 16
6.2.1 Master Transmission 16
6.2.2 Master Reception 18
6.2.3 Master Composite 19
6.3 Software Operation 20
6.4 Software Operating Sequence 21
6.5 Implementation of Slave Device Control 22
6.6 Communication Implementation 23
6.6.1 States During Control 23
6.6.2 Events During Control 23
6.6.3 Protocol State Transitions 24
6.6.4 Protocol State Transition Table 28
6.6.5 Protocol State Transition Registered Functions 28
6.6.6 Processing at Protocol State Transitions 29
6.7 Interrupt Generation Timing 30
6.7.1 Master Transmission 30
6.7.2 Master Reception 31
6.7.3 Master Composite 31
6.8 Callback Function 32
6.9 Relationship of Data Buffers and Transmit/Receive Data 32
6.10 Required Memory Sizes 33
6.11 File Structure 36
6.12 Constants 37
6.12.1 Definitions 37
6.13 Structures and Unions 39
6.13.1 I²C Communication Information Structure 39

6.13.2 Internal Information Management Structure 41
6.14 Enumerated Types 42
6.15 Variables 43
6.16 Functions 44
6.17 State Transition Diagram 45
6.17.1 Error State Definitions 46
6.17.2 Flag States at State Transitions 47
6.18 Function Specifications 48
6.18.1 Common Processing for These Functions 48
6.18.2 I²C Driver Initialization Function 53
6.18.3 Master Transmission Start Function 57
6.18.4 Master Reception Start Function 61
6.18.5 Master Composite Start Function 65
6.18.6 Advance Function 69
6.18.7 SCL Pseudo Clock Generation Function 74
6.18.8 I²C Driver Reset Function 76

7. Application Example 78
7.1 r_iic_drv_api.h 78
7.2 r_iic_drv_sfr.h 80
7.3 r_iic_drv_int.c 82
7.3.1 Integrated Development Environment CS+ for CA,CX (formerly CubeSuite+) 82
7.3.2 Integrated Development Environment CS+ for CC 82
7.3.3 Integrated Development Environment IAR Embedded Workbench 82
7.4 r_iic_drv_sfr.c 83
7.5 r_iic_drv_os.c 83
7.6 Recovery Processing Example 84

8. Usage Notes 85
8.1 Notes on Embedding 85
8.2 Notes on Initialization 85
8.3 Notes on the Channel State Flag and Device State Flag 85
8.4 Control Methods for Multiple Slave Devices on the Same Channel 85
8.5 Performing Advance Function Processing from Within an Interrupt Under OS Control 85
8.6 Transfer Rate Setting 85
8.7 Notes On Setting The #define Definitions of IICAx_ENABLE and MAX_IIC_CH_NUM 86
8.8 Defining the Interrupt Function #pragma interrupt 86
8.9 Notes on User API Calls 86
8.10 About Warnings of Duplicate of Type Declaration 86
8.11 Considerations at Compile-time 86

1. Specifications

This sample code performs I²C bus single master control using the RL78 Family microcontroller's IICA serial interface. The user should acquire or implement software corresponding to the upper layer for slave device control.

The following table lists the used peripheral functions and their uses and Figure 1-1 shows a usage example.

The following provides an overview of the functions provided by this software.

- This sample code is an I²C bus single master device driver that uses the RL78 Family microcontroller as the master device using its IICA serial interface.
- This sample code implements the protocols in the I²C-bus specification. It supports master transmission, master reception, and master composite (master transmission → master reception) operation.
- Four transmission patterns can be set up for master transmission. Table 1-1 lists the operating patterns.
- The sample code supports multiple channels. Simultaneous communication using multiple channels is possible.
- Multiple slave devices with different type name can be controlled on a channel bus. 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.
- Communication is implemented by functions (start functions) that start various protocol control operations and the function (the advance function) that monitors communication and advances the processing. The communication state can be determined from the return values from the advance function.
- The start functions perform the operations from start condition generation through slave address transmission. The operations following that until the stop condition is generated are performed by calling the advance function to perform the processing forward.
- Interrupts are generated on completion of slave address transmission, data transmission, data reception, and stop condition generation.
- The communication rate can be set by the user. (Supported rates: up to 400 kHz (max)) However, if multiple devices are connected on the same channel, the communication rate must be set to match that of the slowest device.
- If communication is stopped by the influence of noise or other issues (in cases where an interrupt is not generated), an error can be returned from the advance function. If the number of advance function calls exceeds the limit, the sample code determines that communication has stopped due to an abnormal situation and a "no response error" is returned. This upper limit can be set by the user.
- If a NACK error occurs, a stop condition is occurred.
- The sample code provides SCL clock generation processing. If a synchronization discrepancy occurs between the master and slave due to noise or other problem and the I²C bus goes to the SDA = low hold state, the SCL pseudo clock generation function can be called to force the slave device internal state to normal and terminate.
- This sample code only supports communication between 7-bit address devices. Special addresses (e.g. general call addresses) are not supported.

Table 1-1 Master Communication Operation Patterns

	ST Generation	Slave Address Transmission	First Data Transmission	Second Data Transmission	SP Generation
Pattern 1	○	○	○	○	○
Pattern 2	○	○	—	○	○
Pattern 3	○	○	—	—	○
Pattern 4	○	—	—	—	○

Legend:
ST: Start condition
SP: Stop condition

Table 1-2 Peripheral Function and Its Application

Peripheral Function	Application
IICA	IICA Serial interface One channel (required)

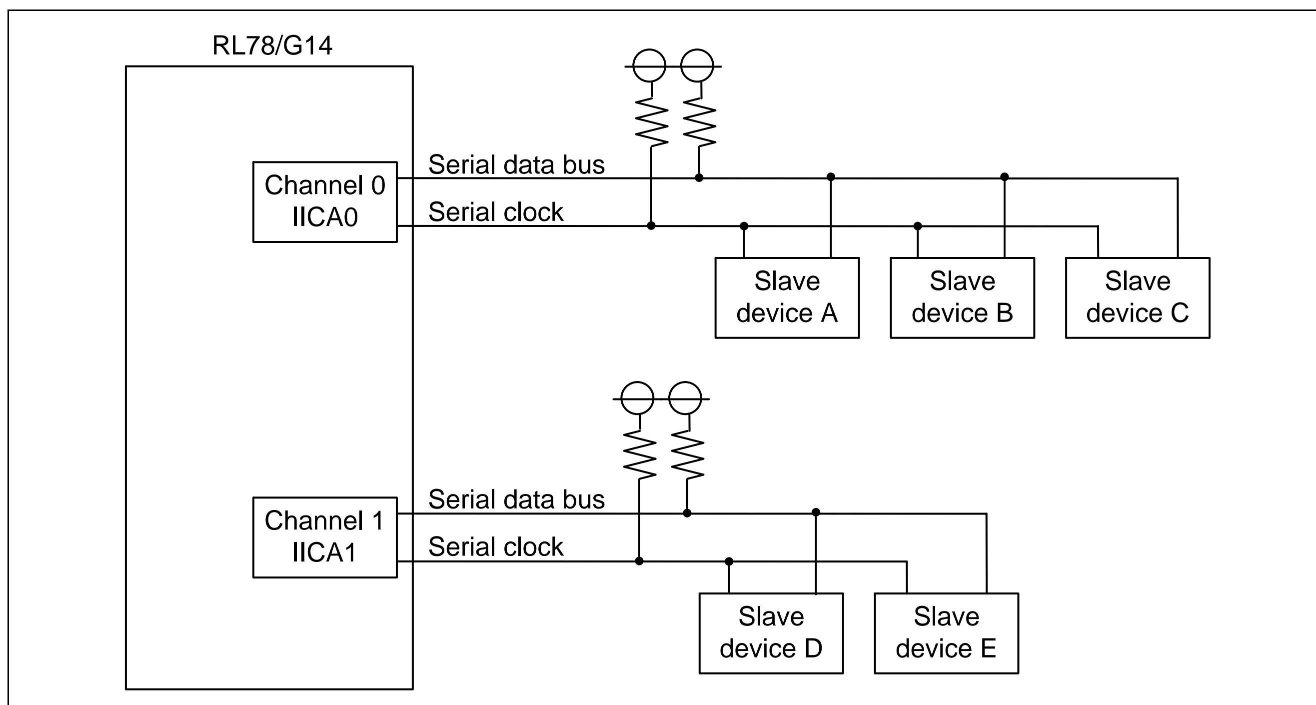


Figure 1-1 Usage Example

2. Operation Confirmation Conditions

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

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

Table 2-1 Operation Confirmation Conditions

Item	Contents
MCU used for evaluation	RL78/G14 group (program ROM: 256 KB, RAM: 24 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.03
Software used	RX Family, RL78 Family Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), Ver. 1.01
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
MCU used for evaluation	RL78/G14 group (program ROM: 256 KB, RAM: 24 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.03
Software used	RX Family, RL78 Family Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), Ver. 1.01
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
MCU used for evaluation	RL78/G14 group (program ROM: 256 KB, RAM: 24 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.00
Board used	Renesas Starter Kit for RL78/G14

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

Table 2-4 Operation Confirmation Conditions

Item	Contents
MCU used for evaluation	RL78/G1C group (program ROM: 32 KB, RAM: 5.5 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
Board used	Renesas 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
MCU used for evaluation	RL78/G1C group (program ROM: 32 KB, RAM: 5.5 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
Board used	Renesas RL78/G1C Target Board QB-R5F10JGC-TB

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

Table 2-6 Operation Confirmation Conditions

Item	Contents
MCU used for evaluation	RL78/L12 group (program ROM: 32 KB, RAM: 1.5 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
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
MCU used for evaluation	RL78/L12 group (program ROM: 32 KB, RAM: 1.5 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
Board used	Renesas Starter Kit for RL78/L12

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

Table 2-8 Operation Confirmation Conditions

Item	Contents
MCU used for evaluation	RL78/L13 group (program ROM: 128 KB, RAM: 8 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
Board used	Renesas Starter Kit for RL78/L13

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

Table 2-9 Operation Confirmation Conditions

Item	Contents
MCU used for evaluation	RL78/L12 group (program ROM:128 KB, RAM: 8 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
Board used	Renesas Starter Kit for RL78/L13

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

Table 2-10 Operation Confirmation Conditions

Item	Contents
MCU used for evaluation	RL78/L1C group (program ROM: 256 KB, RAM: 16 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
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
MCU used for evaluation	RL78/L1C group (program ROM:256 KB, RAM: 16 KB)
Memory used for evaluation	Renesas Electronics R1EX24xxx/HN58X24xxx Series I²C Serial EEPROM
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. 1.02
Software used	Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ), ver. 1.01
Board used	Renesas Starter Kit for RL78/L1C

3. Reference Application Note

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

- Renesas R1EX24xxx Series Serial EEPROM Control Software (R01AN1075EJ)

4. Peripheral Functions

The RL78 Family microcontrollers provide two I^2C bus control peripheral functions: the IICA serial interface and the serial array unit simplified I^2C bus module.

This application note uses the IICA serial interface.

5. Hardware

5.1 Pins Used

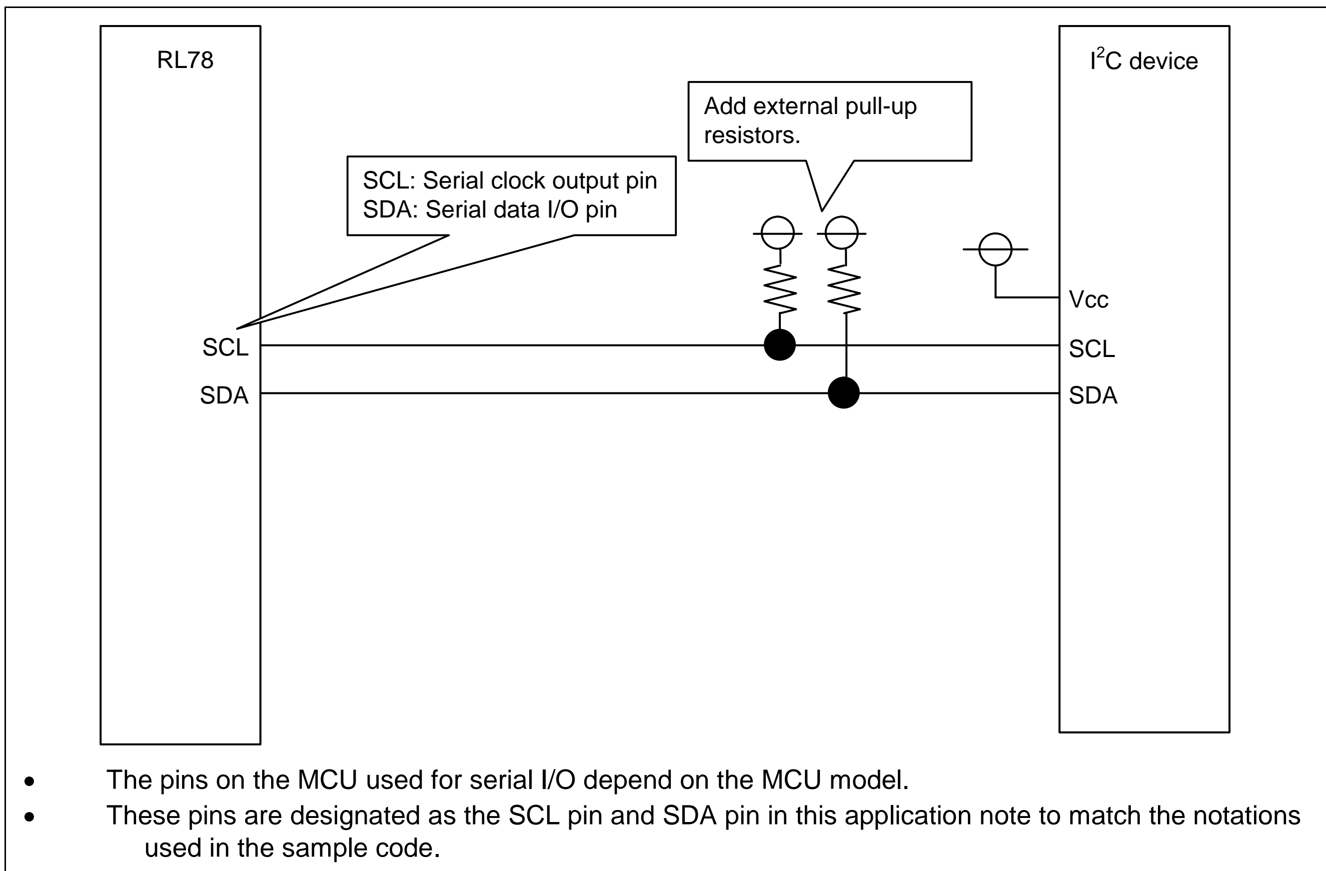
The following table lists the Pins Used and Their Functions.

Table 5-1 Pins Used and Their Functions

Pin Name	I/O	Description
SCLA (SCL in Figure 5-1)	**Output**	Serial clock output
SDAA (SDA in Figure 5-1)	**I/O**	Serial data I/O

5.2 Reference Circuit

The following figure 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 5-1 Connection Between RL78 IICA Serial Interface and I²C Slave Device

5.3 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 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.

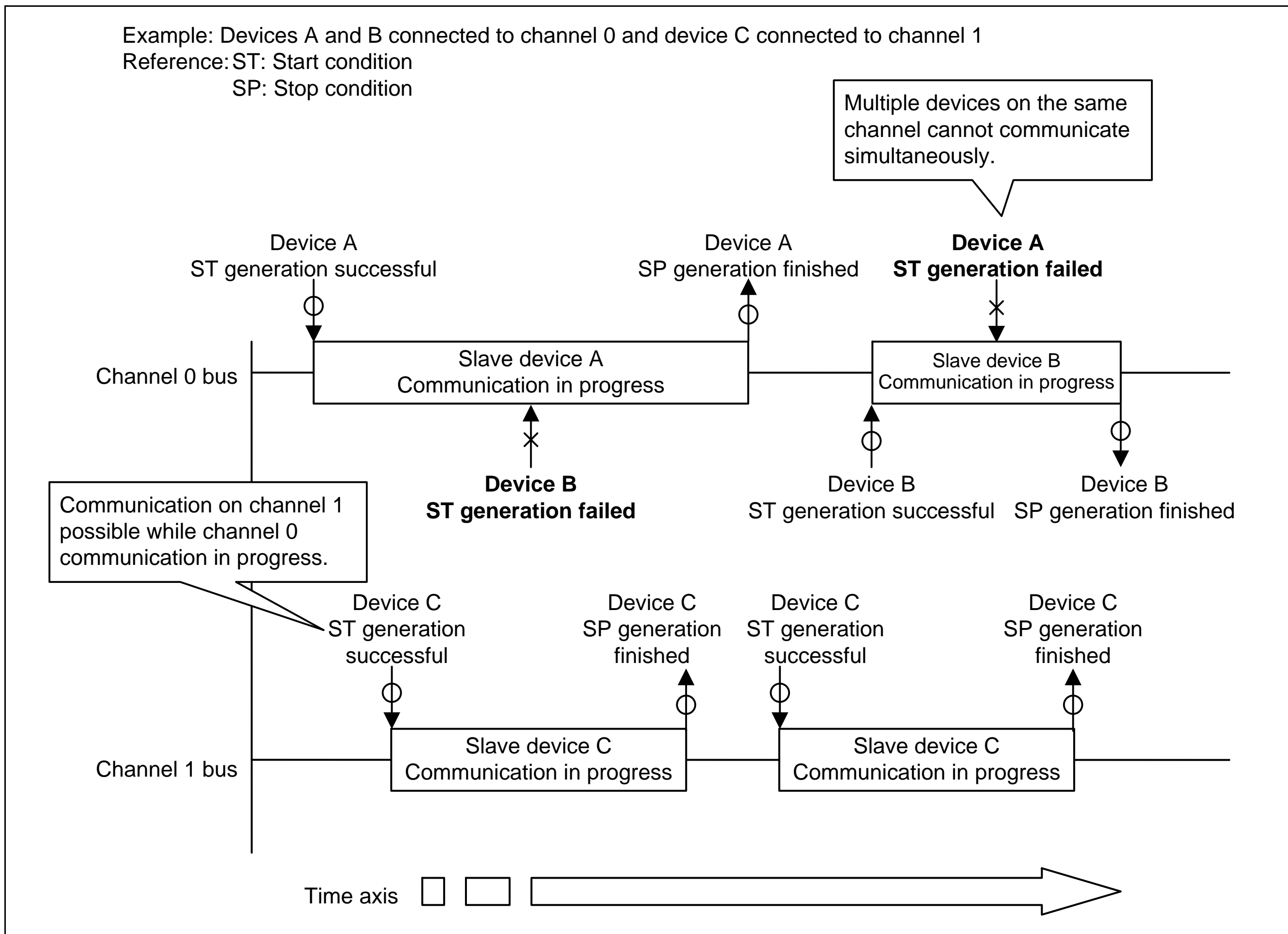


Figure 5-2 Example of Control of Multiple Slave Devices

5.4 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 5-2 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

6. Software

6.1 Software Structure

This sample code takes the software used to control slave devices as the upper layer and the software that implements I²C bus single master basic protocol control to be the lower layer. The upper layer combines protocols provided by the lower layer to control slave devices.

This sample code is positioned as lower layer used for I²C bus single master control.

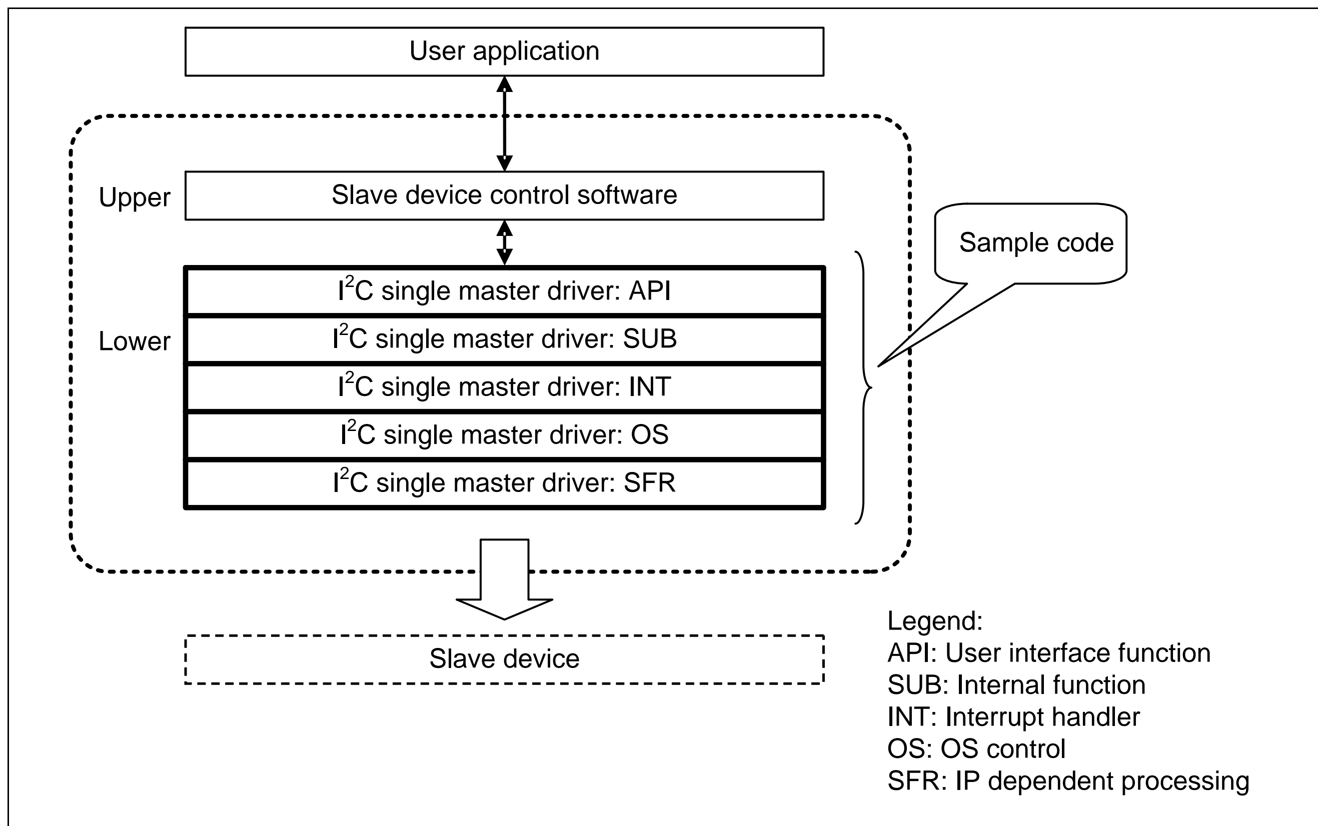


Figure 6-1 Software Structure

6.2 Operation Overview

This sample code implements I²C bus single master control using the RL78 microcontroller's IICA serial interface.

In particular, it implements the following single master protocols.

Table 6-1 Control Protocols

No.	Control Protocol	Outline
1	Master transmission	Transfers data from the master (microcontroller) to the slave device. There are four transmission patterns that can be used.
2	Master reception	The master (microcontroller) receives data from the slave device.
3	Master composite	After master transmission, a master reception operation is performed.

6.2.1 Master Transmission

There are four transmission patterns that can be used for master transmission. The function can be selected by the method used to set up the I²C communication information structure, which manages the communication information. See section 6.13.1, Communication information structure, for details on setting up this structure.

(1) **Pattern 1**

Data is transferred from the master (microcontroller) to the slave device.

First, a start condition (ST) is generated and then the slave device address is transmitted. During this transmission, the 8th bit is the transfer direction specification bit and a 0 (write) is transmitted for data transmission. Next, the first data is transmitted. The first data is used when there is data to be transmitted in advance before performing the data transmission. For example, if the slave device is an EEPROM, the EEPROM internal address can be transmitted. Next, the second data is transmitted. The second data is the data to be written to the slave device. When a data transmission has been started and all data transmission has completed, a stop condition (SP) is generated, releasing the bus.

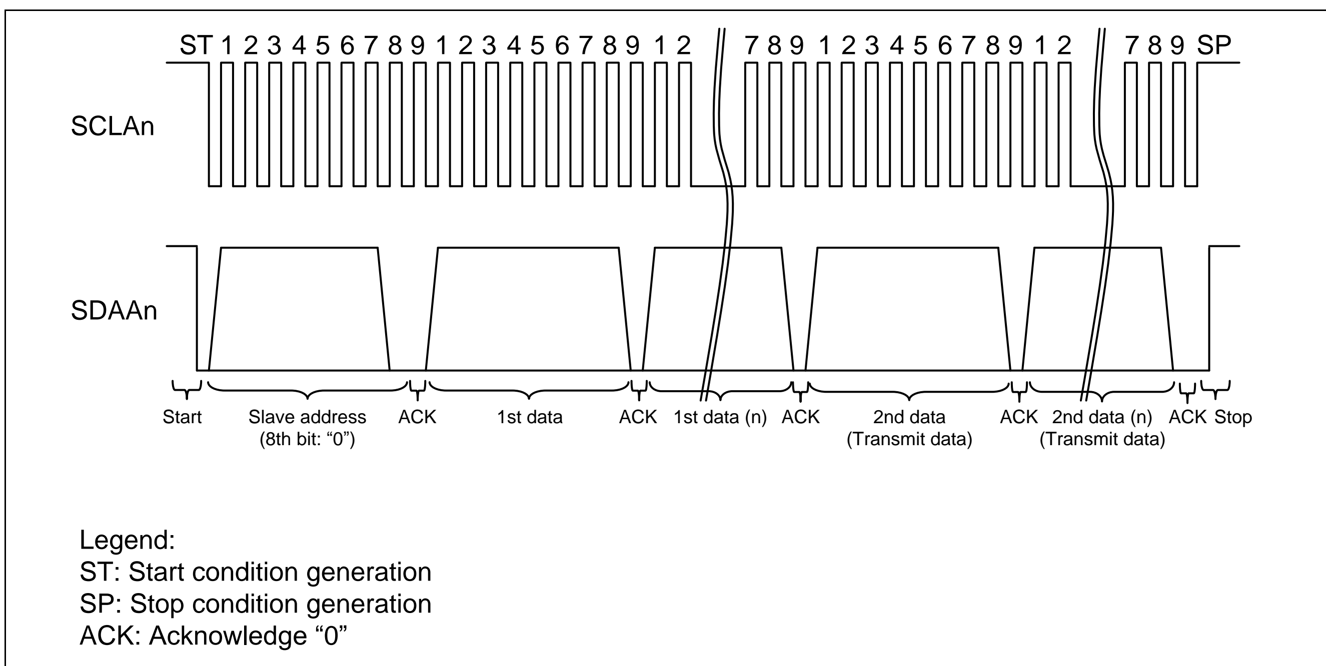


Figure 6-2 Master Transmission (Pattern 1) Signals

(2) **Pattern 2**

Data is transferred from the master (microcontroller) to the slave device. However, the first data is not transferred.

Operation from start condition (ST) generation through slave device address transmission is the same as for pattern 1. However, after that the second data is transferred without sending the first data. When all data transmission has completed, a stop condition (SP) is generated, releasing the bus.

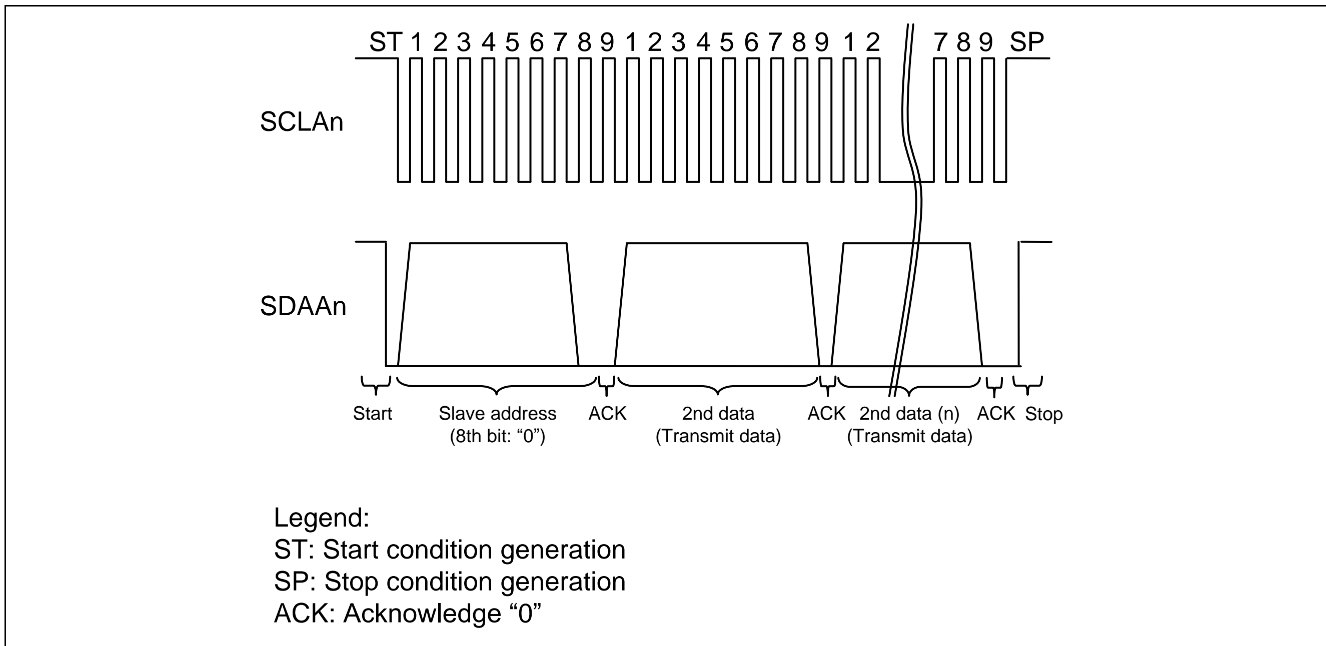


Figure 6-3 Master Transmission (Pattern 2) Signals

(3) **Pattern 3**

Operation from start condition (ST) generation through slave device address transmission is the same as in normal operation. In cases where neither the first data nor the second data are set up, however, a stop condition (SP) is generated releasing the bus without transferring any data.

This pattern is useful for detecting connected devices or when performing acknowledge polling to verify the EEPROM rewriting state.

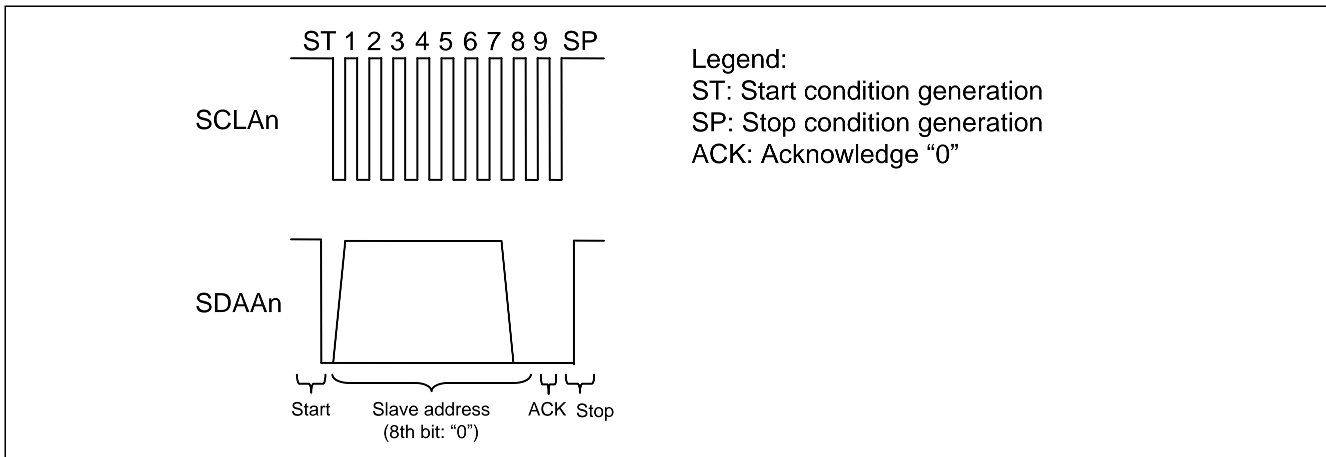


Figure 6-4 Master Transmission (Pattern 3) Signals

(4) **Pattern 4**

In this pattern, after a start condition (ST) is generated, a stop condition (SP) is generated and released the bus without transmitting the slave address, first data, or second data when those data are not set up.

This pattern is useful for just releasing the bus.

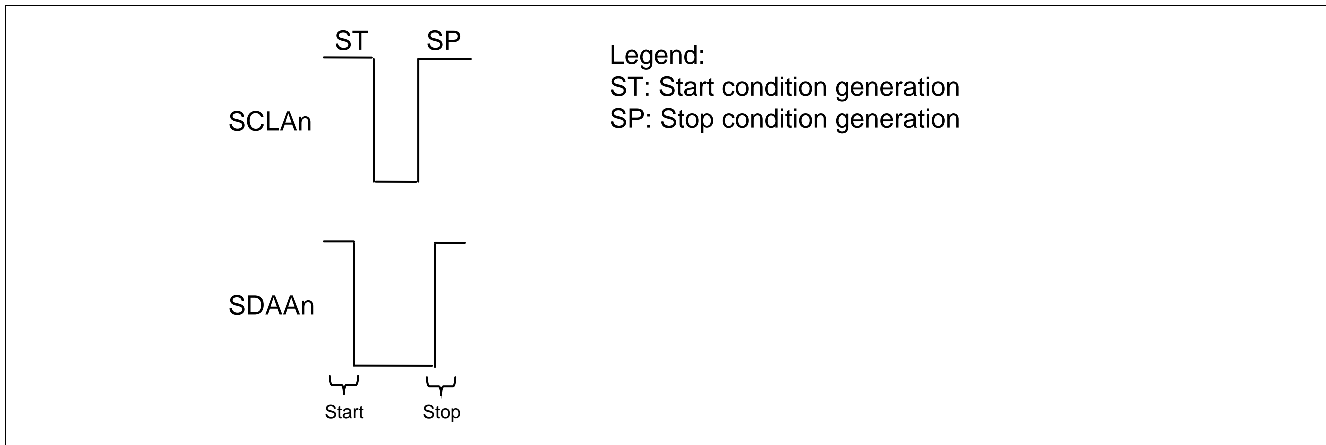


Figure 6-5 Master Transmission (Pattern 4) Signals

6.2.2 Master Reception

In master reception, the master (microcontroller) receives data from a slave device.

Here a start condition (ST) is generated and then the slave device address is transmitted. Since the 8th bit at this time is the transfer direction specification bit, a 1 (read) is transmitted when this data is transmitted. Next, data reception starts. Although an ACK is transmitted after each single byte of data is received during reception, a NACK is transmitted only after the last data to notify the slave device that reception processing has terminated. When all the data has been received, a stop condition (SP) is generated, releasing the bus.

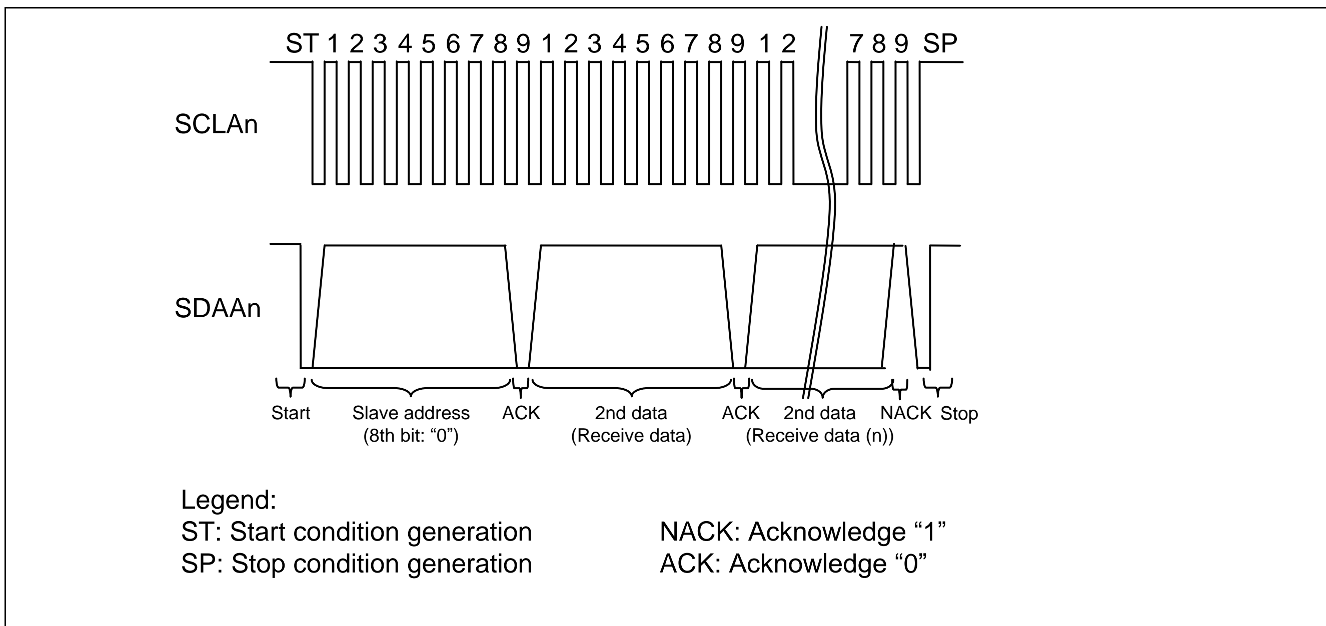


Figure 6-6 Master Reception Signals

6.2.3 Master Composite

In this mode, data is first transmitted from the master (microcontroller) to the slave device (master transmission). After this transmission completes, a restart condition is generated, the transfer direction is changed to 1 (read) and the master receives data from the slave device (master reception).

First, a start condition (ST) is generated and then the slave device address is transmitted. During this transmission, the 8th bit is the transfer direction specification bit and a 0 (write) is transmitted for data transmission. When the data transmission completes, a restart condition (RST) is generated and the slave address is transmitted. At this time, a 1 (read) is transmitted as the transfer direction specification bit. Next, data reception starts. Although an ACK is transmitted after each single byte of data is received during reception, a NACK is transmitted only after the last data to notify the slave device that reception processing has terminated. When all the data has been received, a stop condition (SP) is generated, releasing the bus.

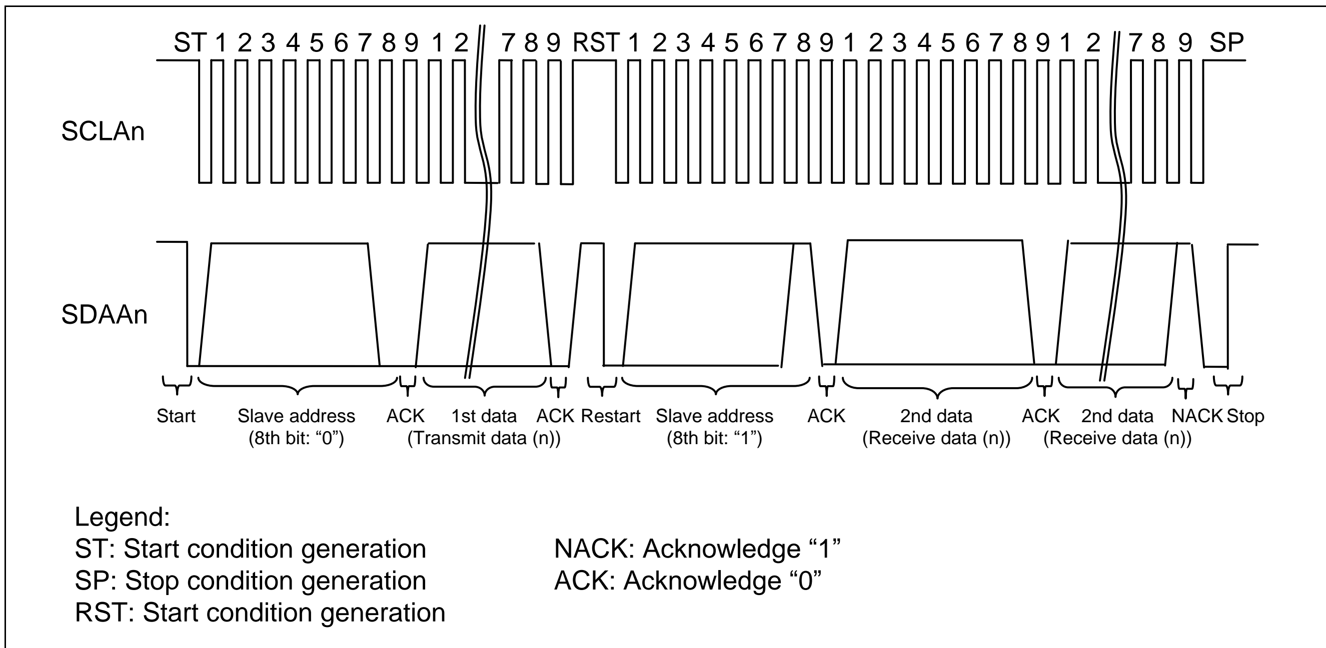


Figure 6-7 Master Composite Signals

6.3 Software Operation

This sample code's specifications take into consideration whether or not OS control*[1] is used. This section describes the processing in these two cases.

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

Here, communication is started by calling the start function. After that, I^2C bus communication is moved forward by the user calling the advance function. Whether or not the I^2C bus communication should be moved forward is determined by the advance function according to whether or not the I^2C bus interrupts have occurred. The sample code's specifications also support multiple calls for I^2C channels performed in the main() routine.

In this normal control (no OS), an event flag (g_iic_Event[]) is set when an interrupt occurs. The advance function verifies those flags and performs the corresponding communication operation. See Table 6-20 for details on these flags.

Note that the states during communication can be verified by checking the return values from the advance function.

(2) **Normal Control (OS present)**

Since operation of this processing has not been included, modifying the code is required.

When on OS is used, OS system calls become the events in the place of the event flags.

Here, after the start function is called, when the advance function is called, the sample code goes to the system call wait state until an event occurs. When an interrupt occurs, an OS system call is generated and a task (the I^2C communication advance processing) is executed by the advance function.

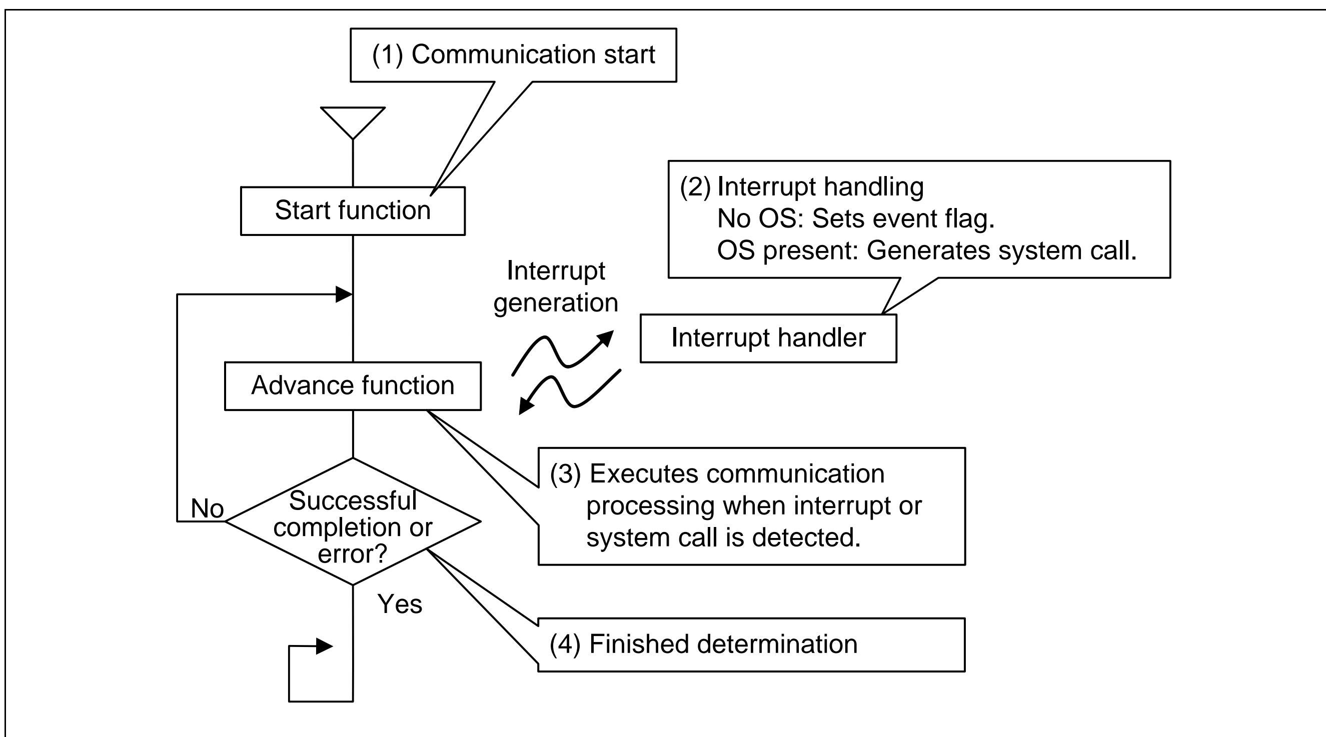


Figure 6-8 Software Operation Outline (No OS/OS Present)

Note: 1. The OS control capabilities the sample code assume µITRON 4.0.

6.4 Software Operating Sequence

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

The following figure shows the operating sequence in normal operation (no OS/OS present).

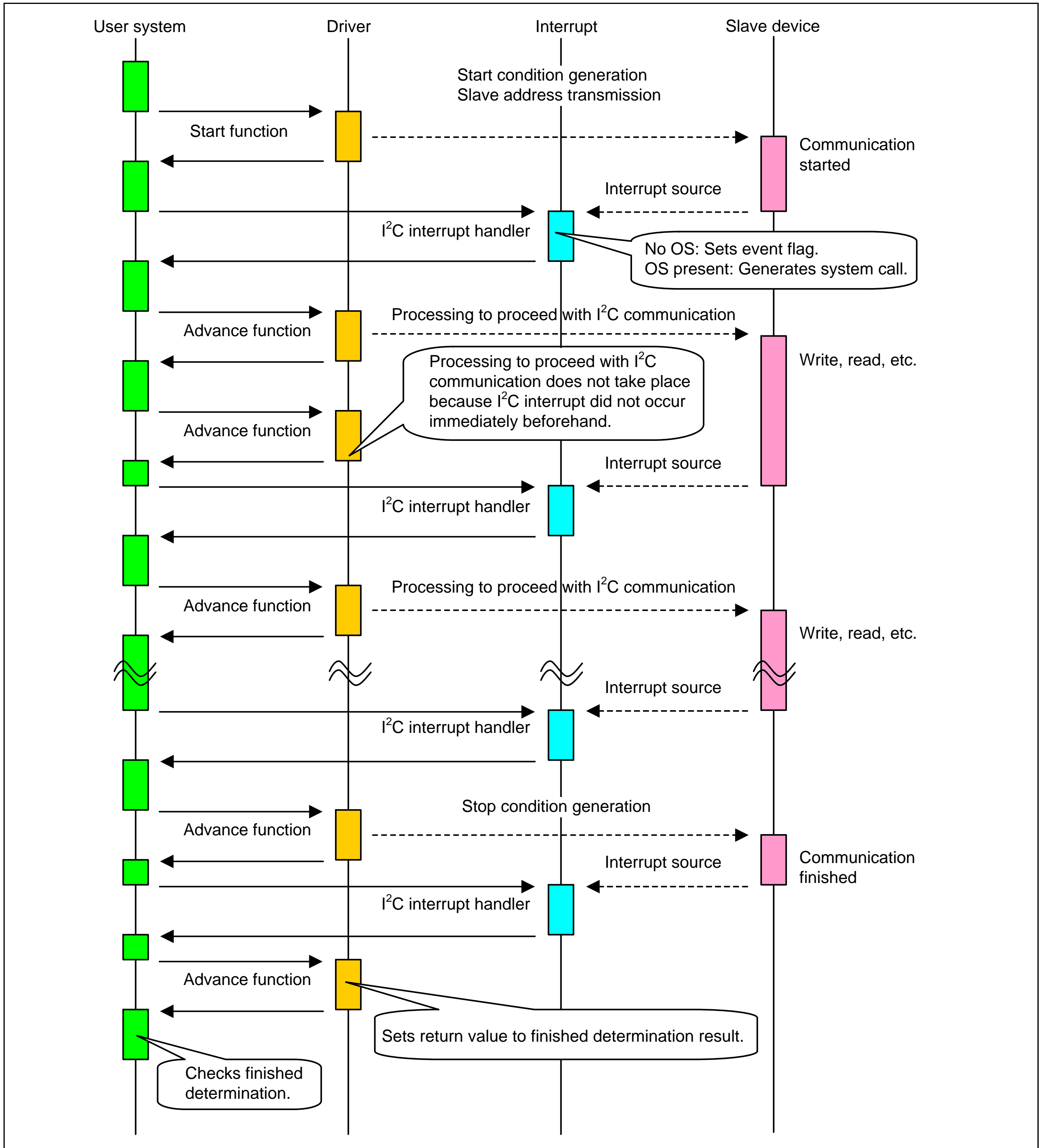


Figure 6-9 Normal Operation (No OS/OS Present) Sequence

6.5 Implementation of Slave Device Control

(1) **Slave Device Management**

Information such as the channels used and the communication data is managed in a structure. Communication between multiple devices on a channel is implemented by setting up a structure for each slave device controlled.

See section 6.13, I²C Communication Information Structure for details on this structure.

(2) **Channel Status Management**

Exclusive control of multiple slave devices connected to a bus is implemented using the g_iic_ChStatus[] channel state flag. See the g_iic_ChStatus[] entry in section 6.15, Variables, for details on the channel state flags.

One of these flags exists for each channel and they are managed in a global variable. These flags are set to the R_IIC_IDLE/R_IIC_FINISH/R_IIC_NACK state (the idle state (communication possible)) if I²C driver initialization completes and communication is not performed on the corresponding bus. The state of these flags is set to R_IIC_COMMUNICATION (communication in progress) during communication. Since these flags are always checked at the start of communication, communication with another device on the same channel will never be started during communication. Simultaneous communication over multiple channels is implemented by managing these flags for each channel.

(3) **Device State Management**

Control of multiple slave devices on the same channel is supported with the *pDevStatus device state flag member in the I²C communication information structure. The communication state of the corresponding device is stored in the device state flag. See section 8.4, Control Methods for Multiple Slave Devices on the Same Channel, for details on the use of these flags.

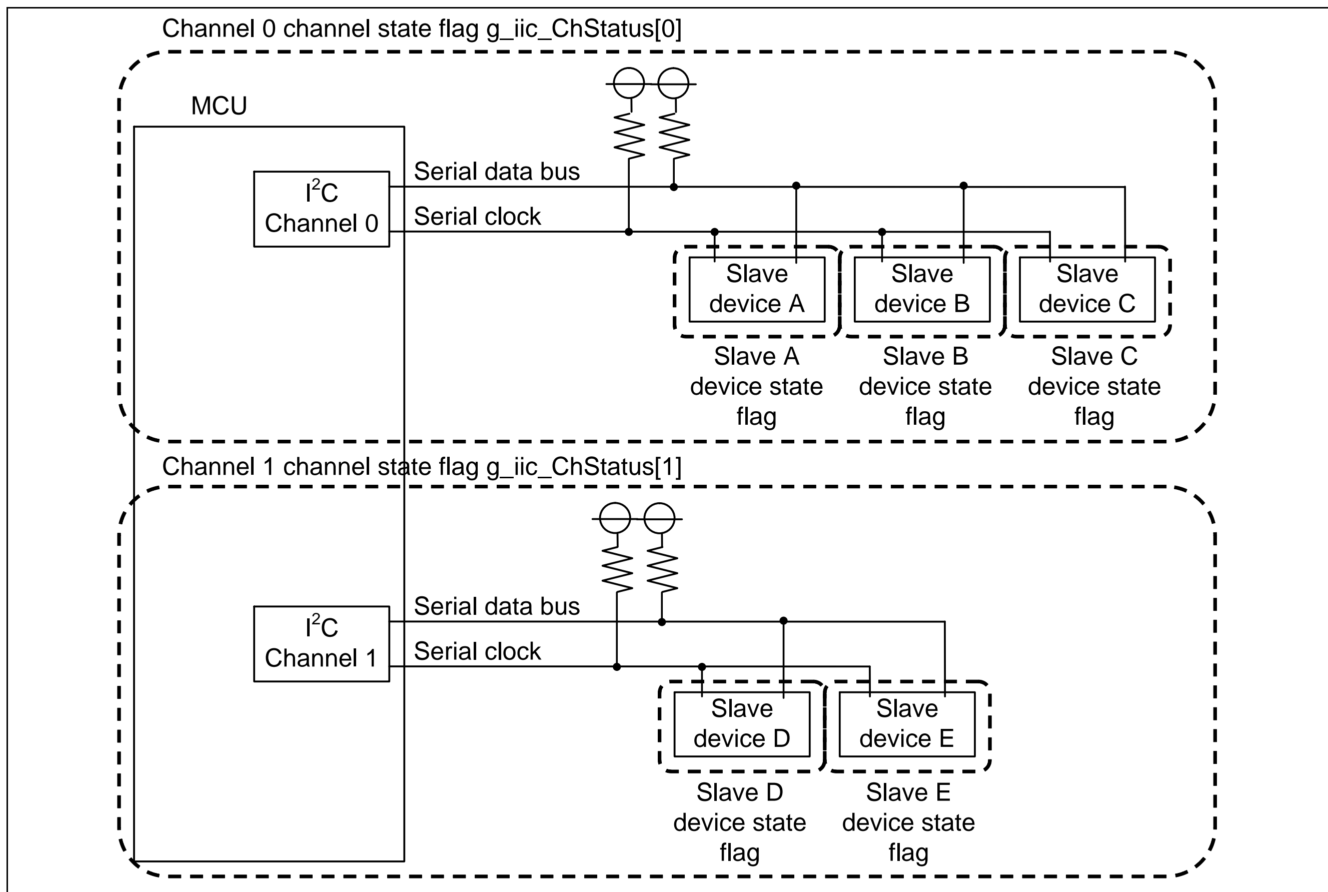


Figure 6-10 Slave Device Control

6.6 Communication Implementation

This sample code manages start conditions, slave device communication, and other processing as a single protocol, and implements communication combinations with this protocol.

6.6.1 States During Control

The following states are defined to implement protocol control.

Table 6-2 States Used for Protocol Control (enum r_iic_drv_internal_status_t)

No.	Constant Name	Description
STS0	R_IIC_STS_NO_INIT	Uninitialized state
STS1	R_IIC_STS_IDLE	Idle state
STS2	R_IIC_STS_ST_COND_WAIT	Start condition generation complete wait state
STS3	R_IIC_STS_SEND_SLVADR_W_WAIT	Slave address [Write] transmission complete wait state
STS4	R_IIC_STS_SEND_SLVADR_R_WAIT	Slave address [Read] transmission complete wait state
STS5	R_IIC_STS_SEND_DATA_WAIT	Data transmission complete wait state
STS6	R_IIC_STS_RECEIVE_DATA_WAIT	Data reception complete wait state
STS7	R_IIC_STS_SP_COND_WAIT	Stop condition generation complete wait state

6.6.2 Events During Control

The following events generated during protocol control are defined.

Note that not only interrupts, but calls the interface functions provided by this sample code are defined as events.

Table 6-3 Events Used for Protocol Control (enum r_iic_drv_internal_event_t)

No.	Event	Event Definition
EV0	R_IIC_EV_INIT	Call r_iic_drv_init()
EV1	R_IIC_EV_GEN_START_COND	Call r_iic_drv_generate_start_cond()
EV2	R_IIC_EV_SEND_SLVADR	Call r_iic_drv_send_slvadr()
EV3	R_IIC_EV_RE_SEND_SLVADR	Call r_iic_drv_re_send_slvadr()
EV4	R_IIC_EV_INT_ADD	Address transmission complete interrupt
EV5	R_IIC_EV_INT_SEND	Data transmission complete interrupt
EV6	R_IIC_EV_INT_RECEIVE	Data reception complete interrupt
EV7	R_IIC_EV_INT_STOP	Stop condition detected interrupt
EV8	R_IIC_EV_INT_AL	Arbitration lost detected interrupt
EV9	R_IIC_EV_INT_NACK	NACK detected interrupt

6.6.3 Protocol State Transitions

In this sample code, the state transitions on calls the provided interface functions and when I²C interrupts occur. The following figures show the protocol state transitions.

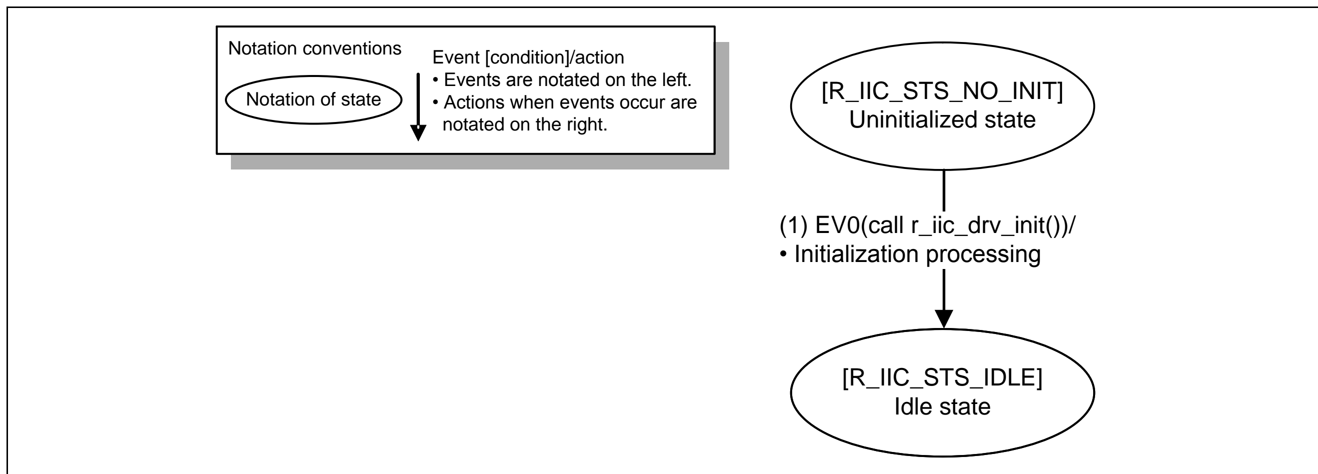


Figure 6-11 Initialization State Transition Diagram

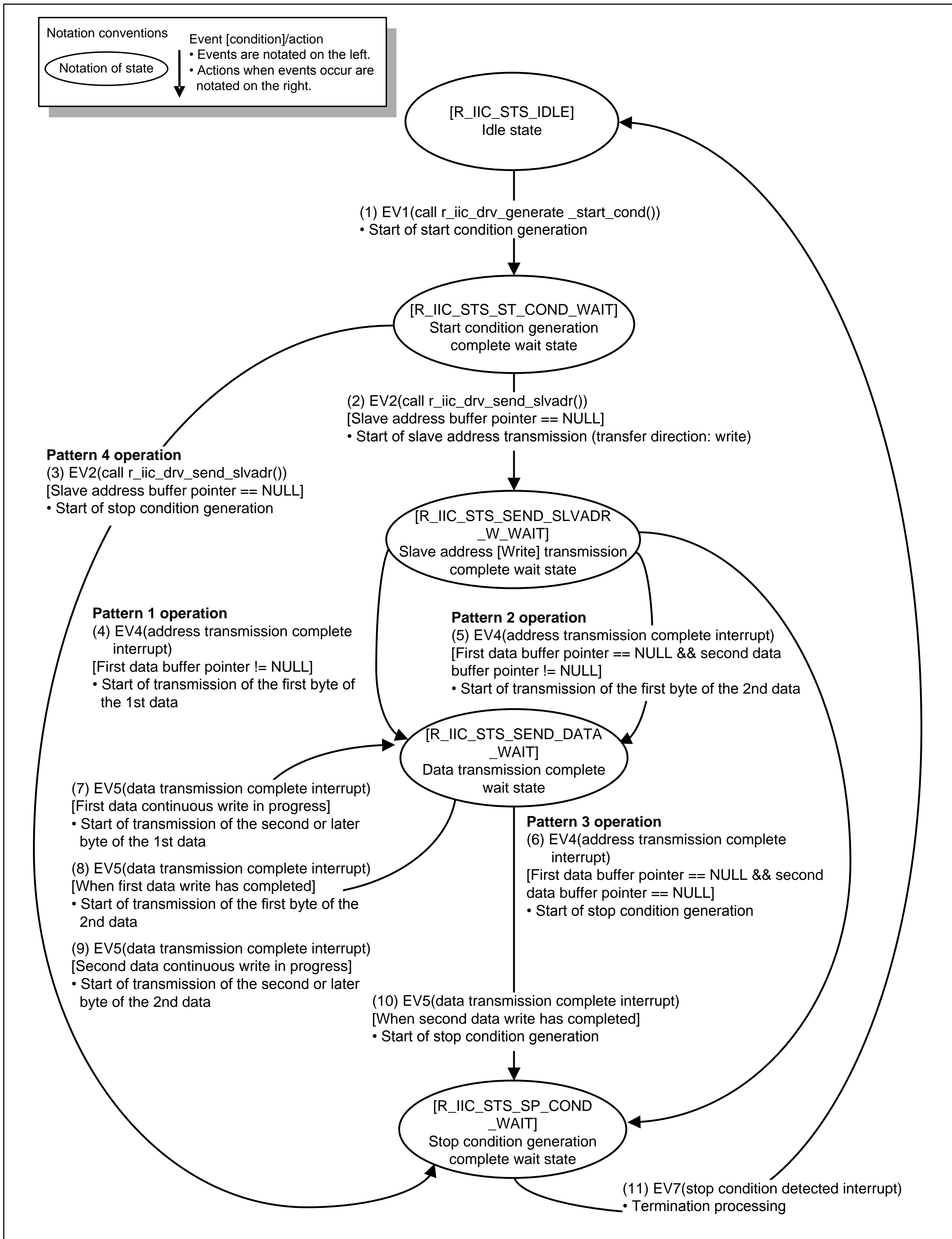


Figure 6-12 Master Transmission State Transition Diagram

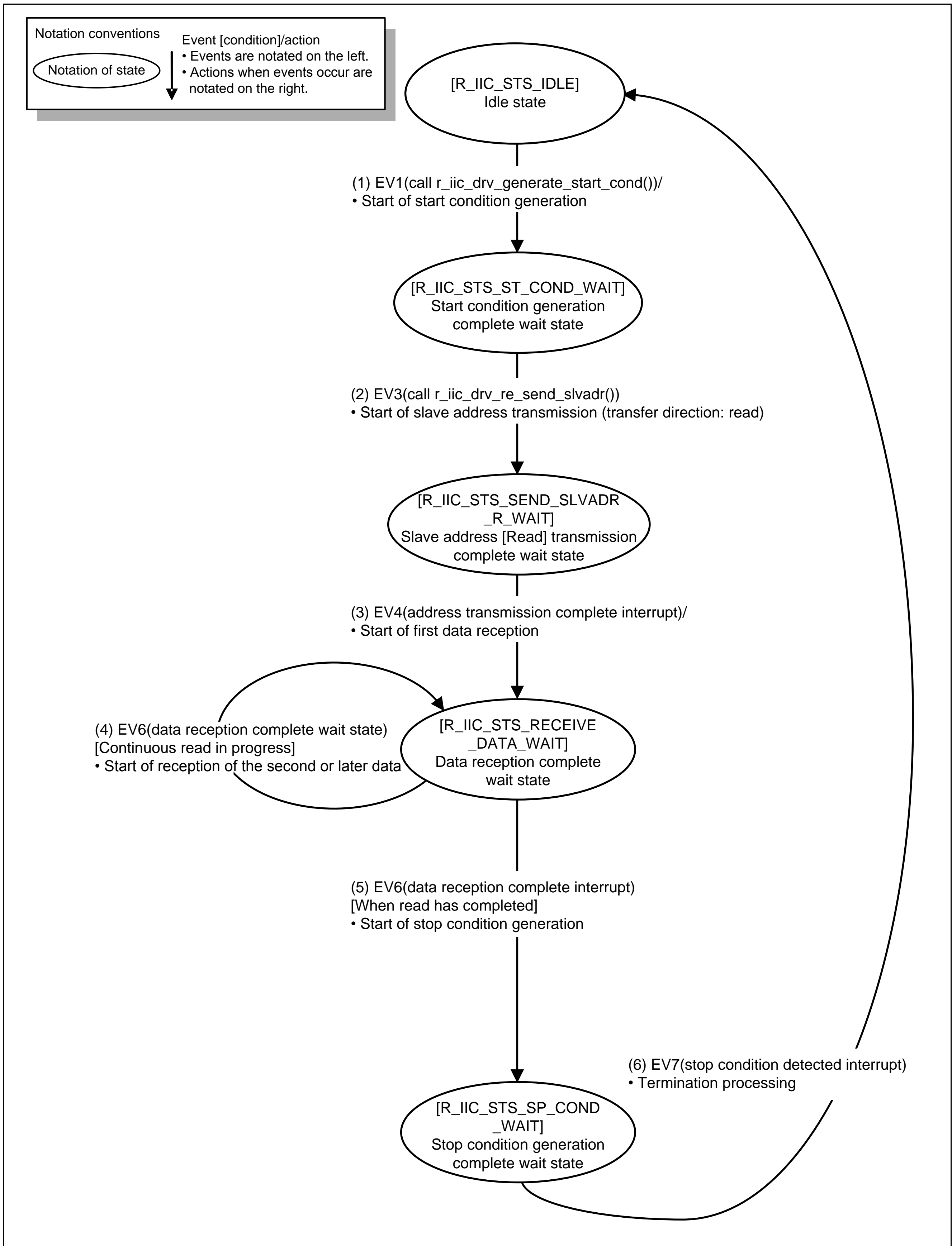


Figure 6-13 Master Reception State Transition Diagram

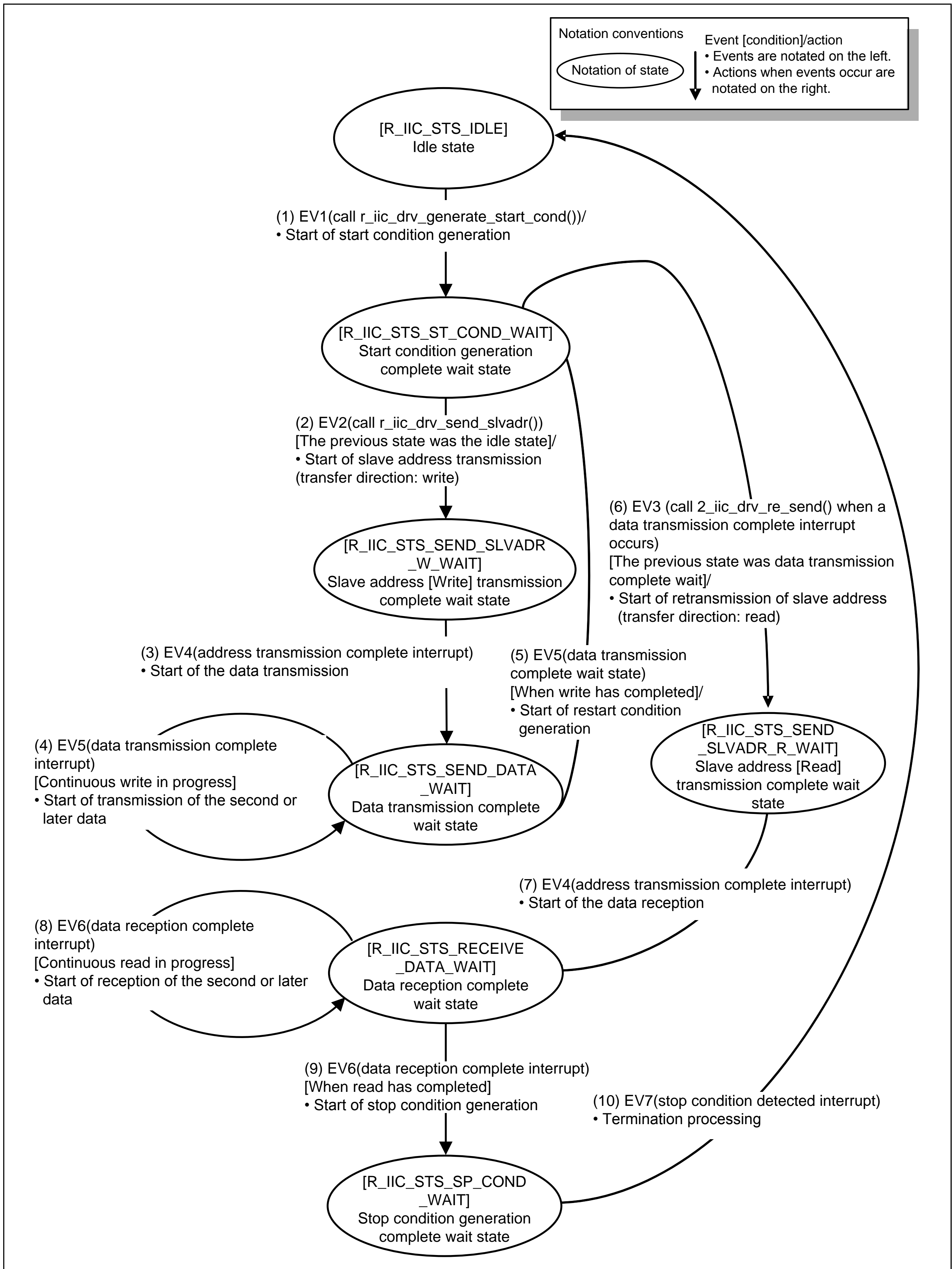


Figure 6-14 Master Composite State Transition Diagram

6.6.4 Protocol State Transition Table

The processing for the operations when the events in Table 6-3 occur in the states shown in Table 6-2 is defined in the following state transition table.

For STS0 and following states, see the "No." column in Table 6-2. For EV0 and other events, see the "No." column in Table 6-3. See Table 6-5 for Func0 and the following functions.

Table 6-4 Protocol State Transition Table (g_iic_mtx_tbl[][])

State		Event									
		EV0	EV1	EV2	EV3	EV4	EV5	EV6	EV7	EV8	EV9
STS0	Uninitialized state [R_IIC_STS_NO_INIT]	Func0	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
STS1	Idle state [R_IIC_STS_IDLE]	ERR	Func1	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
STS2	Start condition generation complete wait state [R_IIC_STS_ST_COND_WAIT]	ERR	ERR	Func2	Func3	ERR	ERR	ERR	ERR	ERR	ERR
STS3	Slave address [Write] transmission complete wait state [R_IIC_STS_SEND_SLVADR_W_WAIT]	ERR	ERR	ERR	ERR	Func4	ERR	ERR	ERR	Func8	Func9
STS4	Slave address [Read] transmission complete wait state [R_IIC_STS_SEND_SLVADR_R_WAIT]	ERR	ERR	ERR	ERR	Func4	ERR	ERR	ERR	Func8	Func9
STS5	Data transmission complete wait state [R_IIC_STS_SEND_DATA_WAIT]	ERR	ERR	ERR	ERR	ERR	Func5	ERR	ERR	Func8	Func9
STS6	Data reception complete wait state [R_IIC_STS_RECEIVE_DATA_WAIT]	ERR	ERR	ERR	ERR	ERR	ERR	Func6	ERR	Func8	Func9
STS7	Stop condition generation complete wait state [R_IIC_STS_SP_COND_WAIT]	ERR	ERR	ERR	ERR	ERR	ERR	ERR	Func7	Func8	Func9

Note: "ERR" indicates R_IIC_ERR_OTHER. Cases where an event that has no meaning in that state is reported are all handled as errors.

6.6.5 Protocol State Transition Registered Functions

The functions registered in the state transition table are defined as follows.

Table 6-5 Protocol State Transition Registered Functions

Processing	Function	Overview
Func0	r_iic_drv_init()	Initialization
Func1	r_iic_drv_generate_start_cond()	Start condition generation
Func2	r_iic_drv_send_slvadr()	Slave address transmission
Func3	r_iic_drv_re_send_slvadr()	Slave address retransmission
Func4	r_iic_drv_after_send_slvadr()	Post slave address transmission completion processing
Func5	r_iic_drv_write_data_sending()	Data transmission
Func6	r_iic_drv_read_data_receiving()	Data reception
Func7	r_iic_drv_release()	Communication termination
Func8	r_iic_drv_arbitration_lost()	Arbitration lost error handling
Func9	r_iic_drv_nack()	NACK error handling

6.6.6 Processing at Protocol State Transitions

This section describes the processing performed by r_iic_drv_func_table() (referred to below as the processing that advances communication) when a protocol state transition occurs.

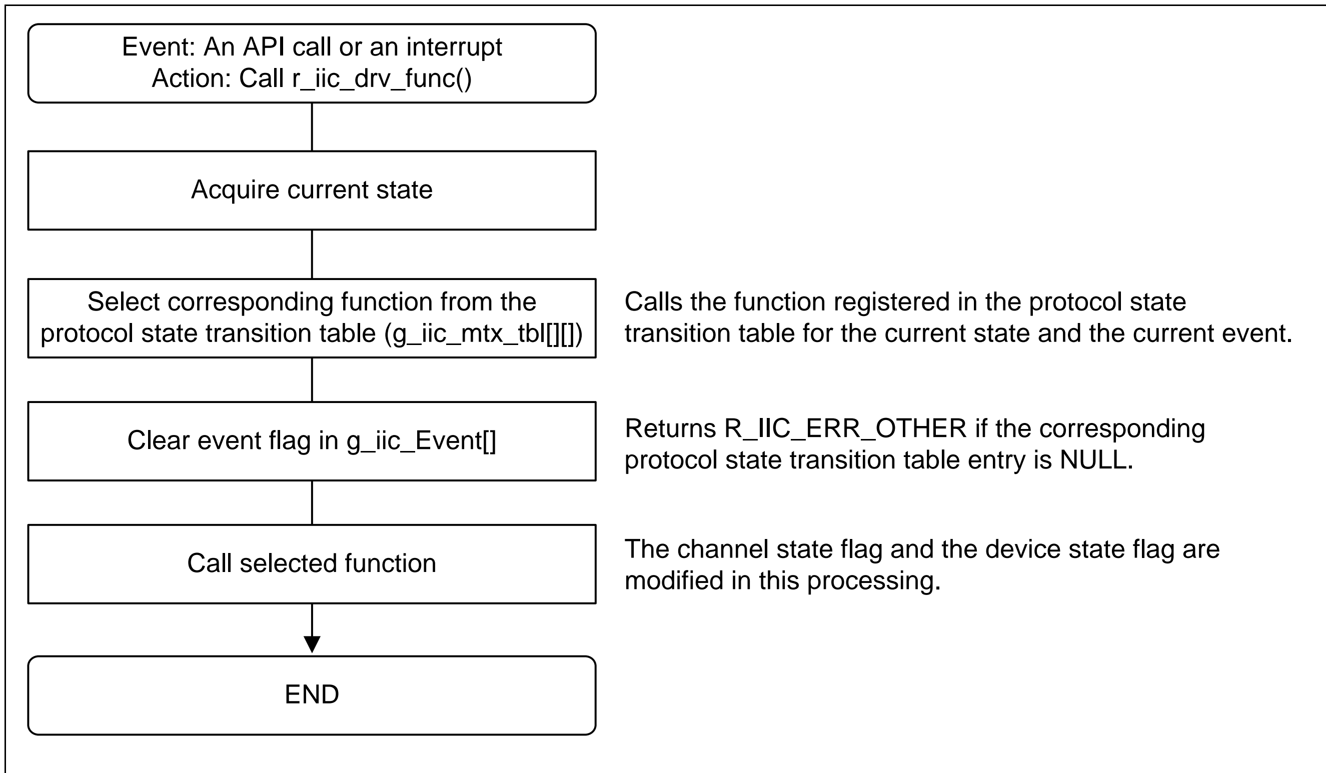


Figure 6-15 Communication Advance Processing Calling Mechanism

6.7 Interrupt Generation Timing

This section describes the interrupt timing in this driver.

Legend:

ST: Start condition

AD6-AD0: Slave address

/W: Transfer direction bit “0” (Write), R: Transfer direction bit “1” (Read)

/ACK: Acknowledge “0”, NACK: Acknowledge “1”

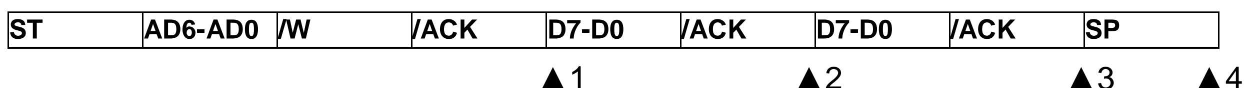
D7-D0: Data

RST: Restart condition

SP: Stop condition

6.7.1 Master Transmission

(1) **Pattern 1**



▲1: Address transmission complete interrupt *[1]

▲2: Data transmission complete interrupt (1st data) *[2]

▲3: Data transmission complete interrupt (2nd data) *[2]

▲4: Stop condition detected interrupt

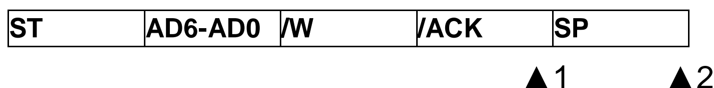
(2) **Pattern 2**

▲1: Address transmission complete interrupt *[1]

▲2: Data transmission complete interrupt (2nd data) *[2]

▲3: Stop condition detected interrupt

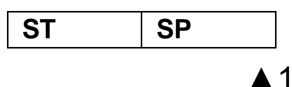
(3) **Pattern 3**



▲1: Address transmission complete interrupt *[1]

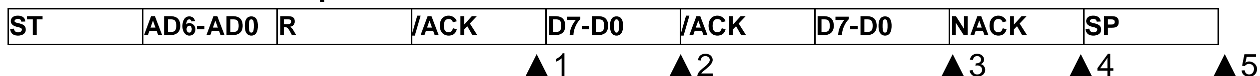
▲2: Stop condition detected interrupt

(4) **Pattern 4**



▲1: Stop condition detected interrupt

6.7.2 Master Reception



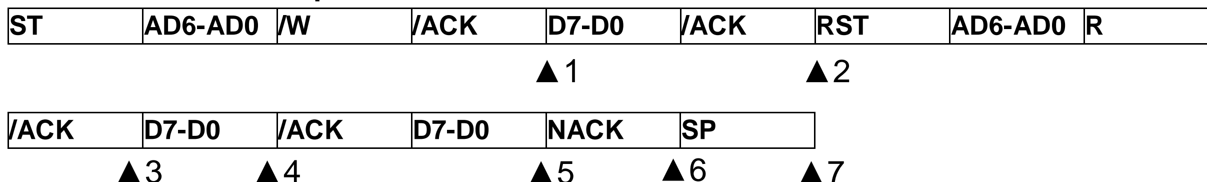
▲1: Address transmission complete interrupt *1

▲2, ▲3: Data reception complete interrupt (2nd data) *3

▲4: Data reception complete interrupt (2nd data) *4

▲5: Stop condition detected interrupt

6.7.3 Master Composite



▲1: Address transmission complete interrupt (transfer direction: write) *1

▲2: Data transmission complete interrupt (1st data) *2

▲3: Address transmission complete interrupt (transfer direction: read) *1

▲4, ▲5: Data reception complete interrupt (2nd data) *3

▲6: Data reception complete interrupt (2nd data) *4

▲7: Stop condition detected interrupt

Notes:
1. Generated on the fall of the ninth clock pulse during address transmission.
2. Generated on the fall of the ninth clock pulse during data transmission.
3. Generated on the fall of the eighth clock pulse during data reception.
4. Generated on the fall of the ninth clock pulse during data reception.

6.8 Callback Function

This function is called either if communication completes successfully or if it terminated with an error. To use this functionality, specify a function name for the CallBackFunc member of the I^2C communication information structure. See section 6.13, I^2C Communication Information Structure for details on this structure.

6.9 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.

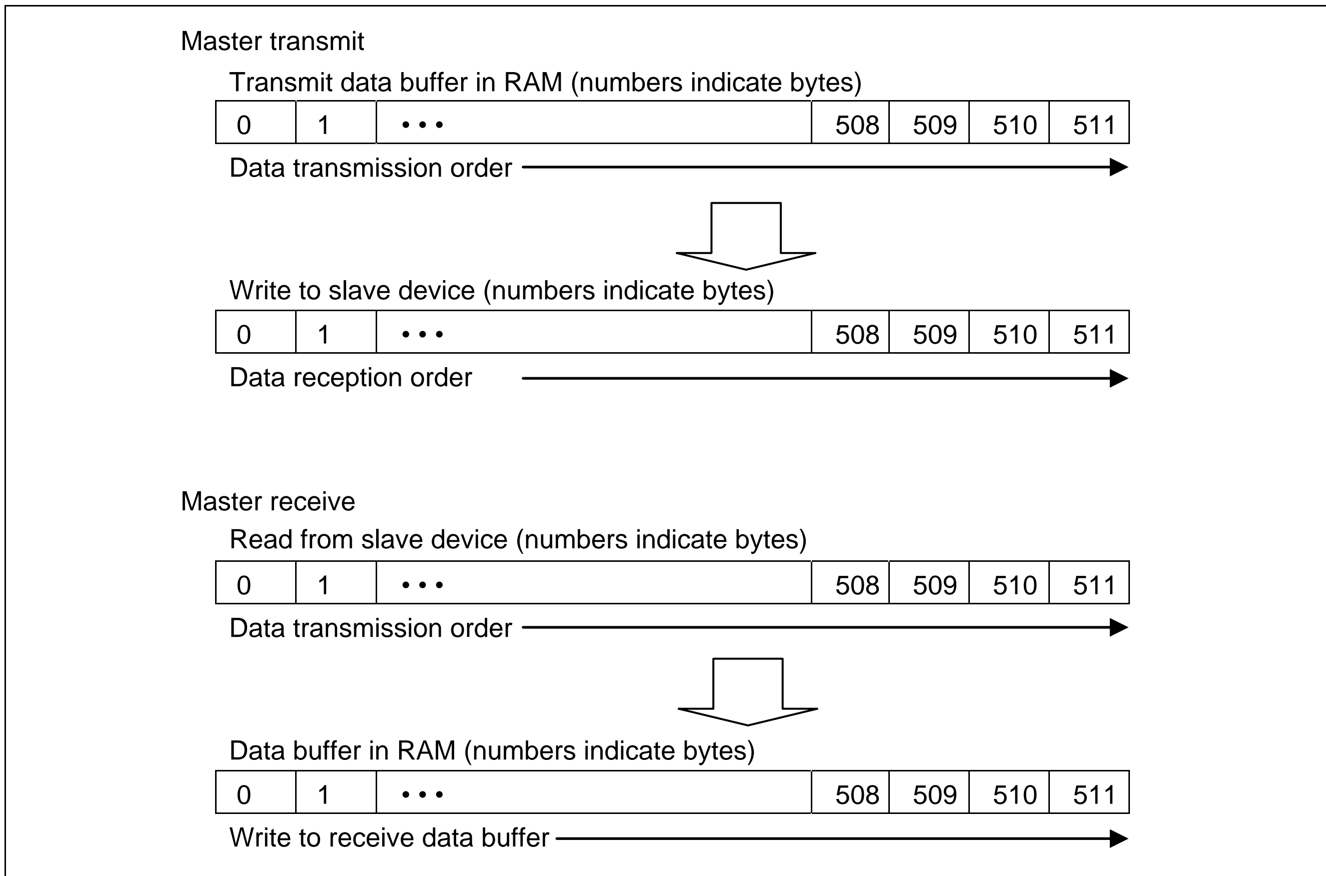


Figure 6-16 Storage of Transfer Data

6.10 Required Memory Sizes

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

The following table lists the required memory sizes.

Table 6-6 Required Memory Sizes

Memory Used	Size	Remarks
ROM	5,856 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
RAM	32 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
Maximum usable user stack	42 bytes	
Maximum usable interrupt stack	6 bytes	

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

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

The following table lists the required memory sizes.

Table 6-7 Required Memory Sizes

Memory Used	Size	Remarks
ROM	4,035 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
RAM	30 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
Maximum usable user stack	30 bytes	
Maximum usable interrupt stack	6 bytes	

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

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

The following table lists the required memory sizes.

Table 6-8 Required Memory Sizes

Memory Used	Size	Remarks
ROM	9,230 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
RAM	48 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
Maximum usable user stack	272 bytes	
Maximum usable interrupt stack	6 bytes	

Note: The required memory sizes differ according to the C compiler version and the compile options.
The maximum user stack size is the stack size for the entire project.

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

The following table lists the required memory sizes.

Table 6-9 Required Memory Sizes

Memory Used	Size	Remarks
ROM	5,835 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
RAM	50 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
Maximum usable user stack	30 bytes	
Maximum usable interrupt stack	6 bytes	

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

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

The following table lists the required memory sizes.

Table 6-10 Required Memory Sizes

Memory Used	Size	Remarks
ROM	9,633 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
RAM	48 bytes	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_os.c r_iic_drv_sfr.c r_iic_drv_sub.c
Maximum usable user stack	146 bytes	
Maximum usable interrupt stack	6 bytes	

Note: The required memory sizes differ according to the C compiler version and the compile options.
The maximum user stack size is the stack size for the entire project.

6.11 File Structure

The following table lists the files used by the sample code. Note that files that are generated automatically by the integrated development environment are not listed.

Table 6-11 File Structure

File/Folder		Description
\an_r01an1074ej0104_rl78_iic	<DIR>	Sample code folder
r01an1074ej0104_rl78.pdf		Application note
\ source	<DIR>	Program storage folder
\ r_iic_drv_rl78	<DIR>	I^{2}C single master control software folder
r_iic_drv_api.c		API source file
r_iic_drv_api.h		API header file
r_iic_drv_int.c		Interrupt handler source file
r_iic_drv_int.h		Interrupt handler header file
r_iic_drv_os.c		OS processing source file
r_iic_drv_os.h		OS processing header file
r_iic_drv_sfr.h.rl78g1c		Common register definitions header file (for the RL78/G1C)
r_iic_drv_sfr.h.rl78g14		Common register definitions header file (for the RL78/G14)
r_iic_drv_sfr.h.rl78l1c		Common register definitions header file (for the RL78/L1C)
r_iic_drv_sfr.h.rl78l12		Common register definitions header file (for the RL78/L12)
r_iic_drv_sfr.h.rl78l13		Common register definitions header file (for the RL78/L13)
r_iic_drv_sfr_rl78g1c.c		Common register definitions source file (for the RL78/G1C)
r_iic_drv_sfr_rl78g14.c		Common register definitions source file (for the RL78/G14)
r_iic_drv_sfr_rl78l1c.c		Common register definitions source file (for the RL78/L1C)
r_iic_drv_sfr_rl78l12.c		Common register definitions source file (for the RL78/L12)
r_iic_drv_sfr_rl78l13.c		Common register definitions source file (for the RL78/L13)
r_iic_drv_sub.c		Internal function source file
r_iic_drv_sub.h		Internal function header file

Note: A file with a filename of the form r_iic_drv_sfr.hXXX has been created for each microcontroller. One of these must be renamed to r_iic_drv_sfr.h and used. If there is no such file for the microcontroller used, the user must refer to these files and create an appropriate r_iic_drv_sfr.h file.

6.12 Constants

6.12.1 Definitions

The definitions of the constants used in the sample code are shown below.

Table 6-12 Macro Definitions (Return values, channel state flag, and device state flag management values)

Constant Name	Setting Value	Description
R_IIC_NO_INIT	(error_t)(0)	Uninitialized state
R_IIC_IDLE	(error_t)(1)	Idle state: ready for communication
R_IIC_FINISH	(error_t)(2)	Idle state: previous communication complete, ready for communication
R_IIC_NACK	(error_t)(3)	Idle state: previous communication NACK complete, ready for communication
R_IIC_COMMUNICATION	(error_t)(4)	Communication in progress
R_IIC_LOCK_FUNC	(error_t)(5)	API processing in progress This state occurs in the following cases: • When another API function is called during API processing
R_IIC_BUS_BUSY	(error_t)(6)	Bus busy This state occurs in the following cases: • When, during communication, either the initialization function or a start function has been called • When another device is communicating over the same channel and either a start function or the advance function has been called
R_IIC_ERR_PARAM	(error_t)(-1)	Parameter error
R_IIC_ERR_AL	(error_t)(-2)	Arbitration lost error
R_IIC_ERR_NON_REPLY	(error_t)(-3)	No response error
R_IIC_ERR_SDA_LOW_HOLD	(error_t)(-4)	SDA held low error when SDL pseudo clock generate function called
R_IIC_ERR_OTHER	(error_t)(-5)	Other error

Table 6-13 Macro Definitions (Must not be modified)

Constant Name	Setting Value	Description
W_CODE	(uint8_t)(0x00)	Setting value for when the slave address transfer direction is write
R_CODE	(uint8_t)(0x01)	Setting value for when the slave address transfer direction is read
R_IIC_TRUE	(uint8_t)(0x01)	Flag “ON”
R_IIC_FALSE	(uint8_t)(0x00)	Flag “OFF”
R_IIC_HI	(uint8_t)(0x01)	Port “H”
R_IIC_LOW	(uint8_t)(0x00)	Port “L”
R_IIC_OUT	(uint8_t)(0x00)	Port Output
R_IIC_IN	(uint8_t)(0x01)	Port Input

Table 6-14 Macro Definitions (User modifiable)

Constant Name	Setting Value	Description
MAX_IIC_CH_NUM	(uint8_t)(2)	One plus the maximum number of channels that can be used at the same time. Sets to 2 in this sample code.
R_IIC_CH0_LCLK	(uint8_t)(20)	Setting value for the IICA low-level setting register 0 (IICWL0)*1
R_IIC_CH0_HCLK	(uint8_t)(18)	Setting value for the IICA high-level setting register 0 (IICWH0)*1
R_IIC_CH1_LCLK	(uint8_t)(20)	Setting value for the IICA low-level setting register 1 (IICWL1)*1
R_IIC_CH1_HCLK	(uint8_t)(18)	Setting value for the IICA high-level setting register 1 (IICWH1)*1
REPLY_CNT	(uint32_t)(10000)	Advanced function counter value*2
START_COND_WAIT	(uint16_t)(100)	Start condition generation wait counter value*2
STOP_COND_WAIT	(uint16_t)(100)	Stop condition generation wait counter value*2
BUSCHK_CNT	(uint16_t)(100)	Bus busy check counter value*2
SDACHK_CNT	(uint16_t)(100)	SDA level check counter value*2
SCLCHK_CNT	(uint16_t)(100)	SCL level check counter value*2
SCL_L_WAIT	(uint16_t)(100)	SCL pseudo clock low-level width wait time*3
SCL_H_WAIT	(uint16_t)(100)	SCL pseudo clock high-level width wait time*3

Notes: 1. Transfer rate setting

The transfer rate must be set for each channel used. See the RL78 Family microcontroller User's Manual - Hardware for details on this setting.

Transfer rates up to a maximum of 400 kHz can be set. However, if standard mode devices and fast mode devices are used together, the standard mode maximum rate of 100 kHz must be set.

2. Counter value settings

These are counters for software loops. This means that the loop time will depend on the system clock actually used. These values must be set according to the system clock used.

3. Transfer rates for pseudo clock output

This sample code includes an SCL pseudo clock generation function for generating an SCL pseudo clock to release SDA when a slave device is holding SDA at the low level.

This function implements the clock high and low levels using the microcontroller port functions. The widths of the pseudo clock high and low level periods must be assured to be at least as long as the minimum values given by the I²C-bus specification. (See the table below.)

The high and low-level width setting waits are implemented with software loops. This means that the wait times will change with the system clock actually used.

The widths of the high and low level periods are set with the SCL_L_WAIT and SCL_H_WAIT macro definitions. The user must manage these values so that they meet the specifications of the I²C-bus specification according to the system clock actually used.

Table 6-15 Minimum Values for High and Low-Level Widths stipulated by the I2C-Bus Specification

	Fast Mode	Standard Mode
Low-Level width (tLow)	1.3 µs	4.7 µs
High-Level width (tHigh)	0.6 µs	4.0 µs

6.13 Structures and Unions

6.13.1 I^{2}C Communication Information Structure

The following figure shows the I^{2}C communication information structure used in the sample code. An instance of this structure must be set up for each slave device used.

```
typedef struct
{
  uint8_t          *pSlvAdr;       /* Pointer for Slave address buffer       */
  uint8_t          *pData1st;      /* Pointer for 1st Data buffer            */
  uint8_t          *pData2nd;      /* Pointer for 2nd Data buffer            */
  error_t          *pDevStatus;    /* Device status flag                     */
  uint32_t         Cnt1st;         /* 1st Data counter                       */
  uint32_t         Cnt2nd;         /* 2nd Data counter                       */
  r_iic_callback   CallBackFunc;   /* Callback function                      */
  uint8_t          ChNo;           /* Channel No.                            */
  uint8_t          rsv1;
  uint8_t          rsv2;
  uint8_t          rsv3;
} r_iic_drv_info_t;
```

Figure 6-17 I^{2}C Communication Information Structure

(1) **Structure Members**

The following table lists the structure members. See Table 6-17 and Table 6-18 for details on setting the r_iic_drv_info_t members.

Table 6-16 Structure r_iic_drv_info_t Members

Structure member	Description
*pSlvAdr	Slave address storage buffer pointer Allocate one byte for this data.
*pData1st	1st data storage buffer pointer
*pData2nd	2nd data storage buffer pointer
*pDevStatus	Device state flag pointer Device states can be checked during communication, even when multiple devices are connected to the same channel. Allocate one byte for this data. See section 8.4 for a usage example.
Cnt1st	1st data counter (byte count)
Cnt2nd	2nd data counter (byte count)
CallBackFunc	Callback function
ChNo	Channel number of the used device Set this to the channel number of the bus used.
rsv1 rsv2 rsv3	Alignment adjustment members

(2) **Settings**

Table 6-17 lists the allowable range of user settings for the structure r_iic_drv_info_t members for master transmission and Table 6-18 lists those for master reception and master composite.

Table 6-17 User Setting Ranges for r_iic_drv_info_t Members: Master Transmission

Structure Member	Allowable User Setting Range			
	Master Transmission Pattern 1	Master Transmission Pattern 2	Master Transmission Pattern 3	Master Transmission Pattern 4
*pSlvAdr	Slave address storage source address	Slave address storage source address	Slave address storage source address	NULL
*pData1st	1st data storage source address	NULL	NULL	NULL
*pData2nd	2nd data (transmit data) storage source address	Second data (transmit data) storage source address	NULL	NULL
*pDevStatus	Device state storage source address	Device state storage source address	Device state storage source address	Device state storage source address
Cnt1st	0000 0001h*[1] to FFFF FFFFh	(Invalid setting)	(Invalid setting)	(Invalid setting)
Cnt2nd	0000 0001h*[1] to FFFF FFFFh	0000 0001h*[1] to FFFF FFFFh	(Invalid setting)	(Invalid setting)
CallBackFunc	If used: specify the name of the function. If not, set to NULL.	If used: specify the name of the function. If not, set to NULL.	If used: specify the name of the function. If not, set to NULL.	If used: specify the name of the function. If not, set to NULL.
ChNo	00h to FFh	00h to FFh	00h to FFh	00h to FFh
rsv1, rsv2, rsv3	(Invalid setting)	(Invalid setting)	(Invalid setting)	(Invalid setting)

Table 6-18 User Setting Ranges for r_iic_drv_info_t Members: Master Reception and Master Composite

Structure Member	Allowable User Setting Range	
	Master Reception	Master Composite
*pSlvAdr	Slave address storage source address	Slave address storage source address
*pData1st	(Invalid setting)	1st data storage source address
*pData2nd	2nd data (receive data) storage destination address	2nd data (receive data) storage destination address
*pDevStatus	Device state storage source address	Device state storage source address
Cnt1st	(Invalid setting)	0000 0001h*[1] to FFFF FFFFh
Cnt2nd	0000 0001h*[1] to FFFF FFFFh	0000 0001h*[1] to FFFF FFFFh
CallBackFunc	If used: specify the name of the function. If not, set to NULL.	If used: specify the name of the function. If not, set to NULL.
ChNo	00h to FFh	00h to FFh
rsv1, rsv2, rsv3	(Invalid setting)	(Invalid setting)

Note: 1. The value 0 is illegal in both Table 6-17 and Table 6-18.

(3) **Callback Function**

This function is called either if communication completes successfully or if it terminated with an error. To use this functionality, specify a function name for the CallBackFunc member.

(4) **Notes On Settings**

During master transmission, the data stored in the members of this structure is referenced to determine what operation to perform. This sample code may fail to operate correctly if any values other than those listed in Table 6-16 are used.

6.13.2 Internal Information Management Structure

The following figure shows the internal information management structure used by the sample code. Since this structure is controlled by the sample code, there is no need for it to be set by the user.

```
typedef struct
{
  r_iic_drv_internal_mode_t      Mode;         /* Mode of Control Protocol        */
  r_iic_drv_internal_status_t    N_status;     /* Internal Status of NOW          */
  r_iic_drv_internal_status_t    B_status;     /* Internal Status of BEFORE       */
} r_iic_drv_internal_info_t;
```

Figure 6-18 Internal information management structure

(1) **Structure Members**

The following table lists the structure members.

Table 6-19 Structure r_iic_drv_internal_info_t Members

Structure Member	Description
Mode	I²C protocol mode See Table 6-19 for the definition of the data stored.
N_status	The protocol control current state. Values defined in Table 6-2 are stored in this member.
B_status	The protocol control previous state. Values defined in Table 6-2 are stored in this member.

6.14 Enumerated Types

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

Table 6-20 I²C Protocol Operating Modes (enum r_iic_drv_internal_mode_t)

	Description
R_IIC_MODE_NONE	No communication state This mode is transitioned to from the uninitialized state or from the idle state.
R_IIC_MODE_WRITE	Master transmission in progress This mode is transitioned to by starting communication with the master transmission start function R_IIC_Drv_MasterTx().
R_IIC_MODE_READ	Master reception in progress This mode is transitioned to by starting communication with the master reception start function R_IIC_Drv_MasterRx().
R_IIC_MODE_COMBINED	Master composite operation in progress This mode is transitioned to by starting communication with the master composite start function R_IIC_Drv_MasterTRx().

6.15 Variables

The following table lists the global variable.

Table 6-21 Global Variable

Type	Valuable	Description	Function Used
uint8_t	g_iic_ChStatus[MAX_IIC_CH_NUM]	Channel state flag The communication state defined in Table 6-11 can be checked. Set this variable to R_IIC_NO_INIT at initialization. After that, do not change the value of this flag except if a forcible initialization is performed.	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance R_IIC_Drv_GenClk R_IIC_Drv_Reset
r_iic_drv_internal_event_t	g_iic_Event[MAX_IIC_CH_NUM]	Event flag This flag is set when an interrupt occurs and is cleared by the advance function. See Table 6-3.	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance
r_iic_drv_internal_info_t	g_iic_InternalInfo[MAX_IIC_CH_NUM]	Internal information management This variable is managed by the sample code and must not be set by the user.	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance
uint32_t	g_iic_ReplyCnt[MAX_IIC_CH_NUM]	Advance function counter This is the upper limit on the number of calls the advance function. It is decremented by the advance function called by the user. It is initialized when an event occurs. If it reaches 0, the channel state flag and the device state flag are set to R_IIC_ERR_NON_REPLY. The counter value can be modified with the REPLY_CNT macro definition. This macro should be set appropriately for the actual user system.	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance
bool	g_iic_Api[MAX_IIC_CH_NUM]	API flag This flag is used to prevent multiple calls this sample code's API. It is set when API processing starts and cleared after that processing completes.	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance R_IIC_Drv_GenClk R_IIC_Drv_Reset

6.16 Functions

The following table lists the Functions.

Table 6-22 Functions

Function	Description
R_IIC_Drv_Init()	I²C driver initialization function
R_IIC_Drv_MasterTx()	Master transmission start function
R_IIC_Drv_MasterRx()	Master reception start function
R_IIC_Drv_MasterTRx()	Master composite start function
R_IIC_Drv_Advance()	Advance function
R_IIC_Drv_GenClk()	SCL pseudo clock generation function
R_IIC_Drv_Reset()	I²C driver reset function

6.17 State Transition Diagram

The following figure is a diagram showing state transitions for each channel.

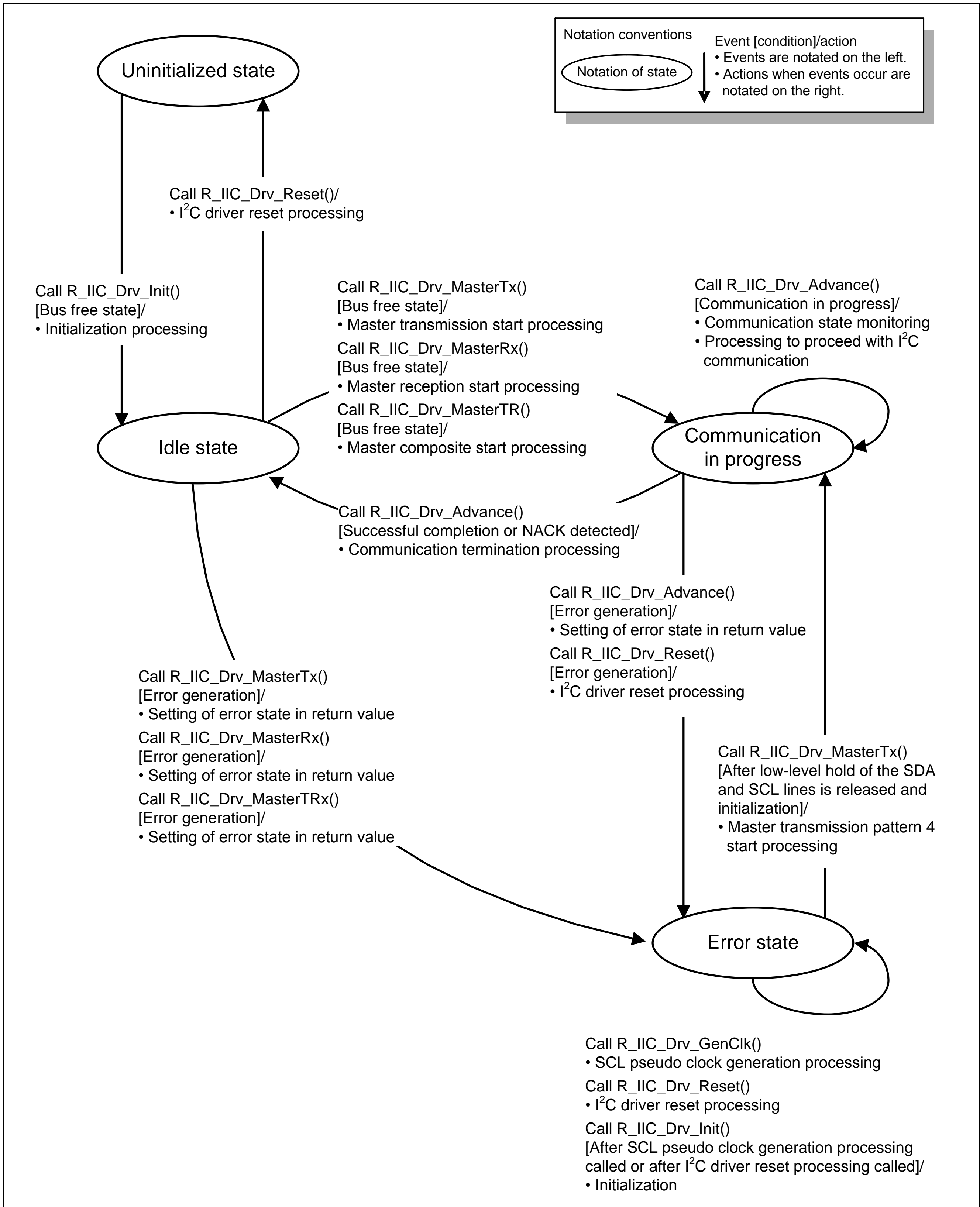


Figure 6-19 State Transition Diagram

6.17.1 Error State Definitions

In this sample code, occurrences of the following phenomena are defined to be error states. Error occurrences can be verified from the return values after return from the API functions. See the return values from each of the API functions in section 6.18, Function Specifications, for methods for responding when an error occurs.

(1) **Parameter Error**

Return value: R_IIC_ERR_PARAM

If the arguments were not set appropriately when an API function was called.

(2) **Arbitration Lost**

Return value: R_IIC_ERR_AL

If arbitration was lost. See the RL78 Family microcontroller User's Manual - Hardware for the conditions where this occurs.

(3) **No Response Error**

Return value: R_IIC_NON_REPLY

The following cases result in a no response error.

- If the number of advance function calls exceeds the limit
- When a start function was called, if the bus was monitored for a fixed time but was not released
- If the start condition generation processing was performed but it was not detected after a fixed time had passed
- If the stop condition generation processing was performed when the advance function was called but it was not detected after a fixed time had passed

Note that the start condition generation wait time is measured with a software loop. The counter value can be set by the user. Set this value according to the system clock used. See Table 6-13 for the definition of this counter.

(4) **SDA Held Low (Recovery not possible)**

Return value: R_IIC_ERR_SDA_LOW_HOLD

If an SCL pseudo clock was generated but SDA remained held at the low level.

(5) **Other Errors**

Return value: R_IIC_ERR_OTHER

If an error other than (1) to (4) above occurred.

6.17.2 Flag States at State Transitions

The following table lists the states of the flags when a state transition occurs.

Table 6-23 Flag States at State Transitions

State	Channel State Flag	Device State Flag (Communicating Device)	I²C Protocol Operating Mode	Current State of The Protocol Control
	g_iic_ChStatus[]	I²C Communication Information Structure *pDevStatus	Internal Communication Information Structure Mode	Internal Communication Information Structure N_status
Uninitialized state	R_IIC_NO_INIT	R_IIC_NO_INIT	R_IIC_MODE_NONE	R_IIC_STS_NO_INIT
Idle state	R_IIC_IDLE R_IIC_FINISH R_IIC_NACK	R_IIC_IDLE R_IIC_FINISH R_IIC_NACK	R_IIC_MODE_NONE	R_IIC_STS_IDLE
Communication in progress (master transmission)	R_IIC_COMMUNICATION	R_IIC_COMMUNICATION	R_IIC_MODE_WRITE	R_IIC_STS_ST_COND_WAIT
				R_IIC_STS_SEND_SLVADR_W_WAIT
				R_IIC_STS_SEND_SLVADR_R_WAIT
				R_IIC_STS_SEND_DATA_WAIT
				R_IIC_STS_RECEIVE_DATA_WAIT
				R_IIC_STS_SP_COND_WAIT
Communication in progress (master reception)	R_IIC_COMMUNICATION	R_IIC_COMMUNICATION	R_IIC_MODE_READ	R_IIC_STS_ST_COND_WAIT
				R_IIC_STS_SEND_SLVADR_W_WAIT
				R_IIC_STS_SEND_SLVADR_R_WAIT
				R_IIC_STS_SEND_DATA_WAIT
				R_IIC_STS_RECEIVE_DATA_WAIT
				R_IIC_STS_SP_COND_WAIT
Communication in progress (master composite)	R_IIC_COMMUNICATION	R_IIC_COMMUNICATION	R_IIC_MODE_COMBINED	R_IIC_STS_ST_COND_WAIT
				R_IIC_STS_SEND_SLVADR_W_WAIT
				R_IIC_STS_SEND_SLVADR_R_WAIT
				R_IIC_STS_SEND_DATA_WAIT
				R_IIC_STS_RECEIVE_DATA_WAIT
				R_IIC_STS_SP_COND_WAIT
Error state	R_IIC_ERR_PARAM	R_IIC_ERR_PARAM	—	—
	R_IIC_ERR_AL	R_IIC_ERR_AL	—	—
	R_IIC_ERR_NON_REPLY	R_IIC_ERR_NON_REPLY	—	—
	R_IIC_ERR_SCL_GENCLK	R_IIC_ERR_SCL_GENCLK	—	—
	R_IIC_ERR_OTHER	R_IIC_ERR_OTHER	—	—

6.18 Function Specifications

6.18.1 Common Processing for These Functions

This sample code has an API that can be operated once. If this sample code's API is called during execution of this API processing, the processing is not performed and the function terminates. The value R_IIC_LOCK_FUNC is returned in this case.

An API flag is provided to prevent simultaneous calls the API. This flag is set while API processing is being performed. This mechanism operates as follows: at the start of API processing the flag is checked and the processing is only performed if the flag is not set. Figure 6-21 to Figure 6-24 present an overview of this processing as flowcharts.

This processing is performed for the functions defined in section 6.16. In section 6.18.2 and the following, we describe the processing for the "API function processing" in the figures starting with Figure 6-21.

Also note that if the μITRON OS is used, task management is performed using semaphores at both I²C driver initialization and at the start of communication. Figure 6-20 presents an example of semaphore operation.

In the I²C driver initialization function, a semaphore is acquired before initialization processing and then that semaphore is released after the initialization processing. During communication, the semaphore is acquired before the processing performed by the start function. When one of the start functions starts communication, it retains that semaphore, and the semaphore is released when communication is completed by the advance function or it terminates with an error.

The operation of the μITRON OS control processing has not been included. An example of the μITRON OS control is described in the flowchart of each function of the following. Refer to them and add the operation.

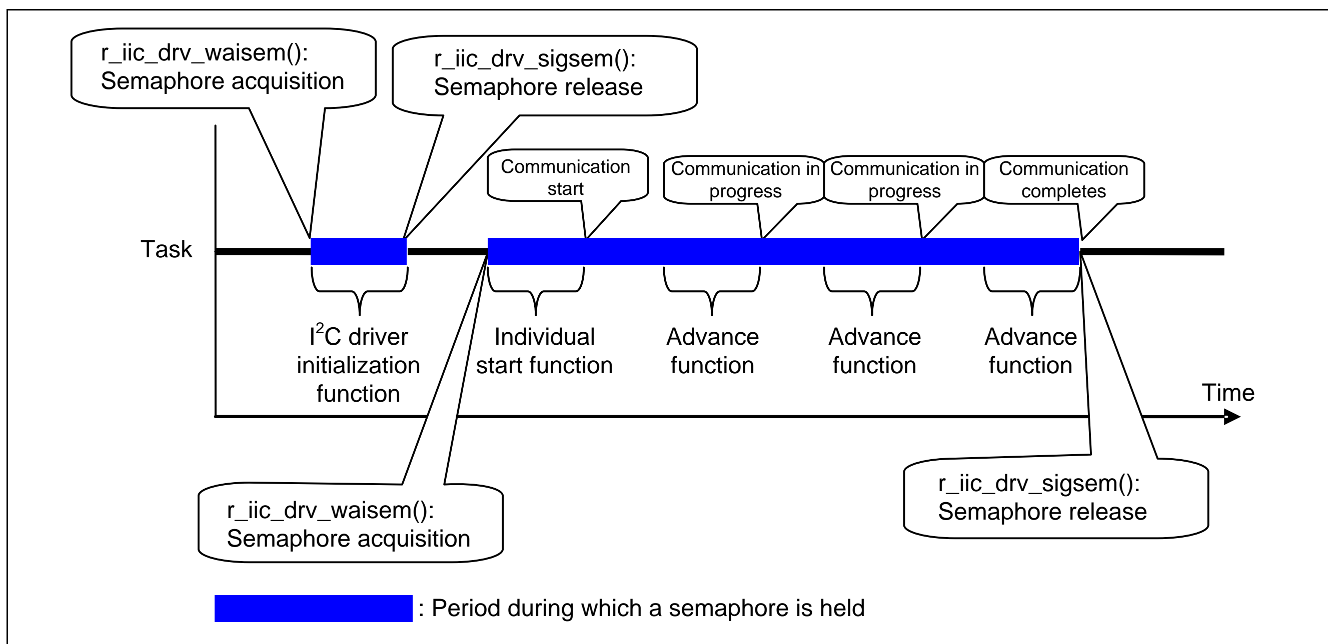


Figure 6-20 Semaphore Operation

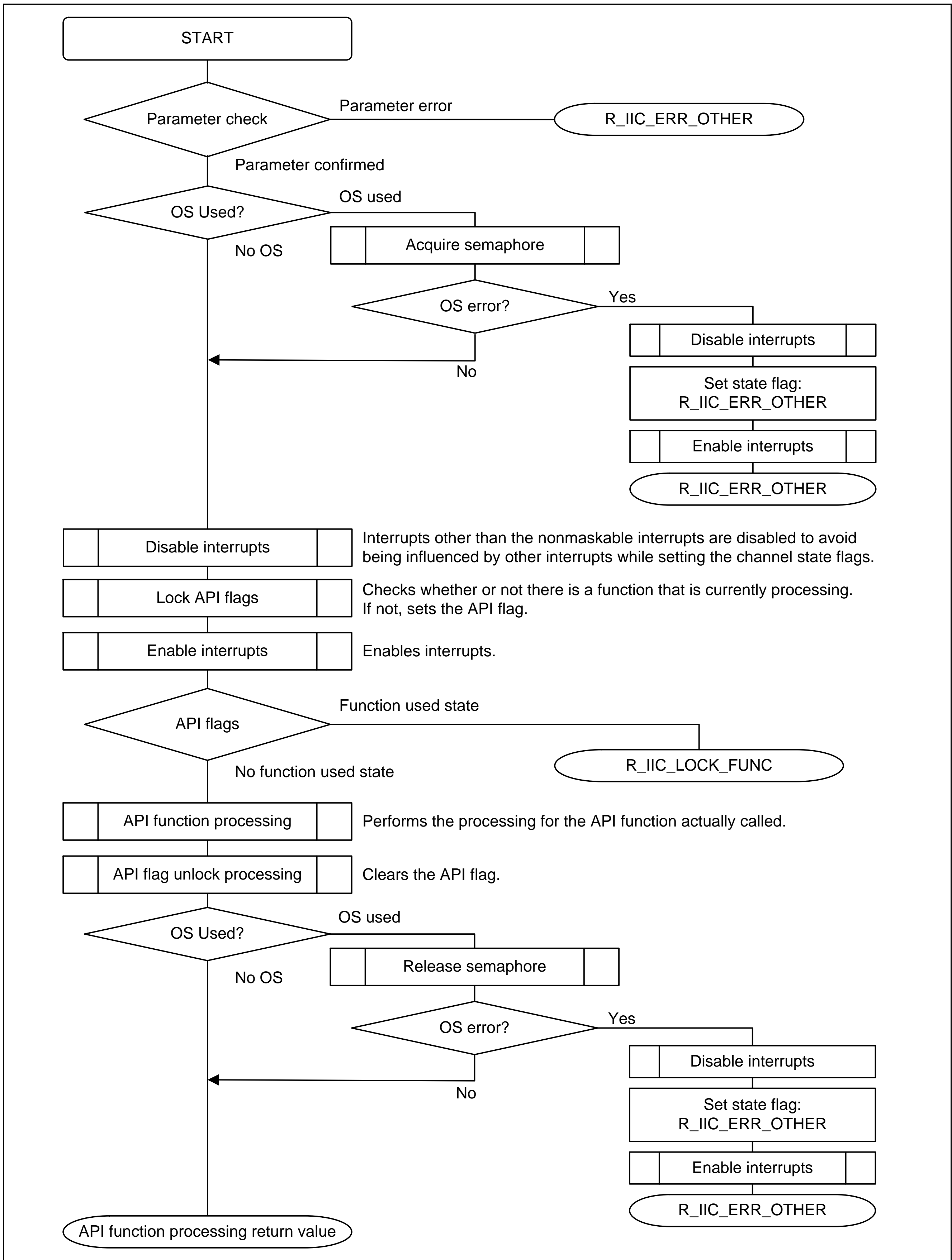


Figure 6-21 I²C Driver Initialization Function Multiple Call Prevention Flowchart

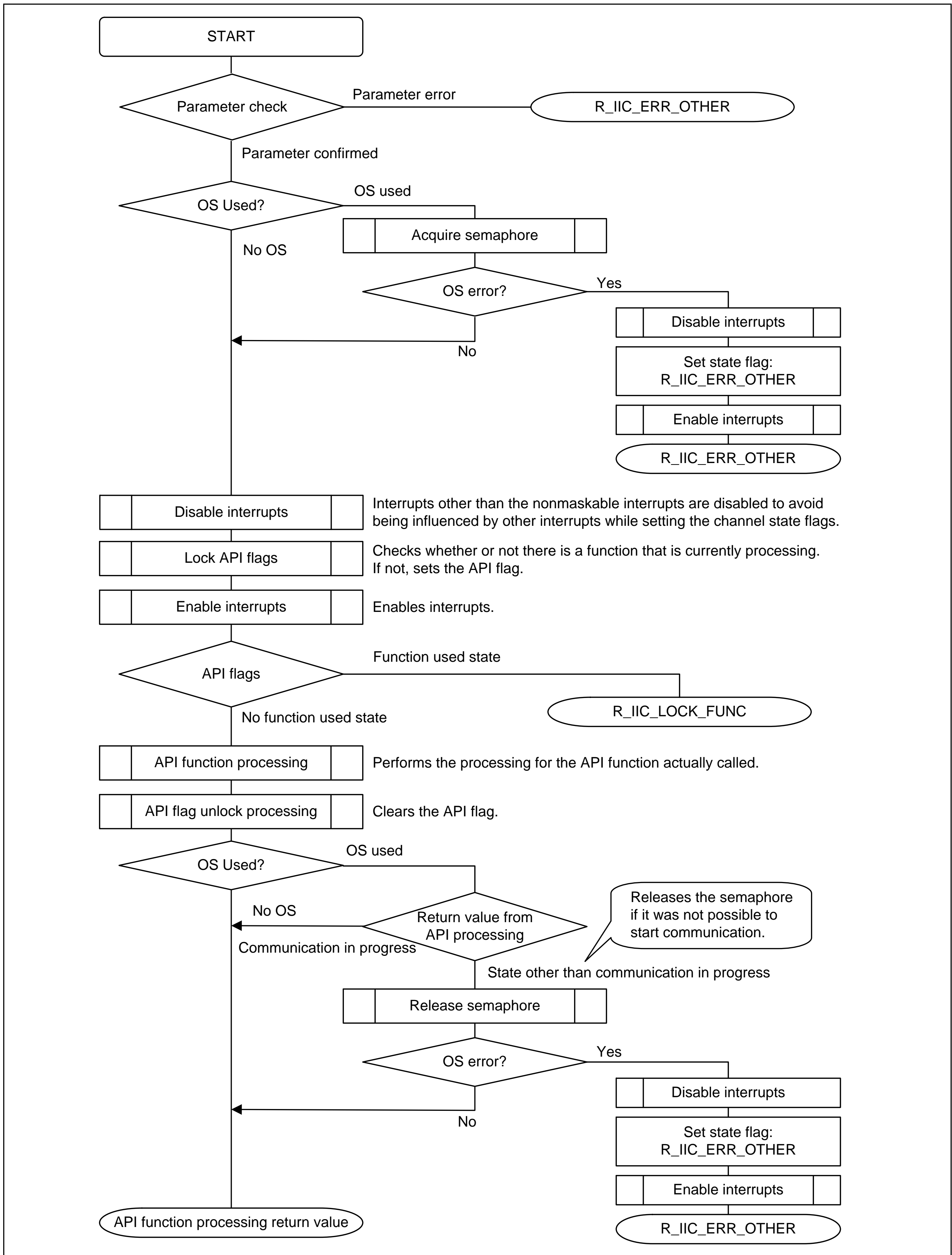


Figure 6-22 Individual Start Function Multiple Call Prevention Flowchart

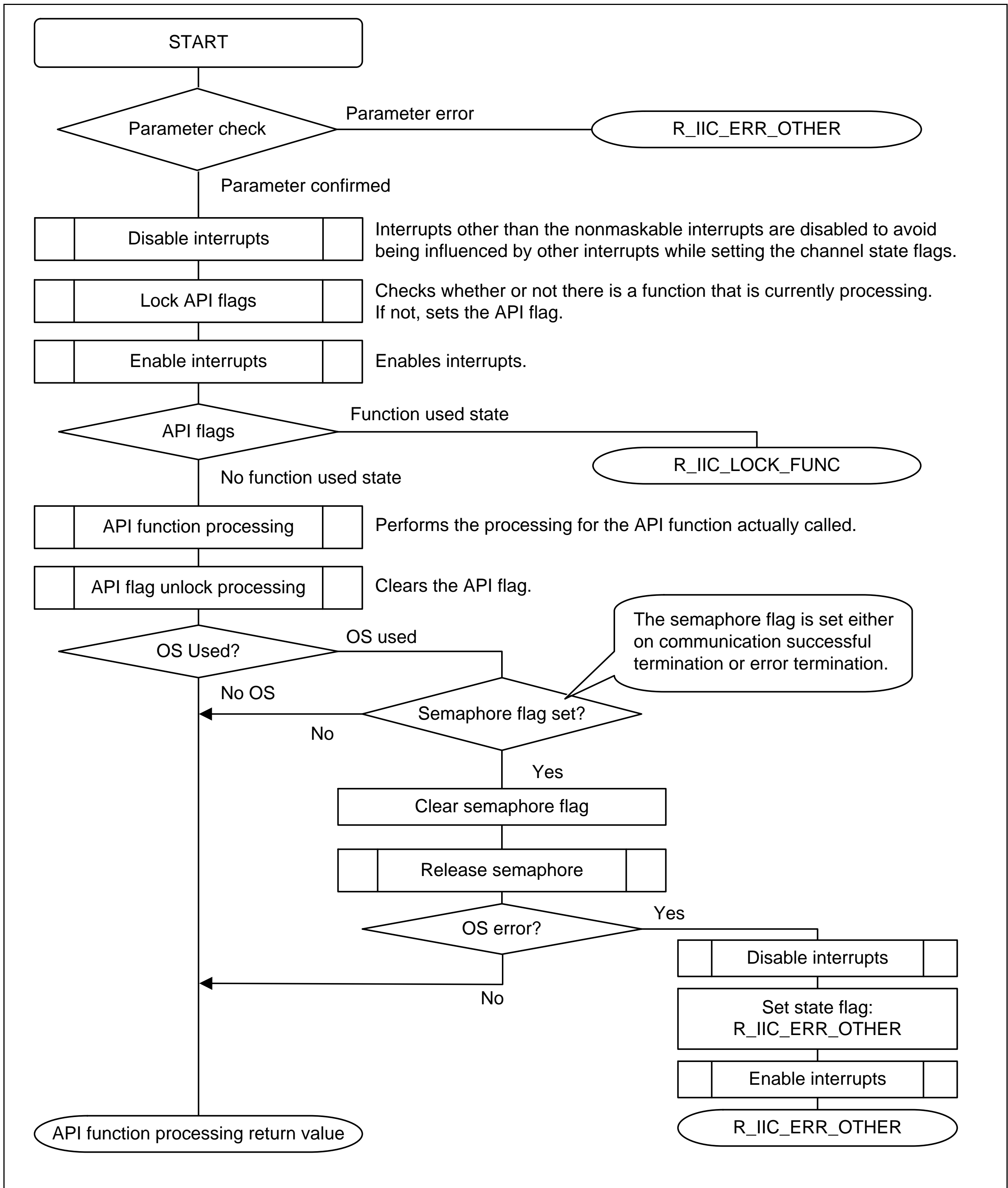


Figure 6-23 Advance Function Multiple Call Prevention Flowchart

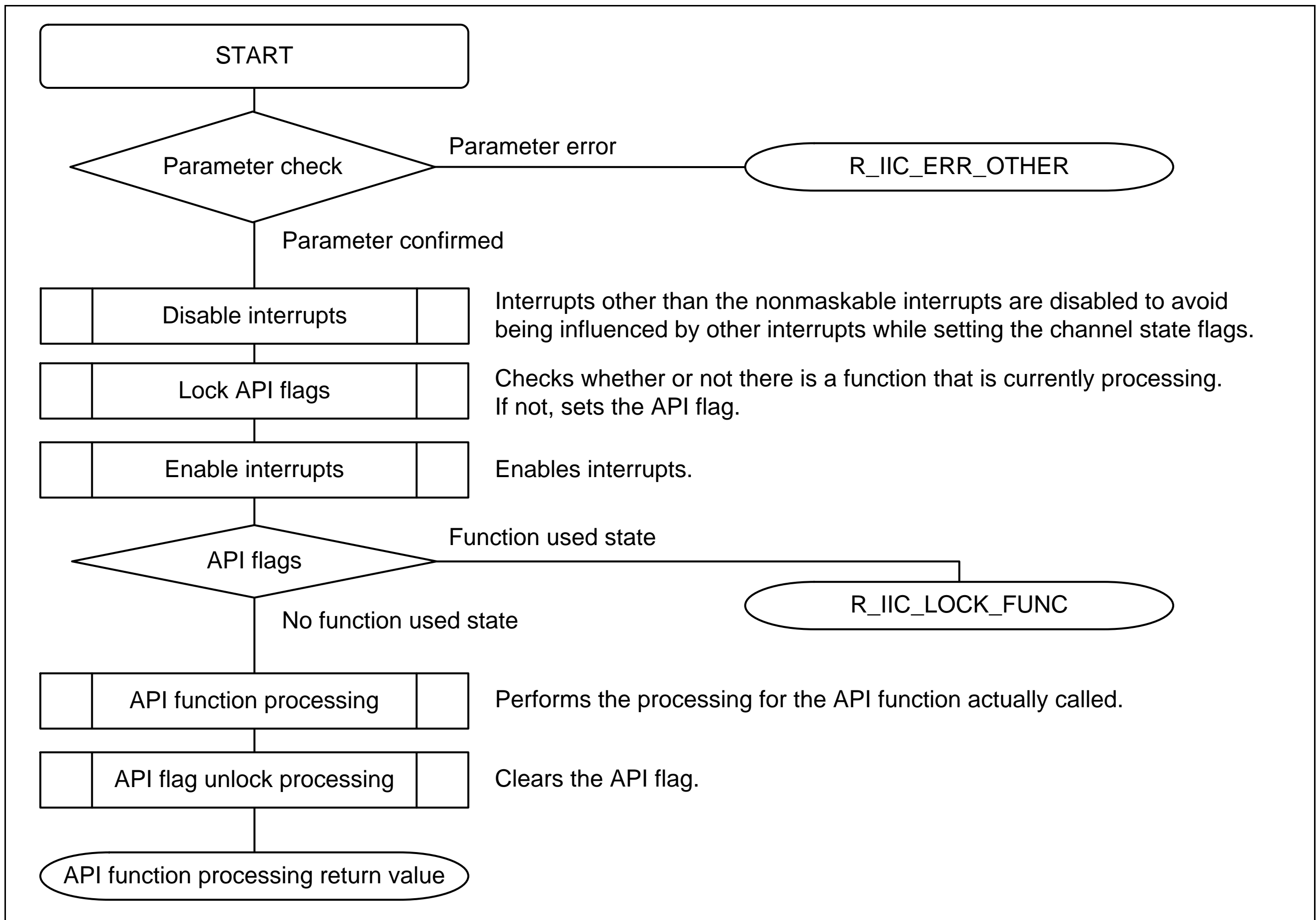


Figure 6-24 SCL Pseudo Clock Generation Function and I2C Driver Reset Function Multiple Call Prevention Flowchart

6.18.2 I²C Driver Initialization Function

R_IIC_Drv_Init

Outline I²C driver initialization function

Header r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h, r_iic_drv_os.h

Declaration error_t R_IIC_Drv_Init(r_iic_drv_info_t *pRIic_Info)

Description
- Initializes the corresponding channel.
- The following must be set up to use this function.
  The ChNo member of the r_iic_drv_info_t structure; The channel number used
  The channel state flag (g_iic_ChStatus[]); Sets R_IIC_NO_INIT*[1]
  The device state flag (*(pRIic_Info.pDevStatus)); Sets R_IIC_NO_INIT*[1]

Arguments r_iic_drv_info_t *pRIic_Info ; Pointer to I²C communication information structure

Return Value

R_IIC_IDLE
  In the channel uninitialized state, this function performs the initialization and transitions to the idle state. The channel state flag and device state flag are set to R_IIC_IDLE.
  In the already initialized state, initialization is not performed and the device state flag is set to R_IIC_IDLE.
  → Communication is now possible by calling the start function.

R_IIC_FINISH / R_IIC_NACK
  This is the result of executing the preprocessing. Since the start function can be called, initialization is not performed. The channel state flag and device state flag are not changed.
  → Communication is now possible by calling the start function.

R_IIC_LOCK_FUNC
  The processing was not performed because another API operation was being performed. The channel state flag and device state flag are not changed.
  → Call the function after processing of the other API finishes.

R_IIC_BUS_BUSY
  Communication is in progress. Initialization is not possible. The channel state flag and device state flag are not changed.
  → Call the advance function to terminate communication.

R_IIC_ERR_PARAM
  A parameter error was detected. The arguments were not set up. The channel state flag and device state flag are not changed.
  → Set up the arguments as required by this function.

R_IIC_ERR_AL
  Arbitration was lost. The channel state flag and device state flag are not changed.
  → See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_NON_REPLY
  A no replay error occurred. The channel state flag and device state flag are not changed.
  → See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_SDA_LOW_HOLD
  SDA is in the state where it has not recovered from the low-level hold state. The channel state flag and device state flag are not changed.
  → Check the system states, including whether a slave device is holding SDA low and whether a low-level signal has not been output from the master device.

R_IIC_ERR_OTHER
  Some other error occurred. The channel state flag and device state flag are set to R_IIC_ERR_OTHER.
  If an error had already occurred, no processing is performed. The channel state flag and device state flag are not changed.

→ Check the following items.

— Check that the I²C communication information structure is set up correctly.

— Check if the error occurred under OS control.

Remarks

- This function checks the parameters.
- It checks the channel state flag.
- It then performs initialization, including the following.
  Enables clock supply to the I²C peripheral hardware (I²C registers can be set)
  Sets ports to input mode and output latch to 0.
  Sets the transfer clock, local address, and start conditions
  Disables I²C interrupts.
  Initializes RAM.
- To avoid effects of other interrupts while setting the channel state flag, this function disables all interrupts other than the non-maskable interrupts.

Note: 1. Before calling the initialization function, set R_IIC_NO_INIT. If the initialization function is called without setting this, the initialization processing may not be performed.

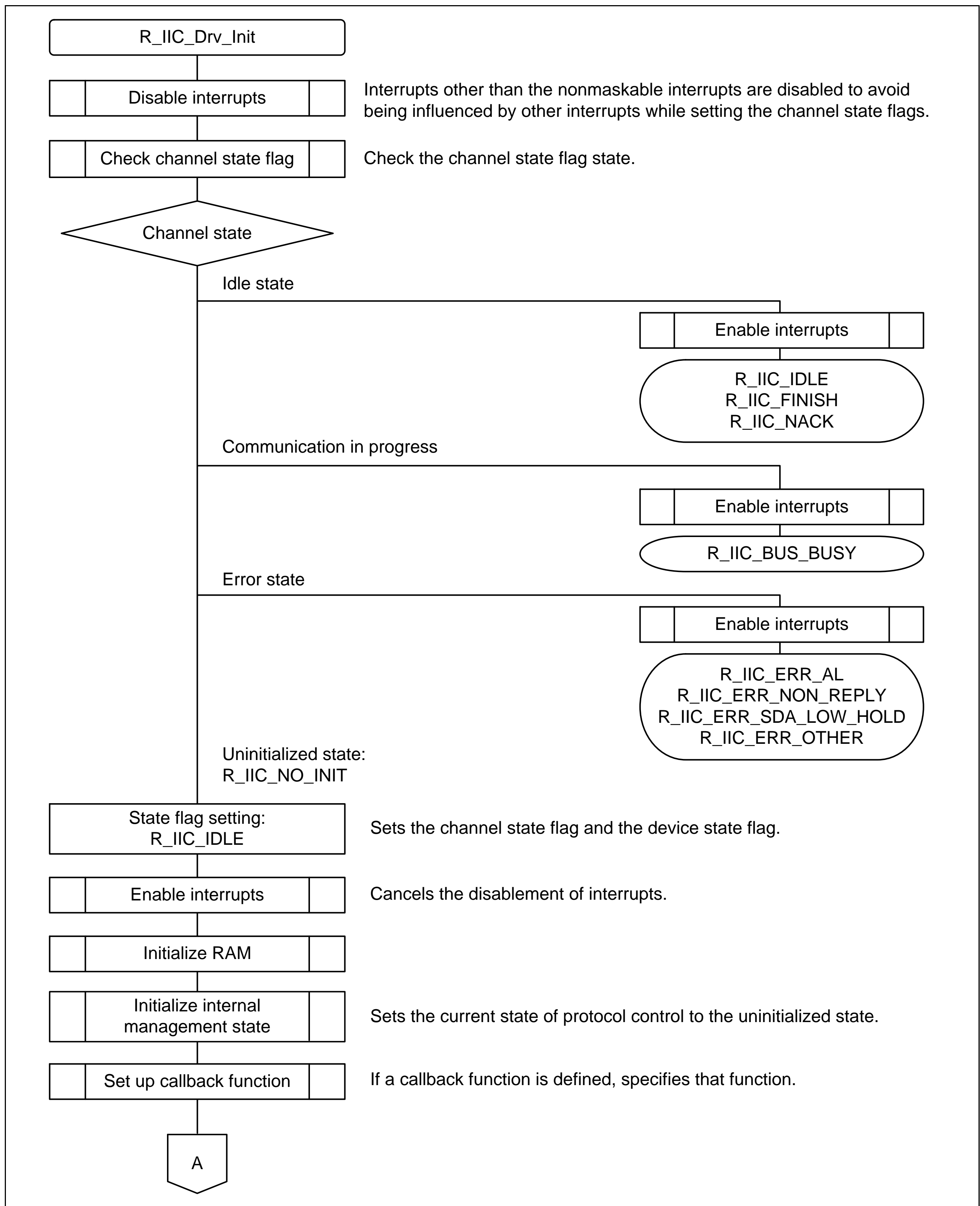


Figure 6-25 I²C Driver Initialization Function Overview Flowchart (1/2)

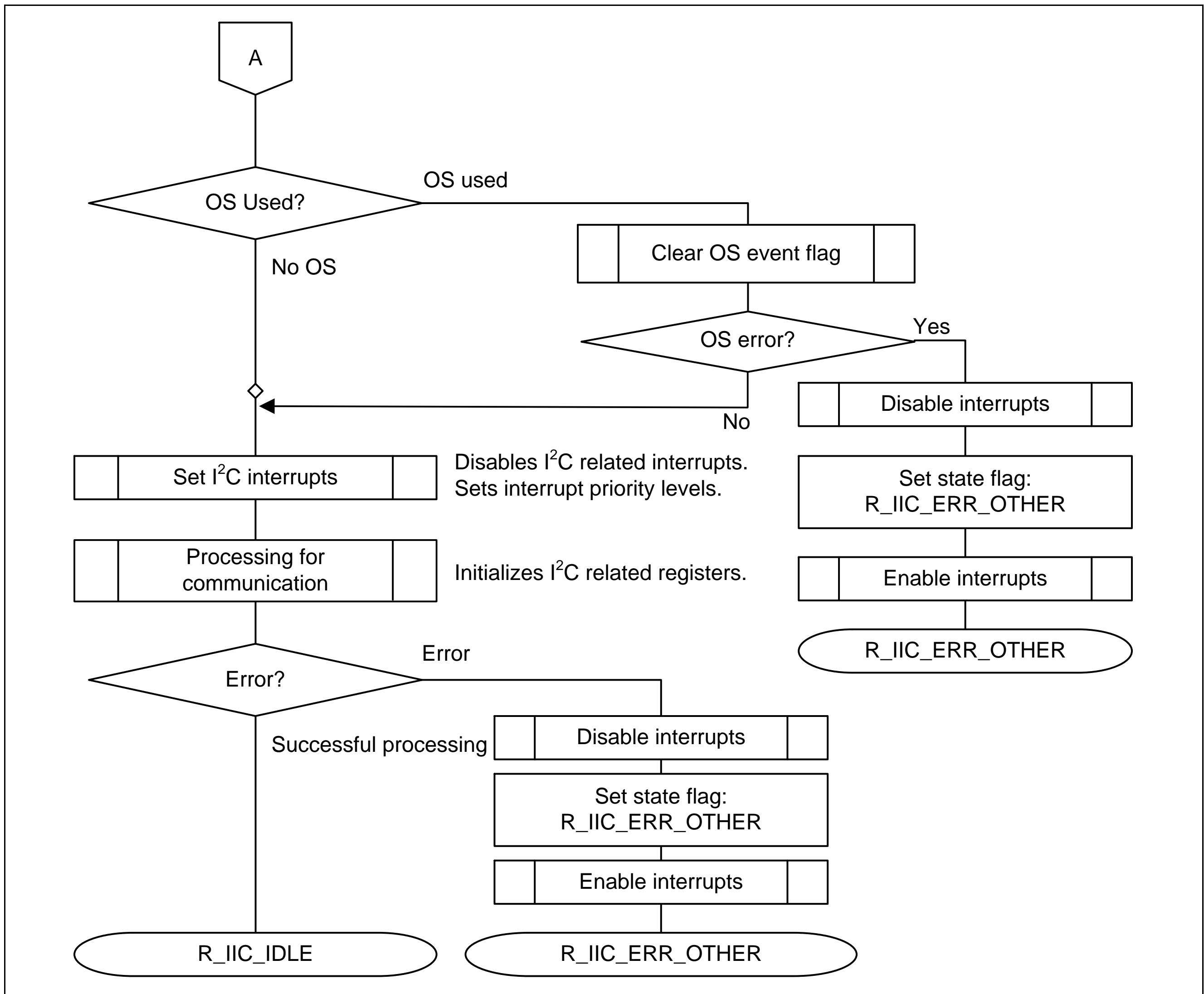


Figure 6-26 I²C Driver Initialization Function Overview Flowchart (2/2)

6.18.3 Master Transmission Start Function

R_IIC_Drv_MasterTx

Outline Master transmission start function

Header r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h, r_iic_drv_os.h

Declaration error_t R_IIC_Drv_MasterTx (r_iic_drv_info_t *pRIic_Info)

Description
- Starts master transmission.
- The r_iic_drv_info_t I²C communication information structure must be set up to perform this operation. See Table 6-16 for details on that setup.

Arguments r_iic_drv_info_t *pRIic_Info ; Pointer to I²C communication information structure

Return Value

R_IIC_COMMUNICATION
Master transmission started. The channel state flag and device state flag are set to R_IIC_COMMUNICATION.
→ Call the advance function to terminate communication.

R_IIC_NO_INIT
Initialization was not performed. The channel state flag and device state flag are not changed.
→ Call the initialization function and assure its processing has completed.

R_IIC_LOCK_FUNC
The processing was not performed because another API operation was being performed. The channel state flag and device state flag are not changed.
→ Call the function after processing of the other API finishes.

R_IIC_BUS_BUSY
Communication is in progress. It was not possible to start master transmission. The channel state flag and device state flag are not changed.
→ Call the advance function to terminate communication.

R_IIC_ERR_PARAM
A parameter error was detected. The arguments were not set up. The channel state flag and device state flag are not changed.
→ Set up the arguments as required by this function.

R_IIC_ERR_AL
Arbitration was lost. The channel state flag and device state flag are not changed.
→ See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_NON_REPLY
Either the bus was not released or it was not possible to detect the start condition. The channel state flag and device state flag are set to R_IIC_ERR_NON_REPLY.
→ See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_SDA_LOW_HOLD
SDA is in the state where it has not recovered from the low-level hold state. The channel state flag and device state flag are not changed.
→ Check the system states, including whether a slave device is holding SDA low and whether a low-level signal has not been output from the master device.

R_IIC_ERR_OTHER
Some other error occurred. The channel state flag and device state flag are set to R_IIC_ERR_OTHER.
If an error had already occurred, no processing is performed. The channel state flag and device state flag are not changed.
→ Check the following items.
— Check that the I²C communication information structure is set up correctly.
— Check if the error occurred under OS control.

Remarks

- This function checks the parameters.
- It checks the channel communication state. (Channel state flag: g_iic_ChStatus[])
- It checks the bus state (busy/released). (I^2C bus state flag in the IICA register)
- It then sets the I^2C protocol operating mode (g_iic_InternalInfo[].Mode) to R_IIC_MODE_WRITE (master transmission).
- It enables I^2C interrupts.
- It sets the ports to output mode.
- It generates a start condition.
- It transmits the slave address (transfer direction: write).
- It initializes the advance function counter.
- At the point this function returns, I^2C communication has not completed. The advance function must be called to terminate I^2C communication.
- The communication state after calling the start function can be checked with the return value from the advance function.
- To avoid effects of other interrupts while setting the channel state flag, this function disables all interrupts other than the non-maskable interrupts.

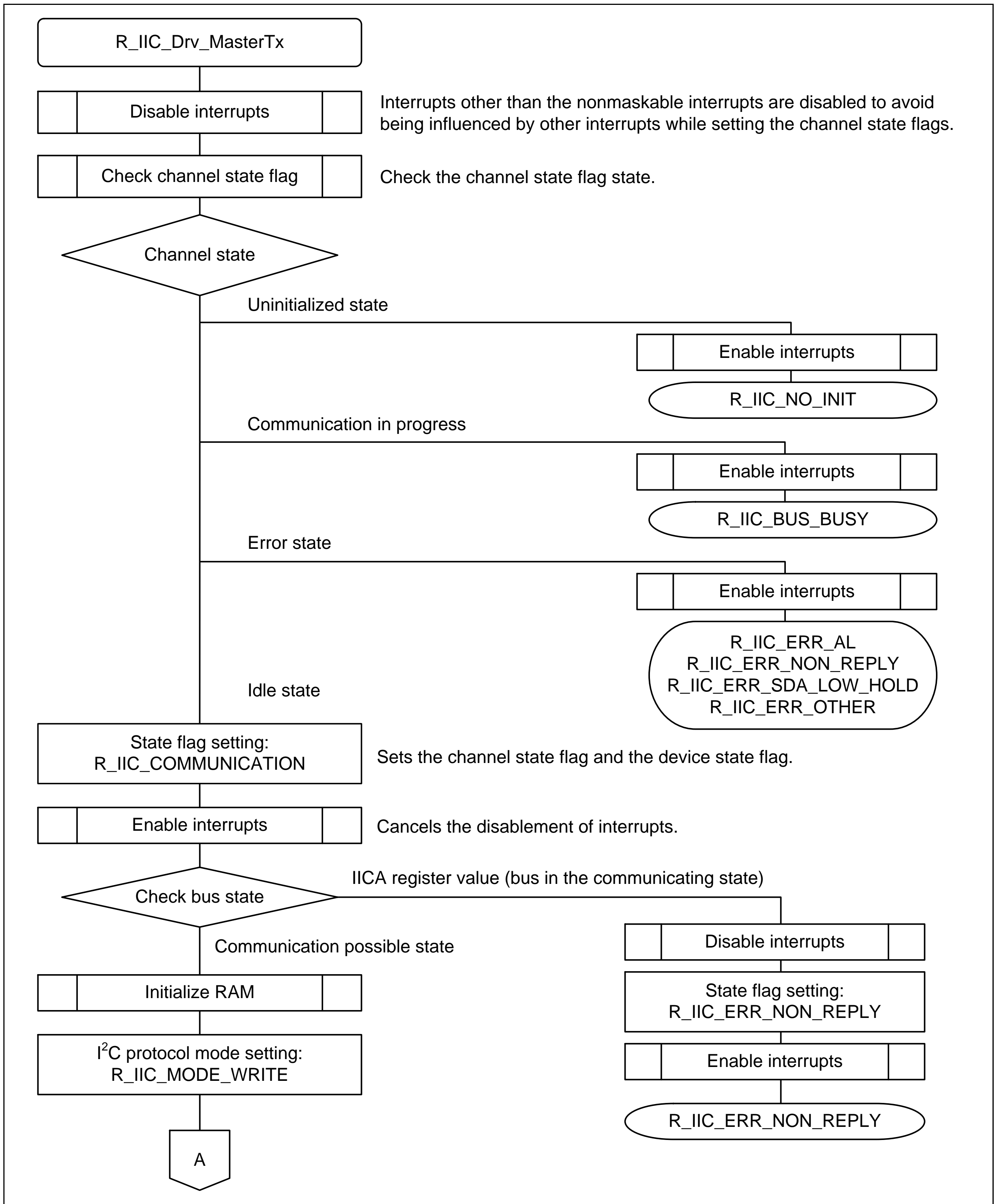


Figure 6-27 Master Transmission Start Function Overview Flowchart (1/2)

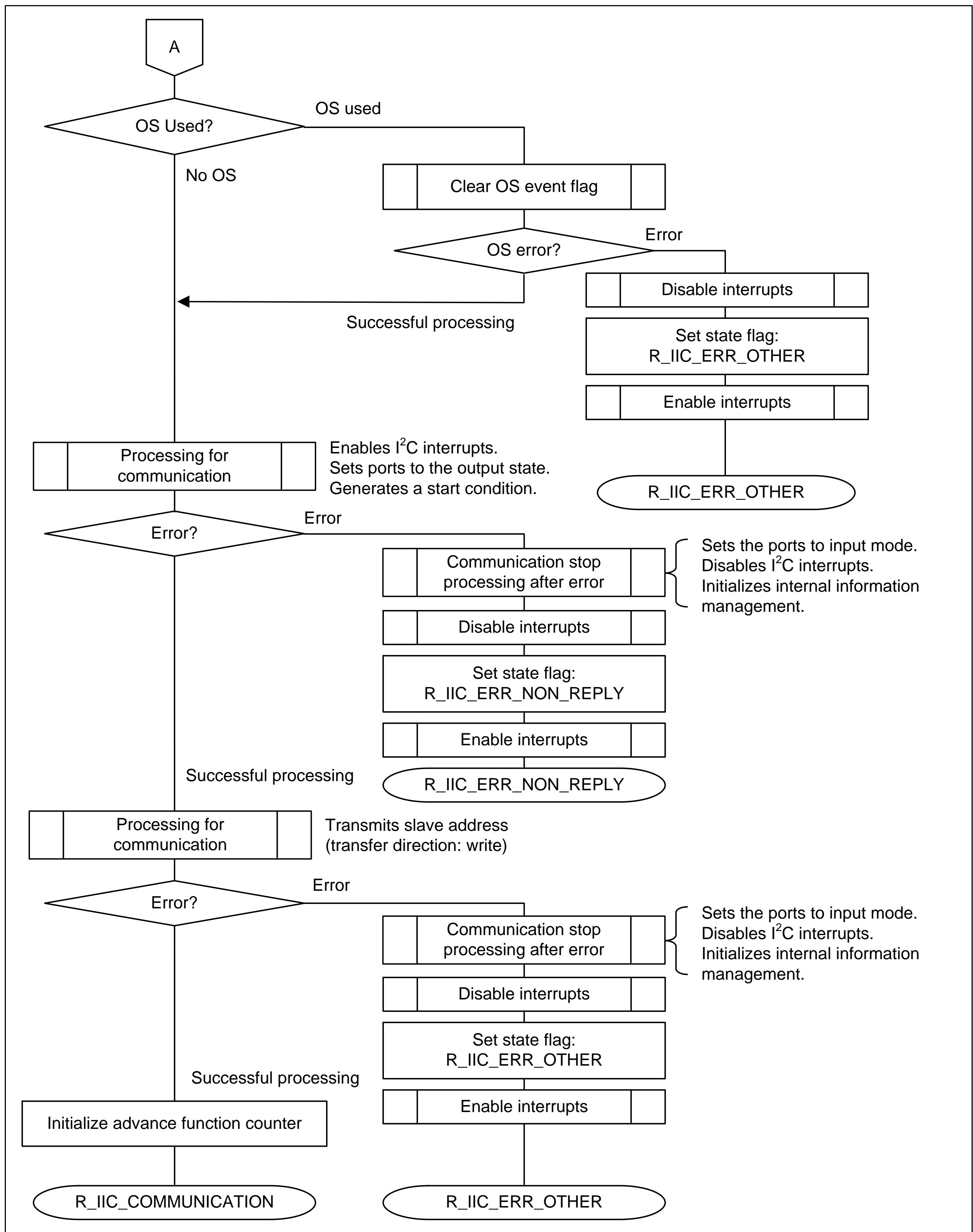


Figure 6-28 Master Transmission Start Function Overview Flowchart (2/2)

6.18.4 Master Reception Start Function

R_IIC_Drv_MasterRx

Outline Master reception start function

Header r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h, r_iic_drv_os.h

Declaration error_t R_IIC_Drv_MasterRx (r_iic_drv_info_t *pRIic_Info)

Description
- Starts master reception.
- The r_iic_drv_info_t I²C communication information structure must be set up to perform this operation. See Table 6-17 for details on that setup.

Arguments r_iic_drv_info_t *pRIic_Info ; Pointer to I²C communication information structure

Return Value

R_IIC_COMMUNICATION
Master reception started. The channel state flag and device state flag are set to R_IIC_COMMUNICATION.
→ Call the advance function to terminate communication.

R_IIC_NO_INIT
Initialization was not performed. The channel state flag and device state flag are not changed.
→ Call the initialization function and assure its processing has completed.

R_IIC_LOCK_FUNC
The processing was not performed because another API operation was being performed. The channel state flag and device state flag are not changed.
→ Call the function after processing of the other API finishes.

R_IIC_BUS_BUSY
Communication is in progress. It was not possible to start master reception. The channel state flag and device state flag are not changed.
→ Call the advance function to terminate communication.

R_IIC_ERR_PARAM
A parameter error was detected. The arguments were not set up. The channel state flag and device state flag are not changed.
→ Set up the arguments as required by this function.

R_IIC_ERR_AL
Arbitration was lost. The channel state flag and device state flag are not changed.
→ See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_NON_REPLY
Either the bus was not released or it was not possible to detect the start condition. The channel state flag and device state flag are set to R_IIC_ERR_NON_REPLY.
→ See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_SDA_LOW_HOLD
SDA is in the state where it has not recovered from the low-level hold state. The channel state flag and device state flag are not changed.
→ Check the system states, including whether a slave device is holding SDA low and whether a low-level signal has not been output from the master device.

R_IIC_ERR_OTHER
Some other error occurred. The channel state flag and device state flag are set to R_IIC_ERR_OTHER.
If an error had already occurred, no processing is performed. The channel state flag and device state flag are not changed.
→ Check the following items.
— Check that the I²C communication information structure is set up correctly.
— Check if the error occurred under OS control.

Remarks

- This function checks the parameters.
- It checks the channel communication state. (Channel state flag: g_iic_ChStatus[])
- It checks the bus state (busy/released). (I²C bus state flag in the IICA register)
- It then sets the I²C protocol operating mode (g_iic_InternalInfo[].Mode) to R_IIC_MODE_READ (master reception).
- It enables I²C interrupts.
- It sets the ports to output mode.
- It generates a start condition.
- It transmits the slave address (transfer direction: read).
- It initializes the advance function counter.
- At the point this function returns, I²C communication has not completed. The advance function must be called to terminate I²C communication.
- The communication state after calling the start function can be checked with the return value from the advance function.
- To avoid effects of other interrupts while setting the channel state flag, this function disables all interrupts other than the non-maskable interrupts.

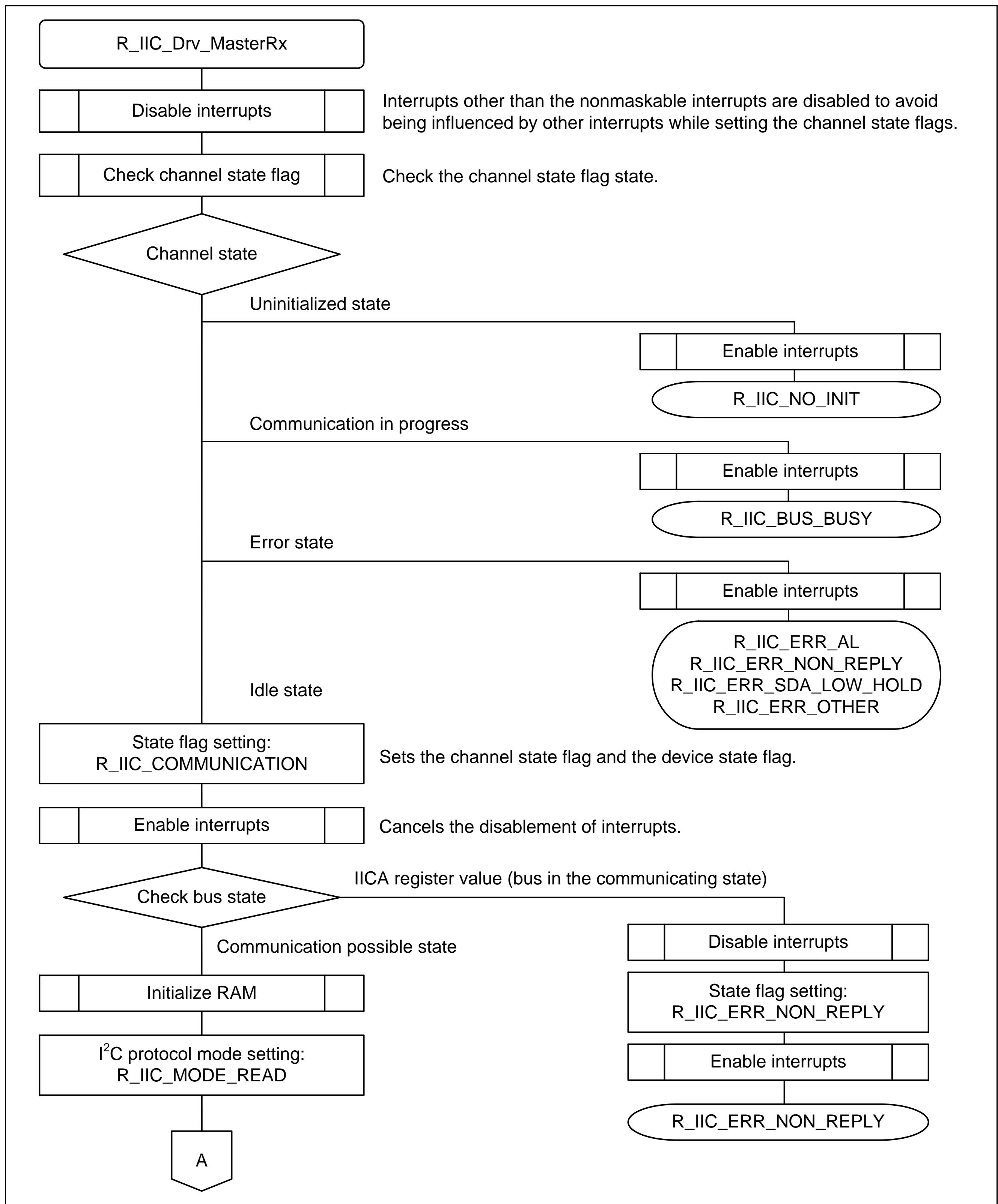


Figure 6-29 Master Reception Start Function Overview Flowchart (1/2)

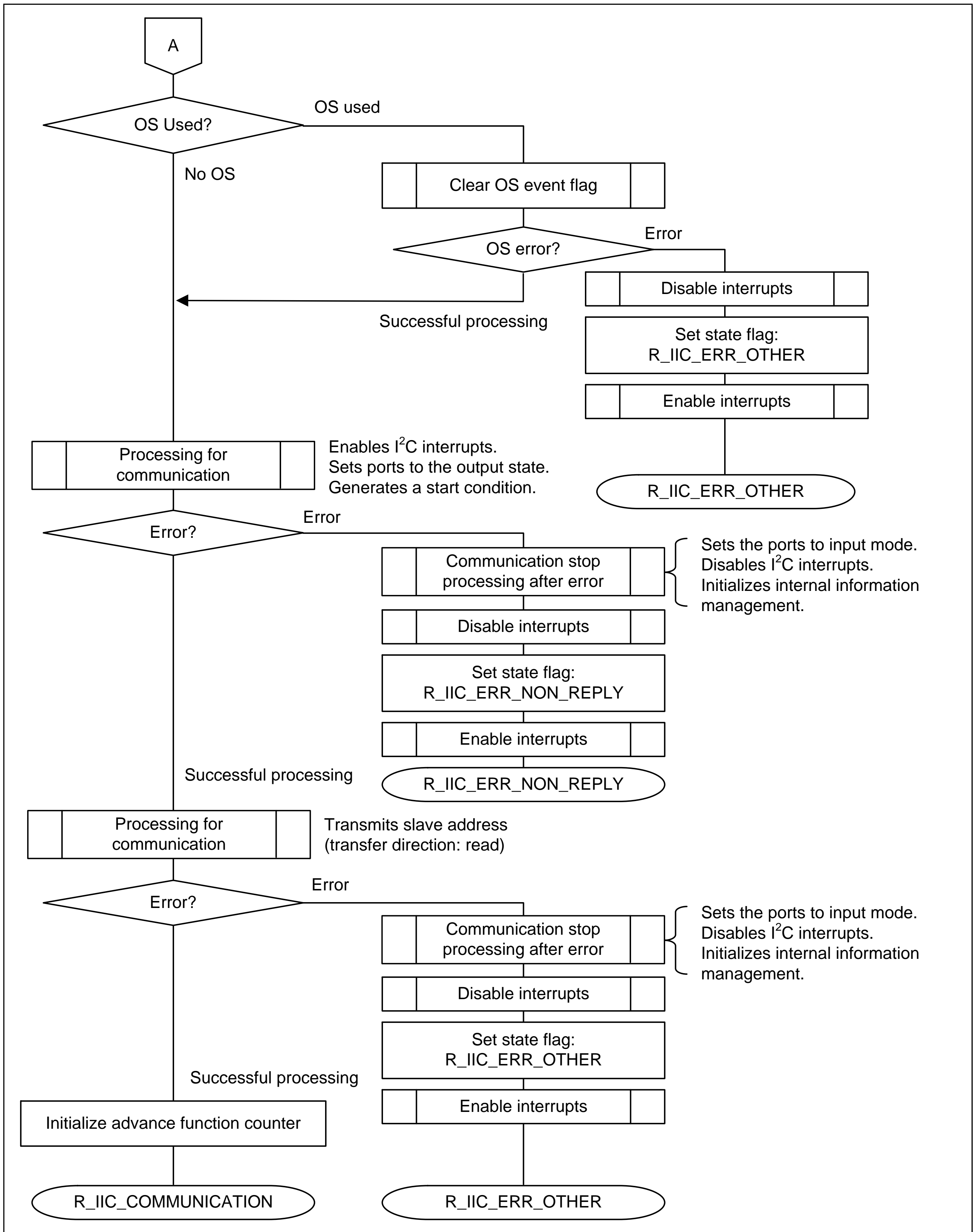


Figure 6-30 Master Reception Start Function Overview Flowchart (2/2)

6.18.5 Master Composite Start Function

R_IIC_Drv_MasterTRx

Outline Master composite start function

Header r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h, r_iic_drv_os.h

Declaration error_t R_IIC_Drv_MasterTRx (r_iic_drv_info_t *pRIic_Info)

Description
- Starts master composite communication.
- The r_iic_drv_info_t I²C communication information structure must be set up to perform this operation. See Table 6-17 for details on that setup.

Arguments r_iic_drv_info_t *pRIic_Info ; Pointer to I²C communication information structure

Return Value

R_IIC_COMMUNICATION

Master composite communication was started. The channel state flag and device state flag are set to R_IIC_COMMUNICATION.

→ Call the advance function to terminate communication.

R_IIC_NO_INIT

Initialization was not performed. The channel state flag and device state flag are not changed.

→ Call the initialization function and assure its processing has completed.

R_IIC_LOCK_FUNC

The processing was not performed because another API operation was being performed. The channel state flag and device state flag are not changed.

→ Call the function after processing of the other API finishes.

R_IIC_BUS_BUSY

Communication is in progress. It was not possible to start master composite communication. The channel state flag and device state flag are not changed.

→ Call the advance function to terminate communication.

R_IIC_ERR_PARAM

A parameter error was detected. The arguments were not set up. The channel state flag and device state flag are not changed.

→ Set up the arguments as required by this function.

R_IIC_ERR_AL

Arbitration was lost. The channel state flag and device state flag are not changed.

→ See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_NON_REPLY

Either the bus was not released or it was not possible to detect the start condition. The channel state flag and device state flag are set to R_IIC_ERR_NON_REPLY.

→ See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_SDA_LOW_HOLD

SDA is in the state where it has not recovered from the low-level hold state. The channel state flag and device state flag are not changed.

→ Check the system states, including whether a slave device is holding SDA low and whether a low-level signal has not been output from the master device.

R_IIC_ERR_OTHER

Some other error occurred. The channel state flag and device state flag are set to R_IIC_ERR_OTHER.

If an error had already occurred, no processing is performed. The channel state flag and device state flag are not changed.

→ Check the following items.

— Check that the I²C communication information structure is set up correctly.

— Check if the error occurred under OS control.

Remarks

- This function checks the parameters.
- It checks the channel communication state. (Channel state flag: g_iic_ChStatus[])
- It checks the bus state (busy/released). (I²C bus state flag in the IICA register)
- It then sets the I²C protocol operating mode (g_iic_InternalInfo[].Mode) to R_IIC_MODE_COMBINED (master composite).
- It enables I²C interrupts.
- It sets the ports to output mode.
- It generates a start condition.
- It transmits the slave address (transfer direction: write).
- It initializes the advance function counter.
- At the point this function returns, I²C communication has not completed. The advance function must be called to terminate I²C communication.
- The communication state after calling the start function can be checked with the return value from the advance function.
- To avoid effects of other interrupts while setting the channel state flag, this function disables all interrupts other than the non-maskable interrupts.

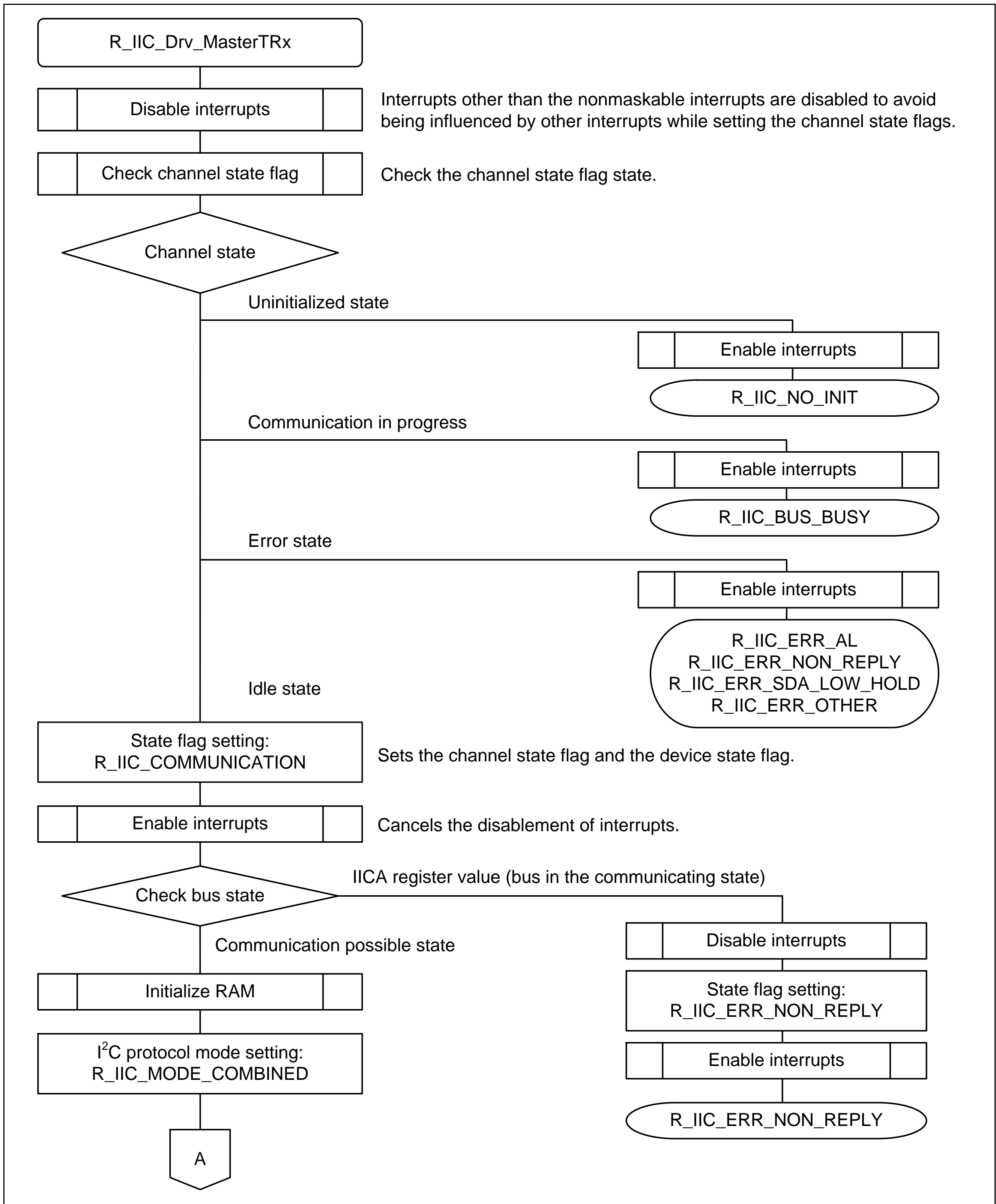


Figure 6-31 Master Composite Start Function Overview Flowchart (1/2)

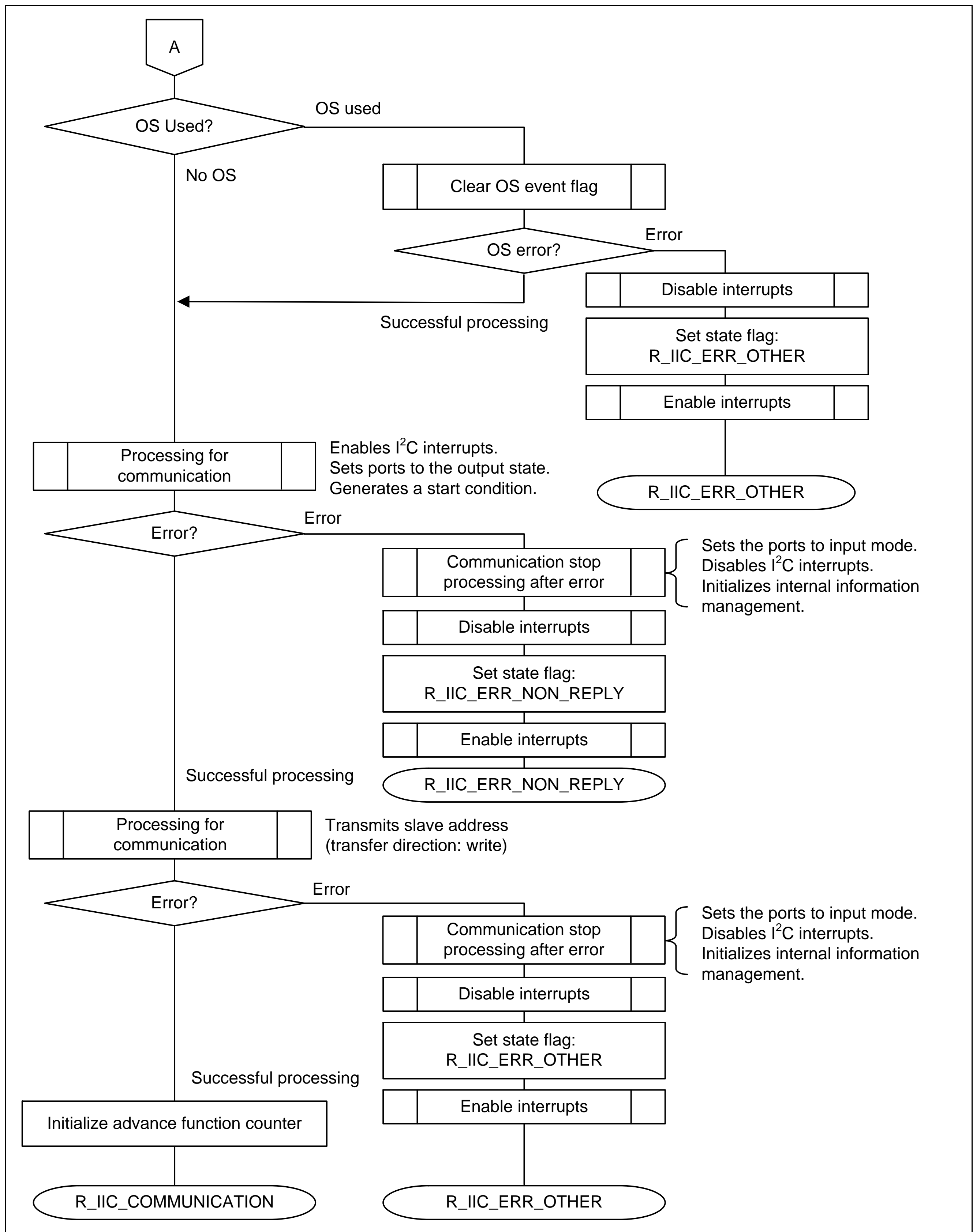


Figure 6-32 Master Composite Start Function Overview Flowchart (2/2)

6.18.6 Advance Function

R_IIC_Drv_Advance

Outline Advance function

Header r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h, r_iic_drv_os.h

Declaration error_t R_IIC_Drv_Advance (r_iic_drv_info_t *pRIic_Info)

Description
- Monitors the communication and performs processing to advance communication. Returns the communication state in the return value.
- It is necessary to terminate communication with the advance function to start the next communication.

Arguments r_iic_drv_info_t *pRIic_Info ; Pointer to I^{2}C communication information structure

Return Value

R_IIC_COMMUNICATION

Communication is in progress. The channel state flag and device state flag are not changed.

→ Call the advance function to terminate communication.

R_IIC_FINISH

All communication completed successfully. The channel state flag and device state flag are set to R_IIC_FINISH.

Performs no processing if communication had already terminated. The channel state flag and device state flag are not changed.

→ Communication is now possible by calling the start function.

R_IIC_NACK

NACK was detected. A stop condition was generated and communication terminated. The channel state flag and device state flag are set to R_IIC_NACK.

Performs no processing if communication had already terminated. The channel state flag and device state flag are not changed.

→ Communication is now possible by calling the start function.

R_IIC_NO_INIT

Initialization was not performed. The channel state flag and device state flag are not changed.

→ Call the initialization function and assure its processing has completed.

R_IIC_IDLE

The system is in the idle state. The channel state flag and device state flag are not changed.

→ Communication is now possible by calling the start function.

R_IIC_LOCK_FUNC

The processing was not performed because another API operation was being performed. The channel state flag and device state flag are not changed.

→ Call the function after processing of the other API finishes.

R_IIC_BUS_BUSY

The requested processing was not performed because another device was communicating on the same channel. The channel state flag and device state flag are not changed.

→ Terminate the communication with the other device.

R_IIC_ERR_PARAM

A parameter error was detected. The arguments were not set up. The channel state flag and device state flag are not changed.

If an error had already occurred, no processing is performed. The channel state flag and device state flag are not changed.

→ Set up the arguments as required by this function.

R_IIC_ERR_AL

Arbitration was lost. The channel state flag and device state flag are set to R_IIC_ERR_AL.

If an error had already occurred, no processing is performed. The channel state flag

and device state flag are not changed.

→ See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_NON_REPLY

The following occurred. The channel state flag and device state flag are set to R_IIC_ERR_NON_REPLY.

— The number of calling the advance function exceeded the limit.

— Although stop condition generation processing was performed, a stop condition was not detected within a fixed period.

If an error had already occurred, no processing is performed. The channel state flag and device state flag are not changed.

→ SDA or SCL may have been held low due to noise or some other problem. See section 7.6.Recovery Processing Example, and perform that recovery processing.

R_IIC_ERR_SDA_LOW_HOLD

SDA is in the state where it has not recovered from the low-level hold state. The channel state flag and device state flag are not changed.

→ Check the system states, including whether a slave device is holding SDA low and whether a low-level signal has not been output from the master device.

R_IIC_ERR_OTHER

Some other error occurred. The channel state flag and device state flag are set to R_IIC_ERR_OTHER.

If an error had already occurred, no processing is performed. The channel state flag and device state flag are not changed.

→ Check the following items.

— Check that the I²C communication information structure is set up correctly.

— Check if the error occurred under OS control.

Remarks

- This function checks the parameters.
- If the event flag is set (g_iic_Event[]), the following processing is performed.
  The advance function counter (g_iic_ReplyCnt[]) is initialized.
  Communication advance processing is performed.
  If the processing proceeded successfully, the function checks whether all communication completed. When all communication has completed, the channel state flag is set to R_IIC_FINISH.
- If the event flag is no set (g_iic_Event[]), the following processing is performed.
  The advance function counter (g_iic_ReplyCnt[]) is decremented.
  If the advance function counter is 0, the return value is set to R_IIC_ERR_NON_REPLY.
- To avoid effects of other interrupts while setting the channel state flag, this function disables all interrupts other than the non-maskable interrupts.

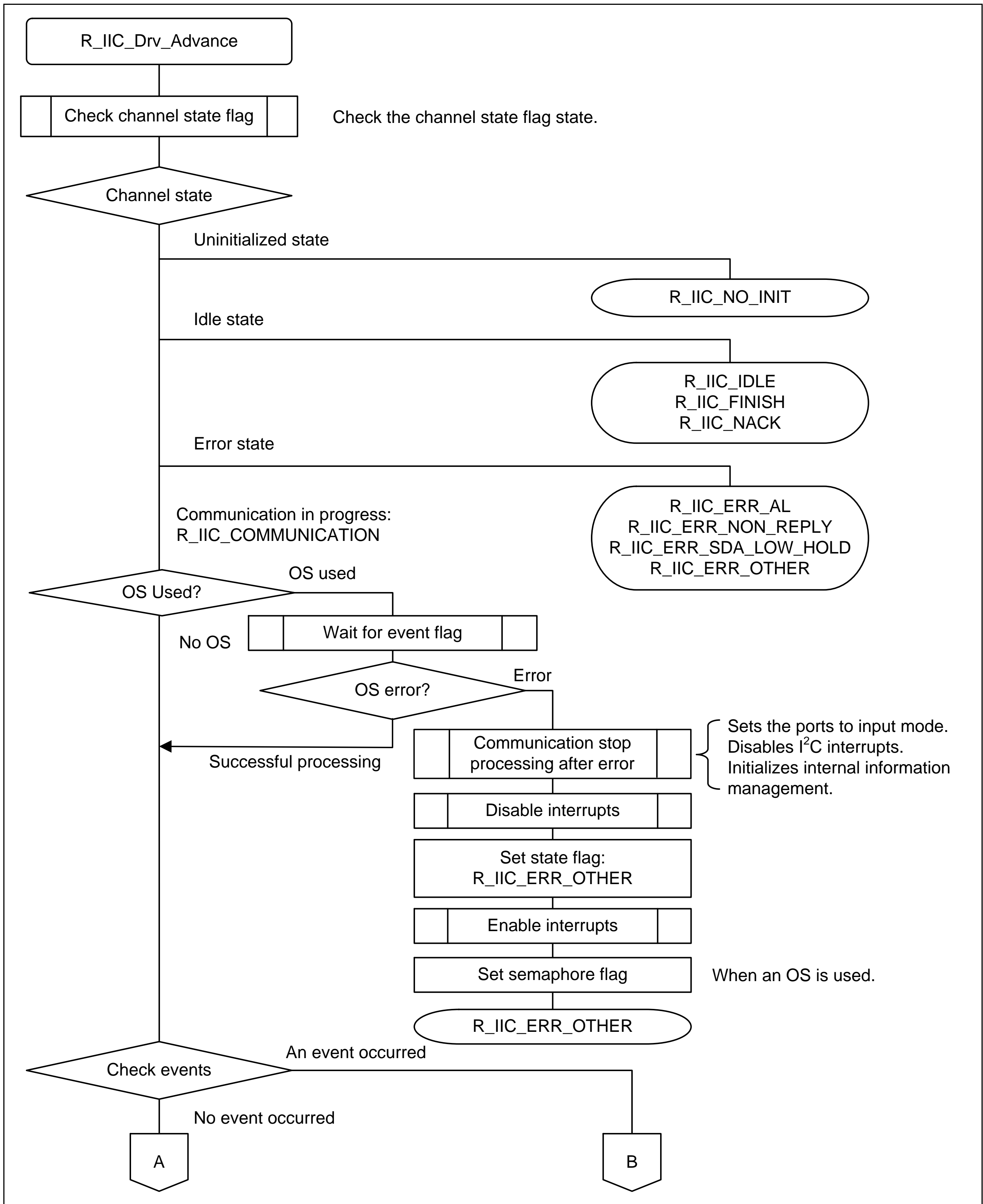


Figure 6-33 Advanced Function Overview Flowchart (1/3)

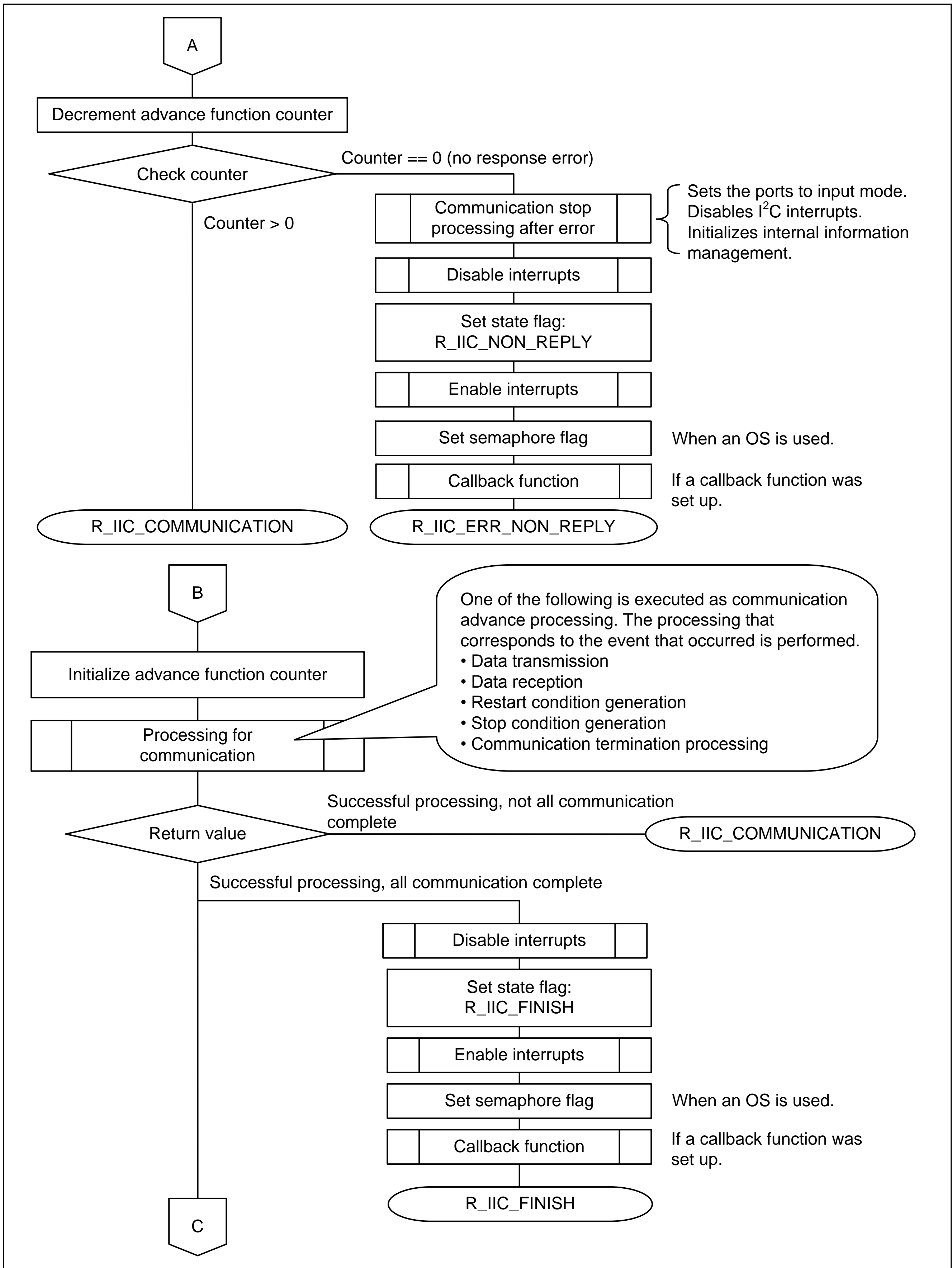


Figure 6-34 Advanced Function Overview Flowchart (2/3)

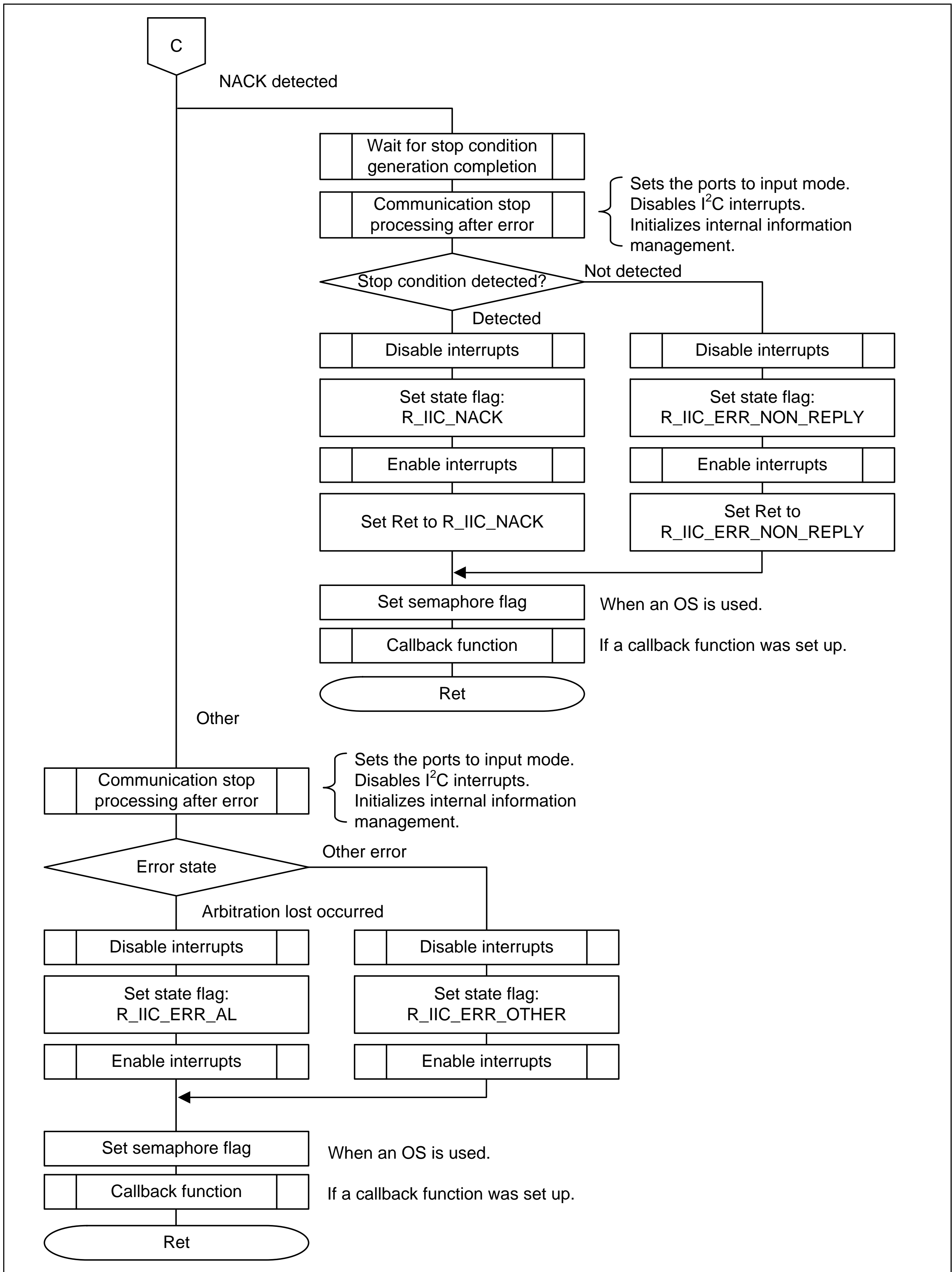


Figure 6-35 Advanced Function Overview Flowchart (3/3)

6.18.7 SCL Pseudo Clock Generation Function

R_IIC_Drv_GenClk

Outline SCL pseudo clock generation function

Header r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h, r_iic_drv_os.h

Declaration error_t R_IIC_Drv_GenClk (r_iic_drv_info_t *pRIic_Info, uint8_t ClkCnt)

Description

- This function generates an SCL pseudo clock. If a synchronization discrepancy occurs between the master and slave due to noise or other problem and SDA is held at the low level, this function can correct the internal state of the slave.
- Do not use this function in normal states. Use of this function during normal operation can result in communication problems.
- The following must be set up to use this function.
  The ChNo member of the r_iic_drv_info_t structure; The channel number used
  The clock count ClkCnt; 01h to FFh
  The clock high and low-level width; See note 1.

Arguments

r_iic_drv_info_t *pRIic_Info ; Pointer to I^{2}C communication information structure
uint8_t ClkCnt ; SCL clock count

Return Value

R_IIC_NO_INIT
The SDA line has gone to the high level, correction of the internal state of the slave device completed, and the system is in the uninitialized state. The channel state flag and device state flag are set to R_IIC_NO_INIT.
→ Perform the following operations to restart communication.
(1) Call the initialization function
(2) Call master transmission with pattern 4
(3) Terminate communication by calling the advance function.

R_IIC_LOCK_FUNC
The processing was not performed because another API operation was being performed. The channel state flag and device state flag are not changed.
→ Call the function after processing of the other API finishes.

R_IIC_ERR_PARAM
A parameter error was detected. The arguments were not set up. The channel state flag and device state flag are not changed.
→ Set up the arguments as required by this function.

R_IIC_ERR_SDA_LOW_HOLD
Although an SCL pseudo clock was generated, SDA remains in the low hold state. The channel state flag and device state flag are set to R_IIC_ERR_SDA_LOW_HOLD.
→ Check the system states, including whether a slave device is holding SDA low and whether a low-level signal has not been output from the master device.

Remarks

- If SDA is at the low level when SDA is set to the high-impedance state, the bus will be seen as not having been released.
- When SDA is low, the SCL pin is switched to port output, and a clock (low->high) is input to the bus until SDA goes high.
- An error is returned if SDA remains low when the set number of clock cycles have been generated.
- Since it is common for communication units to consist of 9 clock cycles, we recommend setting the number of clock cycles to at least 9 cycles.
- To avoid effects of other interrupts while setting the channel state flag, this function disables all interrupts other than the non-maskable interrupts.

Note: 1. Notes on the pseudo clock output transfer rate
The width of the high and low periods in the pseudo clock output must meet or exceed the minimum value as stipulated in the I^{2}C-bus specification. (See the following table.)
The high and low-level width set wait times are implemented by software

looping. Thus the wait time will differ depending on the system clock used.
The high and low-level width set wait times are set with the definitions of the SCL_L_WAIT and SCL_H_WAIT macros. The user must manage the values of these values to meet the I^2C-bus specification according to the system clock used. (See Table 6-13 in section 6.12.1 for the macro definitions.)

Table High and Low-Level Width Minimum Values Stipulated in the I^2C-Bus Specification

	Fast mode	Standard mode
Low-level width (tLow)	1.3 µs	4.7 µs
High-level width (tLow)	0.6 µs	4.0 µs

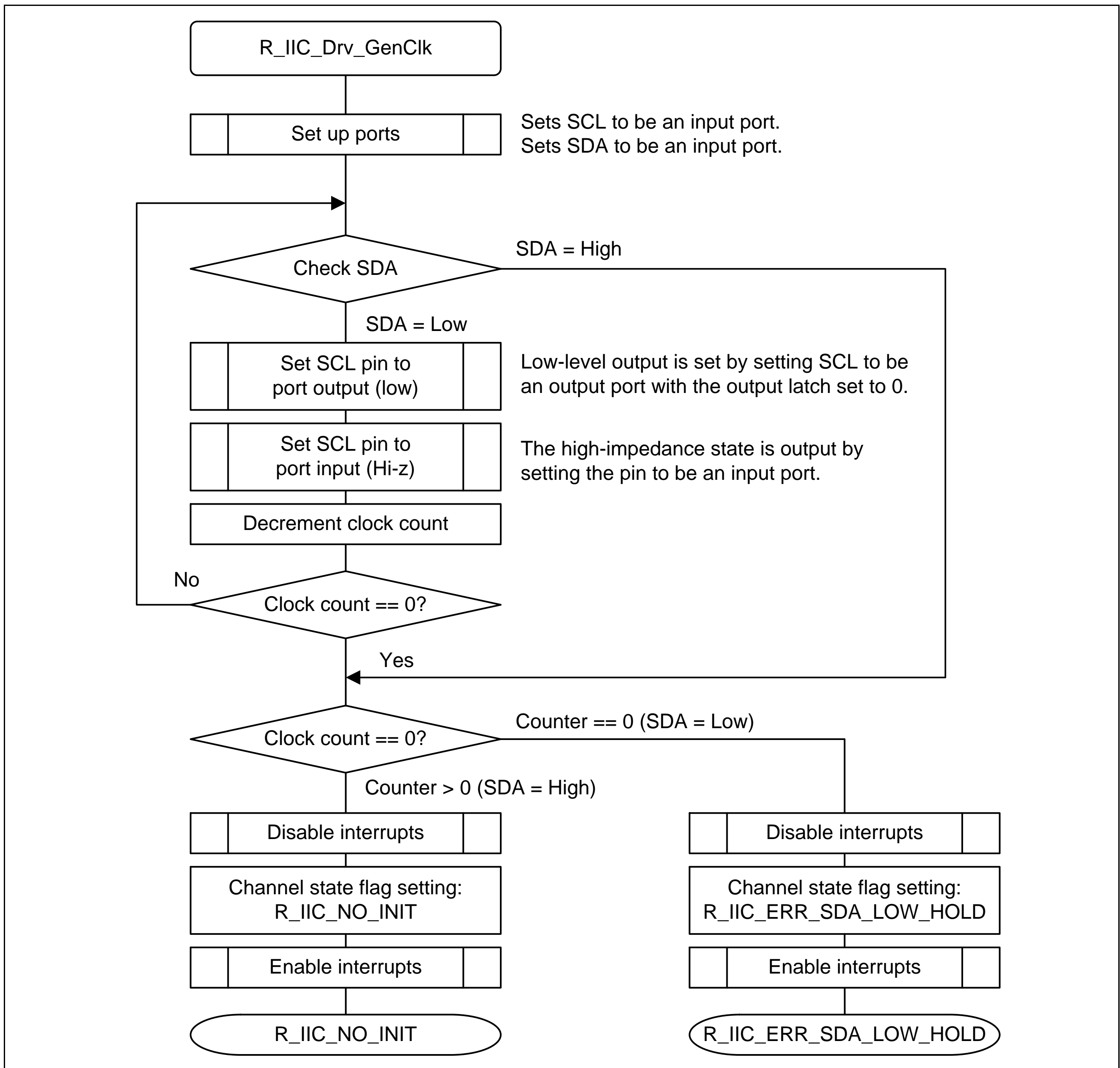


Figure 6-36 SCL Pseudo-Clock Generation Function Overview Flowchart

6.18.8 I²C Driver Reset Function

R_IIC_Drv_Reset

Outline I²C driver reset function

Header r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h, r_iic_drv_os.h

Declaration error_t R_IIC_Drv_Reset(r_iic_drv_info_t *pRIic_Info)

Description

- Resets the I²C driver for the corresponding channel.
- Stops IICA by setting IICE to 0 and clears the IICA related registers.*[1]
- If this function is called while communication is in progress, it forcibly stops that communication.
- The following must be set up to use this function.
  The ChNo member of the r_iic_drv_info_t structure; The channel number used

Arguments r_iic_drv_info_t *pRIic_Info ; Pointer to I²C communication information structure

Return Value

R_IIC_NO_INIT
An internal reset was performed and the IICA goes to the uninitialized state. The channel state flag and device state flag are set to R_IIC_NO_INIT.
→ Perform the following operations to restart communication.
(1) Call the initialization function
(2) Call master transmission with pattern 4
(3) Terminate communication by calling the advance function.

R_IIC_LOCK_FUNC
The processing was not performed because another API operation was being performed. The channel state flag and device state flag are not changed.
→ Call the function after processing of the other API finishes.

R_IIC_ERR_PARAM
A parameter error was detected. The arguments were not set up. The channel state flag and device state flag are not changed.
→ Set up the arguments as required by this function.

Remarks

- To restart communication, it is also necessary to call the I²C driver initialization function.
- If the IICA is forcibly stopped during communication, the results of that communication are not guaranteed.
- To avoid effects of other interrupts while setting the channel state flag, this function disables all interrupts other than the non-maskable interrupts.

Note: 1. The items that are reset are the IICA status register (IICA), the STCF and IICBSY bits in the IICA flag register (IICF), and the CLD and DAD bits in IICA control register 1 (IICCTL1).

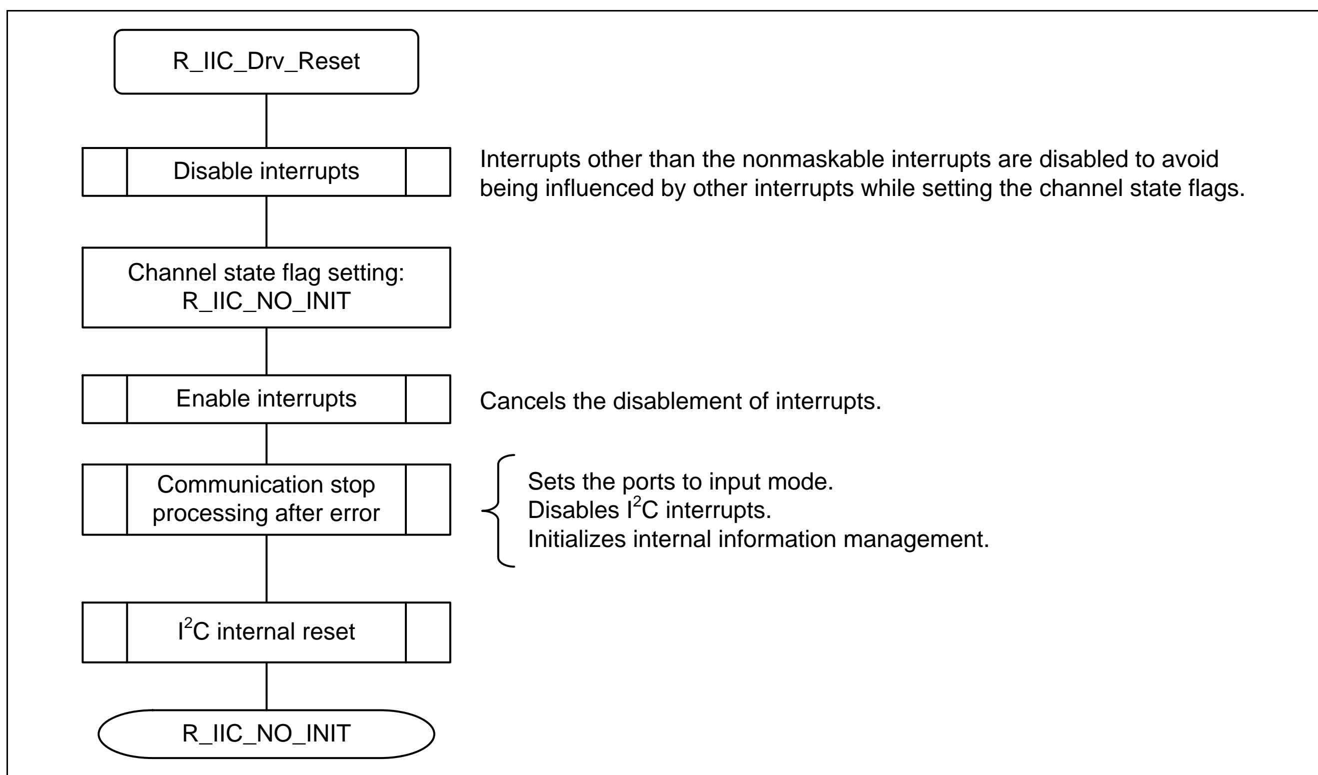


Figure 6-37 I²C Driver Reset Function Overview Flowchart

7. Application Example

7.1 r_iic_drv_api.h

This section presents and examples of settings for actual use.

The section in each file that need to be set are marked with the comment "/**SET**".

(1) **Selecting the IICA Channel Used**

Specify the I^{2}C channel used. The amount of ROM used can be minimized by commenting out the unused channels.

In the example below, channels 0 and 1 are used.

```
/*------------------------------------------------------------------------*/
/*   Select channels to enable.                                           */
/*------------------------------------------------------------------------*/
#define IICA0_ENABLE
#define IICA1_ENABLE
```

(2) **Defining the Maximum Number of Channels Used**

Set this item to the largest channel number used plus one.

In the example below, channels 0 and 1 are used. Since the largest channel number used here is 1, this item is set to 2.

```
/*------------------------------------------------------------------------*/
/*   Define channel No.(max) + 1.                                         */
/*------------------------------------------------------------------------*/
#define MAX_IIC_CH_NUM          (uint8_t)(2)
```

(3) **Defining the Transfer Clock**

Define the setting values for IICA low-level width register x (IICWLx) and IICA high-level width register x (IICWHx) (where x is the channel number). These values must be set for each channel used. See the RL78 Family microcontroller User's Manual: Hardware for details on this setting.

Transfer rates up to a maximum of 400 kHz can be set. However, if standard mode devices and fast mode devices are used together, the standard mode maximum rate of 100 kHz must be set. Also note that the SDA and SCL signal rise time (tR) and fall time (tF) will differ depending on the pull-up resistors used and the line capacitances, and that may require revising these values.

The example shown below applies for the following conditions.

- System clock: 32 kHz
- Transfer clock: 400 kHz
- SDA and SCA signal rise time (tR) and fall time (tF): 300 ns

```
/*------------------------------------------------------------------------*/
/*   Define frequency as iic channel. (Please add a channel as needed.)   */
/*------------------------------------------------------------------------*/
/* Freq = 400KHz at main system clock = 32MHz */
#define R_IIC_CH0_LCLK  (uint8_t)(20) /* Channel 0 IICWL0 register setting  */
#define R_IIC_CH0_HCLK  (uint8_t)(18) /* Channel 0 IICWH0 register setting  */
#define R_IIC_CH1_LCLK  (uint8_t)(20) /* Channel 1 IICWL1 register setting  */
#define R_IIC_CH1_HCLK  (uint8_t)(18) /* Channel 1 IICWL1 register setting  */
```

(4) **Counter Definitions**

These are the counter values for various software loops. As such, the loop times will change with the system clock used. These setting values should be reviewed as necessary.

```
/*---------------------------------------------------------------------------*/
/*   Define counter.                                                         */
/*---------------------------------------------------------------------------*/
#define REPLY_CNT       (uint32_t)(100000) /* Counter of non-reply errors   */
#define START_COND_WAIT (uint16_t)(100)
                         /* Counter of waiting start condition generation   */
#define STOP_COND_WAIT  (uint16_t)(100)
                         /* Counter of waiting stop condition generation    */
#define BUSCHK_CNT      (uint16_t)(100) /* Counter of checking bus busy     */
#define SDACHK_CNT      (uint16_t)(100) /* Counter of checking SDA level    */
#define SCLCHK_CNT      (uint16_t)(100) /* Counter of checking SCL level    */
#define SCL_L_WAIT      (uint16_t)(100)
                         /* Counter of waiting SCL Low clock setting        */
#define SCL_H_WAIT        (uint16_t)(100)
                         /* Counter of waiting SCL High clock setting       */
```

7.2 r_iic_drv_sfr.h

A file with a filename of the form r_iic_drv_sfr.hXXX has been created for each microcontroller. One of these must be renamed to r_iic_drv_sfr.h and used. If there is no such file for the microcontroller used, the user must refer to these files and create an appropriate r_iic_drv_sfr.h file.

This section presents and examples of settings for actual use.

The section in each file that need to be set are marked with the comment "/**SET**/".

(1) Defining the Pins Used

The user must define the port numbers for the pins used. These settings are required for each channel used.

The example below applies when channel 0 is used.

```
/*------------------------------------------------------------------------------*/
/*   Define channel register.                                                   */
/*------------------------------------------------------------------------------*/
#ifdef IICA0_ENABLE

/* Define port */
#define R_IIC_PM_SCL0    PM6.0    /* SCL0 Port mode registers                  */
#define R_IIC_PM_SDA0    PM6.1    /* SDA0 Port mode registers                  */
#define R_IIC_P_SCL0     P6.0     /* SCL0 Port registers                       */
#define R_IIC_P_SDA0     P6.1     /* SDA0 Port registers                       */
```

(2) Defining the Input Clock Supply Control Bit (IICAxEN)

The user must define the clock supply control bit (IICAxEN) for the IICA serial interface.

The example below applies when channel 0 is used.

```
/* Define peripheral enable register */
#define R_IIC_IICA0EN    IICA0EN  /* IICA0 peripheral enable register          */
```

(3) Defining IICA Control Register x1 (IICCTLx1)

Change this setting according to the maximum transfer rate used. The set value differs depending on whether the maximum transfer rate is 100 or 400 kHz.

The example below applies when the maximum transfer rate used will be 400 kHz.

```
/*------------------------------------------------------------------------------*/
/*   Define register setting.                                                   */
/*------------------------------------------------------------------------------*/
#define R_IIC_IICCTL1_INIT (uint8_t)(0x0D)  /* SMC=1(400KHz),DFC=1,PRS=1    */
```

The example below applies when the maximum transfer rate used will be 100 kHz.

```
/*------------------------------------------------------------------------------*/
/*   Define register setting.                                                   */
/*------------------------------------------------------------------------------*/
#define R_IIC_IICCTL1_INIT (uint8_t)(0x01)  /* SMC=0(100KHz),DFC=0,PRS=1    */
```

(4) **Defining the IICA Interrupt Priorities**

The user must define the interrupt priorities for the channels used.

The example below applies for the use of channel 0 with interrupt priority level 2.

```
* Interrupt register setting */
#ifdef IICA0_ENABLE                                 /* Channel 0                */
#define R_IIC_CH0_PR0_INIT (uint8_t)(0x00)    /* Priority level 2         */
#define R_IIC_CH0_PR1_INIT (uint8_t)(0x01)    /* Priority level 2         */
```

7.3 r_iic_drv_int.c

This section presents examples of settings for actual use.

The location of the settings in each file is indicated by the comment line “/**SET**/”.

7.3.1 Integrated Development Environment CS+ for CA,CX (formerly CubeSuite+)

(1) **Defining the Interrupt Function #pragma interrupt**

Define the interrupt function #pragma interrupt for the I²C channels to be used. Comment out the unused channels. Note that a compile error will result if a definition corresponds to a channel that is not implemented on the microcontroller.

The example below applies when channel 0 is used and channel 1 is unused.

```
#pragma interrupt INTIICA0 r_iic_drv_intiica0_isr
                                        /* Uses RL78 channel 0 interrupt IICA. */
/* #pragma interrupt INTIICA1 r_iic_drv_intiica1_isr */
                                        /* Uses RL78 channel 1 interrupt IICA. */
```

7.3.2 Integrated Development Environment CS+ for CC

(1) **Defining the Interrupt Function #pragma interrupt**

Define the interrupt function #pragma interrupt for the I²C channels to be used. Comment out the unused channels. Note that a compile error will result if a definition corresponds to a channel that is not implemented on the microcontroller.

The example below applies when channel 0 is used and channel 1 is unused.

```
#pragma interrupt r_iic_drv_intiica0_isr(vect=INTIICA0)
                        /* Uses RL78 channel 0 interrupt IICA.  */ /** SET **/
/* #pragma interrupt r_iic_drv_intiica1_isr(vect=INTIICA1) */
                        /* Uses RL78 channel 1 interrupt IICA.  */ /** SET **/
```

7.3.3 Integrated Development Environment IAR Embedded Workbench

(1) **Define Setting for SFR Area**

When using the IAR integrated development environment with the RL78 microcontroller, a header file in which SFR is defined for the microcontroller is required.

Also refer to the version of the clock-synchronous signal master control software for the specific microcontroller.

This setting is used for the SPI slave device select control signal.

Table 7-1 Define Setting for Microcontroller and SFR Area

Integrated development environment	MCU	SFR setting required?	Setting Method
CubeSuite+ CS+	RL78	Not required	Not required
IAR Embedded Workbench	RL78	Required	#ifdef __ICCRL78__ #include <ior5f104pj.h> ← Modify to match microcontroller. #include <ior5f104pj_ext.h> ← Modify to match microcontroller. #endif

The example below applies to the 100-pin version of the RL78/G14.

```
#ifdef __ICCRL78__                          /* IAR RL78 Compiler                    */
   #include <ior5f104pj.h>                  /* for RL78/G14 100pin (R5F104PJ)       */
   #include <ior5f104pj_ext.h>              /* for RL78/G14 100pin (R5F104PJ)       */
#endif   /* __ICCRL78__ */
```

(2) **Defining the Interrupt Function #pragma interrupt**

Define the interrupt function #pragma interrupt for the I²C channels to be used. Comment out the unused channels. Note that a compile error will result if a definition corresponds to a channel that is not implemented on the microcontroller.

The example below applies when channel 0 is used and channel 1 is unused.

```
/* #pragma vector=INTIICA1_vect */                                  /** SET **/
__interrupt __root void r_iic_drv_intiica1_isr(void)
```

7.4 r_iic_drv_sfr.c

This section presents examples of settings for actual use.

The location of the settings in each file is indicated by the comment line “/**SET**/”.

(1) **Define Setting for SFR Area**

When using the IAR integrated development environment with the RL78 microcontroller, a header file in which SFR is defined for the microcontroller is required.

Also refer to the version of the clock-synchronous signal master control software for the specific microcontroller.

This setting is used for the SPI slave device select control signal.

Table 7-2 Define Setting for Microcontroller and SFR Area

Integrated development environment	MCU	SFR setting required?	Setting Method
CubeSuite+ CS+	RL78	Not required	Not required
IAR Embedded Workbench	RL78	Required	#ifdef __ICCRL78__ #include <ior5f104pj.h> ← Modify to match microcontroller. #include <ior5f104pj_ext.h> ← Modify to match microcontroller. #endif

The example below applies to the 100-pin version of the RL78/G14.

```
#ifdef __ICCRL78__                          /* IAR RL78 Compiler                    */
   #include <ior5f104pj.h>                  /* for RL78/G14 100pin (R5F104PJ)       */
   #include <ior5f104pj_ext.h>              /* for RL78/G14 100pin (R5F104PJ)       */
#endif   /* __ICCRL78__ */
```

7.5 r_iic_drv_os.c

Do not include because of not evaluating.

7.6 Recovery Processing Example

The recovery processing to restore communication is explained when SDA or SCL are in the low-hold state.

Execute the following operation. Figure 7-1 illustrates the recovery processing using SCL pseudo clock generation.

The device transitions to the idle state when processing finishes. It is possible to start communication by calling start function.

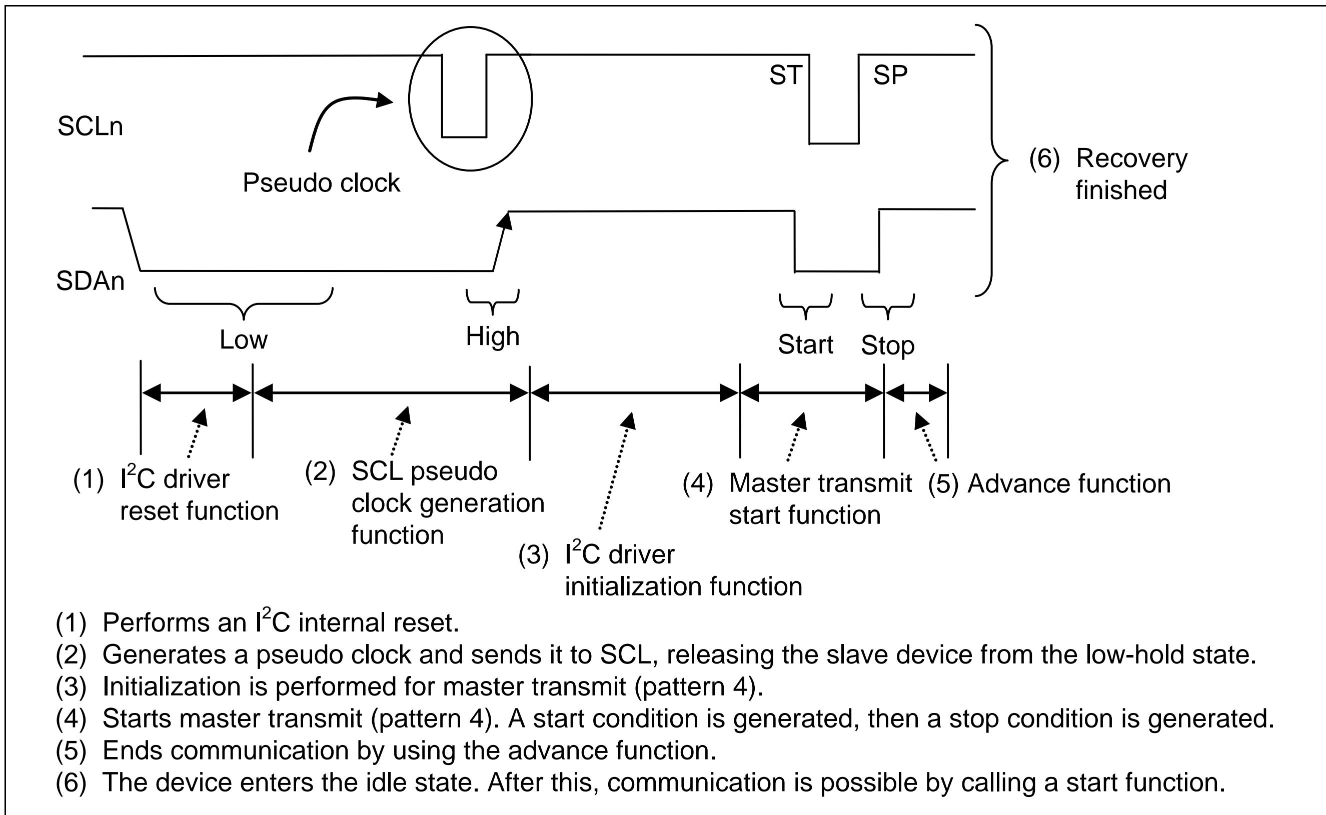


Figure 7-1 Recovery Process Using SCL Pseudo Clock Generation

8. Usage Notes

8.1 Notes on Embedding

Include the following header files when embedding this sample code in an application.

- 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.

8.2 Notes on Initialization

When initializing a target channel for the first time, set the channel state flag g_iic_ChStatus[] and the device state flag *(pRIic_Info.pDevStatus) to R_IIC_NO_INIT. After that, since these flags are managed by this sample code, they may not be set by user code.

8.3 Notes on the Channel State Flag and Device State Flag

This sample code maintains the consistency of the communication state using the channel state flag and device state flag. Communication operation is not guaranteed if these flags are modified after first initialization.

8.4 Control Methods for Multiple Slave Devices on the Same Channel

Use the following procedure to control multiple slave devices on the same channel.

The processing in the item (1) below can prevent communication from being performed with devices in the not communicating state.

(1) Verify that the device state flag in the I^2C communication information structure for the slave device that is the object of the advance function call is “R_IIC_COMMUNICATION”.
(2) Call the advance function.
(3) Repeat steps (1) and (2) until communication completes.
(4) Communication has completed. After this, communication is possible by calling a start function.

8.5 Performing Advance Function Processing from Within an Interrupt Under OS Control

We have not looked into performing advance function processing from within an interrupt when and OS*[1] is used. To use this approach, thorough evaluation and testing will be required.

Note: 1. This sample code assumes μITRON 4.0 will be the OS.

8.6 Transfer Rate Setting

The transfer rate must be set for each channel. Transfer rates up to a maximum of 400 kHz can be set.

Note, however, that if standard mode devices and fast mode devices are used together, the standard mode maximum rate of 100 kHz must be set. Set the transfer rate using R_IIC_CHx_LCLK and R_IIC_CHx_LCLK (where x is the channel number) defined in Table 6-13.

8.7 Notes On Setting The #define Definitions of IICAx_ENABLE and MAX_IIC_CH_NUM

This section described the settings for the case where only channel 2 will be used.

Enable only the definition of IICA2_ENABLE for the IICAx_ENABLE #define definitions. This masks out the source code for channel 0 and channel 1.

```
/*--------------------------------------------------------------------------*/
/*   Select channels to enable.                                             */
/*--------------------------------------------------------------------------*/
/* #define IICA0_ENABLE */
/* #define IICA1_ENABLE */
#define IICA2_ENABLE
```

Set the #define definition of MAX_IIC_CH_NUM to 3. Note that although the number of channels used is 1, the value set here must be the largest channel number used plus one.

```
/*--------------------------------------------------------------------------*/
/*   Define channel No.(max) + 1.                                           */
/*--------------------------------------------------------------------------*/
#define MAX_IIC_CH_NUM        (uint8_t)(3)
```

8.8 Defining the Interrupt Function #pragma interrupt

Define the interrupt function #pragma interrupt for the I^{2}C channels to be used. Comment out the unused channels. Note that a compile error will result if a definition corresponds to a channel that is not implemented on the microcontroller.

8.9 Notes on User API Calls

Only call the user APIs described in this application note from the main processing routine. A malfunction may result if they are called from an interrupt handler.

8.10 About Warnings of Duplicate of Type Declaration

This driver has declared the intN_t and uintN_t that are declared in the "stdint.h". There is a possibility that the warning occurs when including the "stdint.h". If the type of declaration is unnecessary, delete the declaration of this driver.

8.11 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/

All trademarks and registered trademarks are the property of their respective owners.

Revision History

Rev.	Date	Description	
		Page	Summary
1.03	Oct. 30, 2014	—	First edition issued
1.04	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)
		33	Changed the following title to section 6.10 (1) RL78/G14 IICA Integrated Development Environment CS+ for CA,CX (Compiler: CA78K0R) Added the following title to section 6.10 (2) RL78/G14 IICA Integrated Development Environment CS+ for CC (Compiler: CC-RL)
		36	Section 6.11 File Structure Changed Application Note Number. Changed Folder names.
		38, 79	Added #define SCLCHK_CNT.
		82-83	Changed section 7.3 r_iic_drv_int.c. Added section 7.3.1, 7.3.2, and 7.3.3.
		86	Added section 8.10 and 8.11.

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.

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

© 2016 Renesas Electronics Corporation. All rights reserved.

Colophon 5.0