# RL78/G23

## I²S Communication with ELCL and SPI

### Introduction

This application note describes how to use the logic and event link controller (ELCL) and Serial Array Unit (SAU) to achieve I²S communication.

### Target Device

RL78/G23

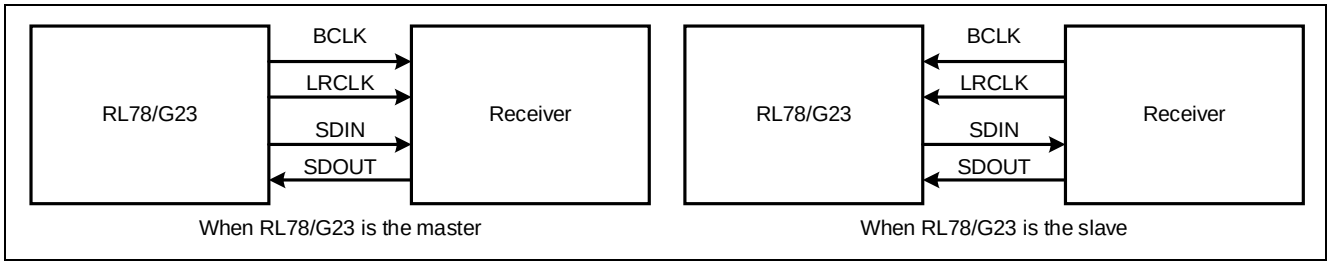
When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

## Contents

# 1. Specifications

Figure 1-1 shows the I²S communication configuration supported by this application note and Table 1-1 shows the specifications.

**Figure 1-1 RL78/G23 and receiver configuration**




Note LRCLK: LR clock (word select)
A signal that distinguishes between L and R channels. Low level is defined as L channel and High level is defined as R channel.
Matches the sampling frequency.
BCLK: I²S bit clock
SDIN: The signal that the receiver receives the sound data.
SDOUT: The signal that the receiver sends sound data.

**Table 1-1 I²S Communication Format**

| Items | Contents |
|---|---|
| sound data format | PCM |
| Function | Can operate on both master and slave. |
| | BCLK frequency: 32fs, 48fs, 64fs (fs: Sampling frequency) |
| | Sampling frequency: 8-96kHz Note 1 |
| | Data size: 16, 24, 32bits |

Note 1. It can operate with the combinations of data size and sampling frequency shown in the table. However, at a sampling frequency of 96 kHz, a data size of 24 or 32 bits cannot be supported because the electrical characteristics cannot be satisfied.

Figure 1-2 shows the configuration of I²S communication (master) using ELCL.

BCLK and LRCLK are generated using Timer Array Unit 0 (TAU0). The BCLK generated in the interval timer mode of TAU00 is connected to SCK01 of SAU01 via ELCL for adjusting the timing. TAU00 is used as an external event in the event counter mode of TAU01 and is output from the pin as LRCLK.

**Figure 1-2 System Configuration of I²S communication (master) using ELCL**

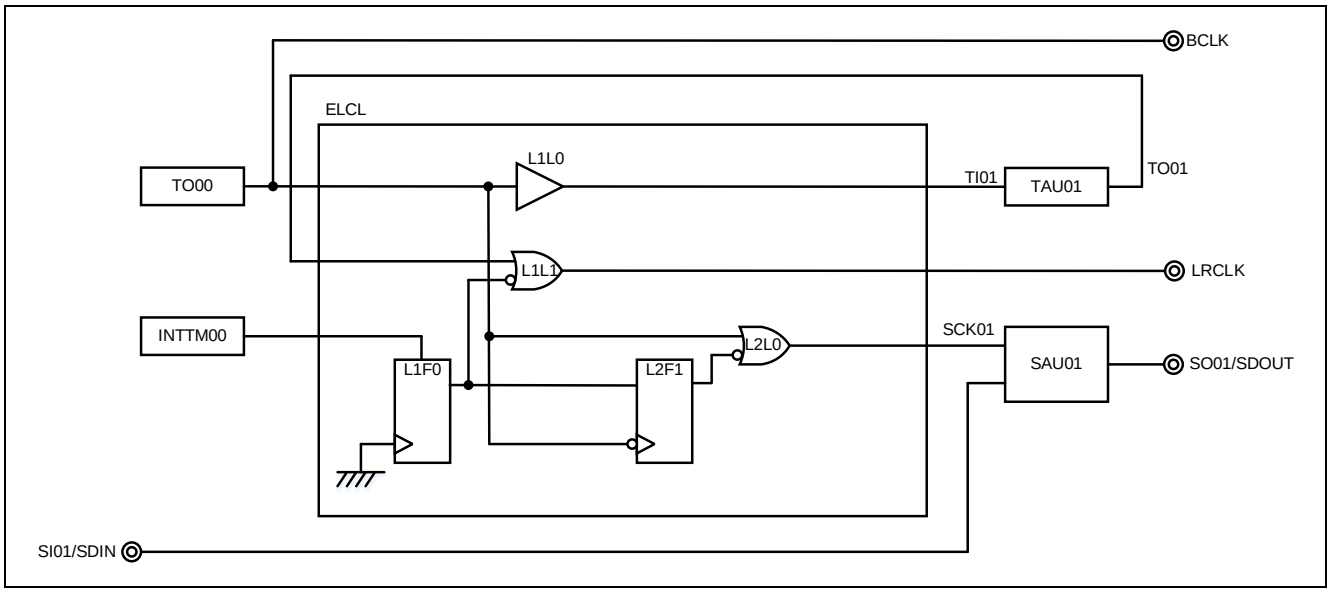



Figure 1-3 shows the timing chart (data size 16bits) of I²S communication (master) using ELCL.

(1) TAU00 is set to interval timer mode, TAU01 to event counter mode, and SAU01 to continuous transmission mode (slave).
(2) TAU00 and TAU01 start operation.
(3) Latch INTTM00 with L1F0 of ELCL and set LRCLK Low.
(4) The signal latched in (3) delays one clock cycle of BCLK and generates SCK01.
(5) SO01 starts operating in sync with SCK01.

**Figure 1-3 Timing chart of I²S communication (master) using ELCL**

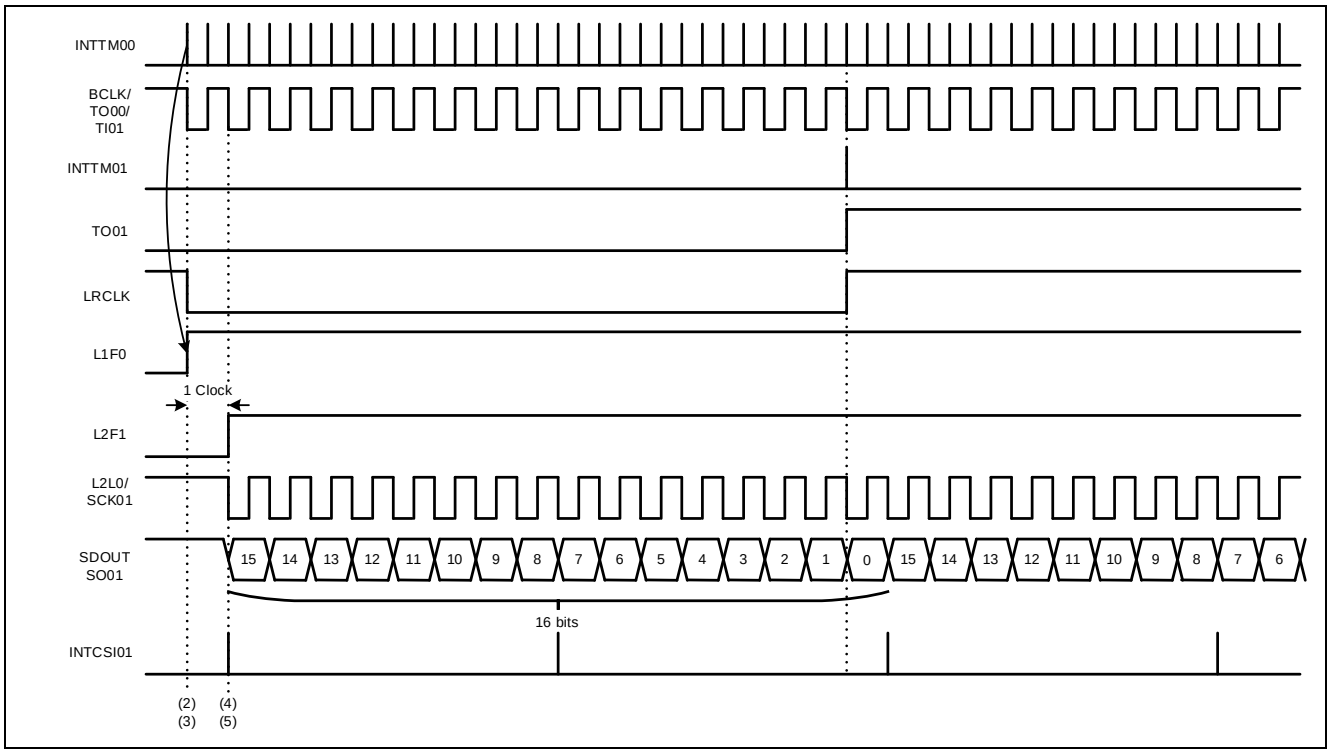


Figure 1-4 shows the configuration of I²S communication (slave) using ELCL.

BCLK is connected to SCK01 of SAU01 by adjusting the timing with ELCL. LRCLK is used by ELCL as a signal to generate SCK01 timing.

**Figure 1-4 System Configuration of I²S communication (slave) using ELCL**

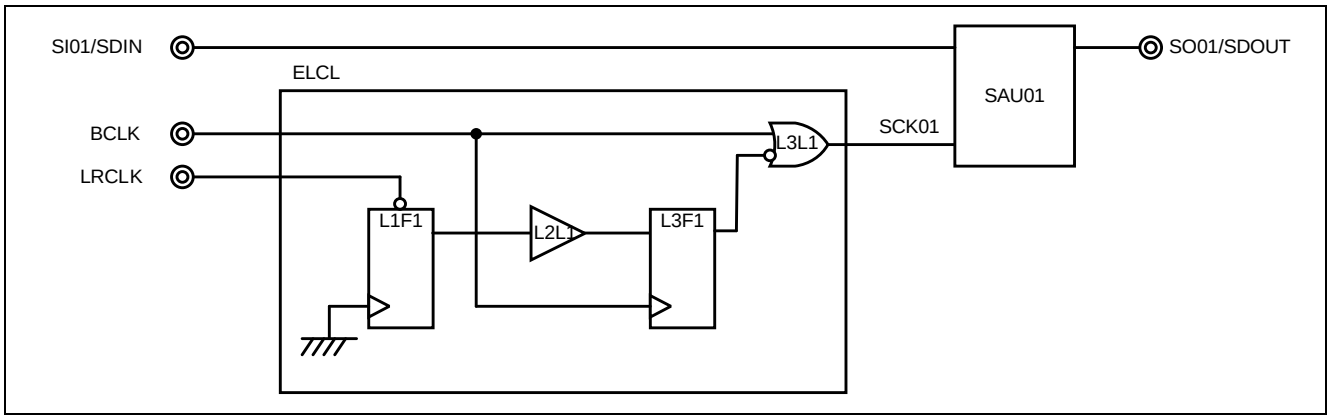
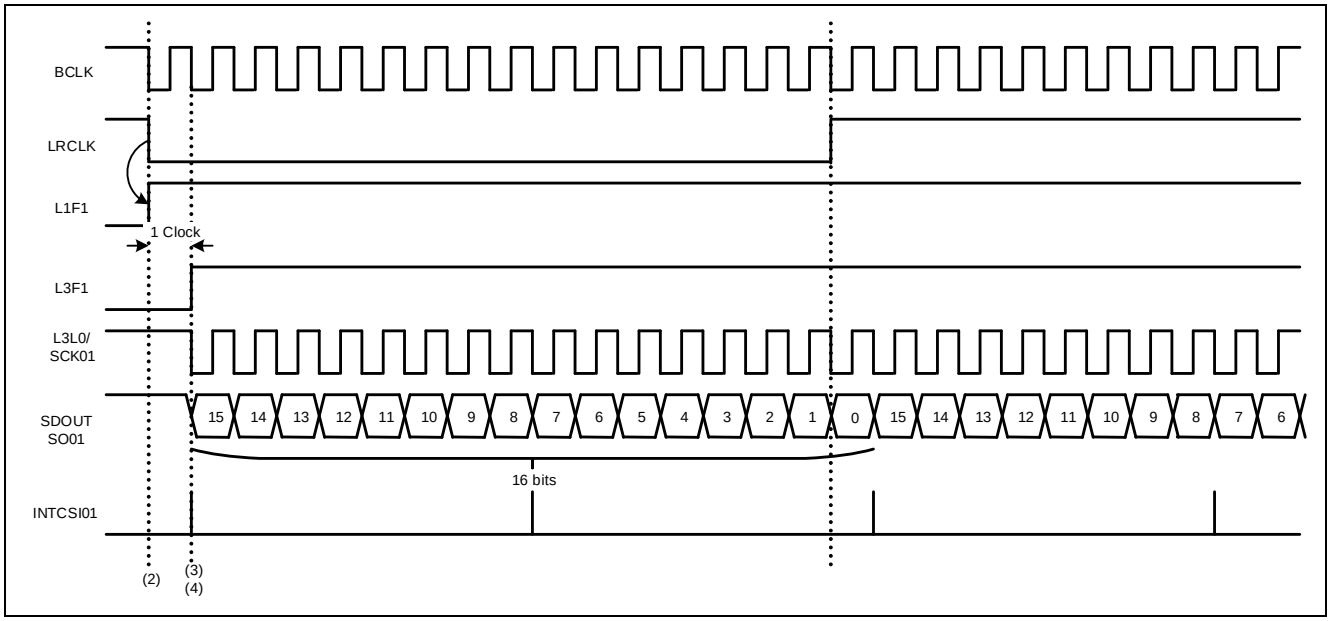



Figure 1-5 shows a timing chart (data size 16 bits) of I²S communication (slave) using ELCL.

(1) SAU01 is set to continuous transmit mode (slave).
(2) Latch the high to low level change of LRCLK with L1F1 of ELCL.
(3) Delay BCLK by 1 clock cycle with the signal latched in (2) to generate SCK01.
(4) SO01 starts operating in synchronization with SCK01.

**Figure 1-5 Timing chart of I²S communication (slave) using ELCL**




Note. The initial values of BCLK and LRCLK depend on the CODEC module.

## 2. Conditions for Operation Confirmation Test

The sample code with this application note runs properly under the condition below.

**Table 2-1 Operation Confirmation Conditions**

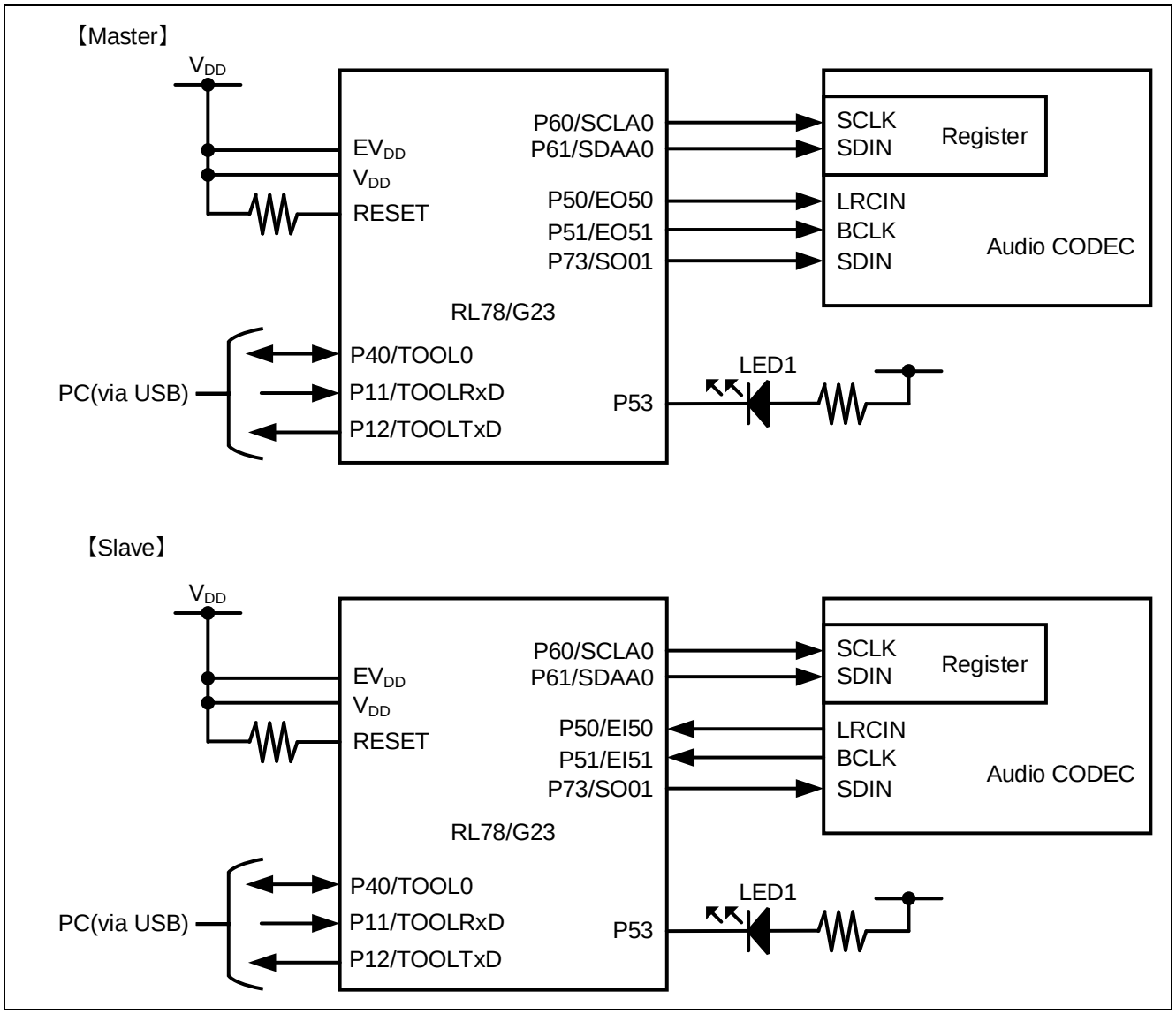
| Items | Contents |
|---|---|
| MCU | RL78/G23 (R7F100GLG) |
| Operating frequencies | • High-speed on-chip oscillator clock: 32MHz<br>• CPU/peripheral hardware clock: 32MHz |
| Operating voltage | • 3.3V<br>• LVD0 operations ($V_{LVD0}$): Reset mode<br>Rising edge TYP. 1.90V<br>Falling edge TYP. 1.86V |
| Integrated development environment (CS+) | CS+ for CC V8.07.00 from Renesas Electronics Corp. |
| C compiler (CS+) | CC-RL V1.11 from Renesas Electronics Corp. |
| Integrated development environment (e² studio) | e² studio 2022-04 (22.04.0) from Renesas Electronics Corp. |
| C compiler (e² studio) | CC-RL V1.11 from Renesas Electronics Corp. |
| Integrated development environment (IAR) | IAR Embedded Workbench for Renesas RL78 v4.21.1 from IAR Systems |
| C compiler (IAR) | |
| Smart Configurator | V.1.3.0 |
| Board support package (r_bsp) | V.1.20 |
| Emulator | CS+、e² studio: COM port<br>IAR: E2 Emulator Lite |
| Board | RL78/G23 Fast Prototyping Board (RTK7RLG230CLG000BJ) |

# 3. Hardware

## 3.1 Example of Hardware Configuration

Figure 3-1 shows an example of the hardware configuration in this application.

**Figure 3-1 Hardware Configuration**




Note. For I²C communication, refer to 4.1.1 Overview of the sample program.

Caution 1. This simplified circuit diagram was created to show an overview of connections only. When actually designing your circuit, make sure the design includes sufficient pin processing and meets electrical characteristic requirements. (Connect each input-only port to $V_{DD}$ or $V_{SS}$ through a resistor.)

Caution 2. Connect the $EV_{SS}$ pin to $V_{SS}$ and the $EV_{DD}$ pin to $V_{DD}$.

Caution 3. $V_{DD}$ must be held at not lower than the reset release voltage ($V_{LVD0}$) that is specified as LVD.

## 3.2 Used Pins

Table 3-1 shows list of used pins and assigned functions.

**Table 3-1 List of Pins and Functions**

| Pin name | Input/Output | Function |
|---|---|---|
| P53 | Output | LED1 lights (Low Active) |
| P60/SCLA0 | Output | Serial clock |
| P61/SDAA0 | Output | Serial data |
| P50/EO50 | Output (master) | LRCLK |
| P50/EI50 | Input (slave) | |
| P51/EO51 | Output (master) | BCLK |
| P51/EI51 | Input (slave) | |
| P73/SO01 | Output | SDIN |

Caution. In this application note, only the used pins are processed. When actually designing your circuit, make sure the design includes sufficient pin processing and meets electrical characteristic requirements.

# 4. I²S communication (master) using ELCL

## 4.1 Software

### 4.1.1 Overview of the sample program

This sample code uses RS Audio CODEC 754-1974. It also uses I²C communication for register settings in the audio CODEC and I²S communication for sound data transmission processing.

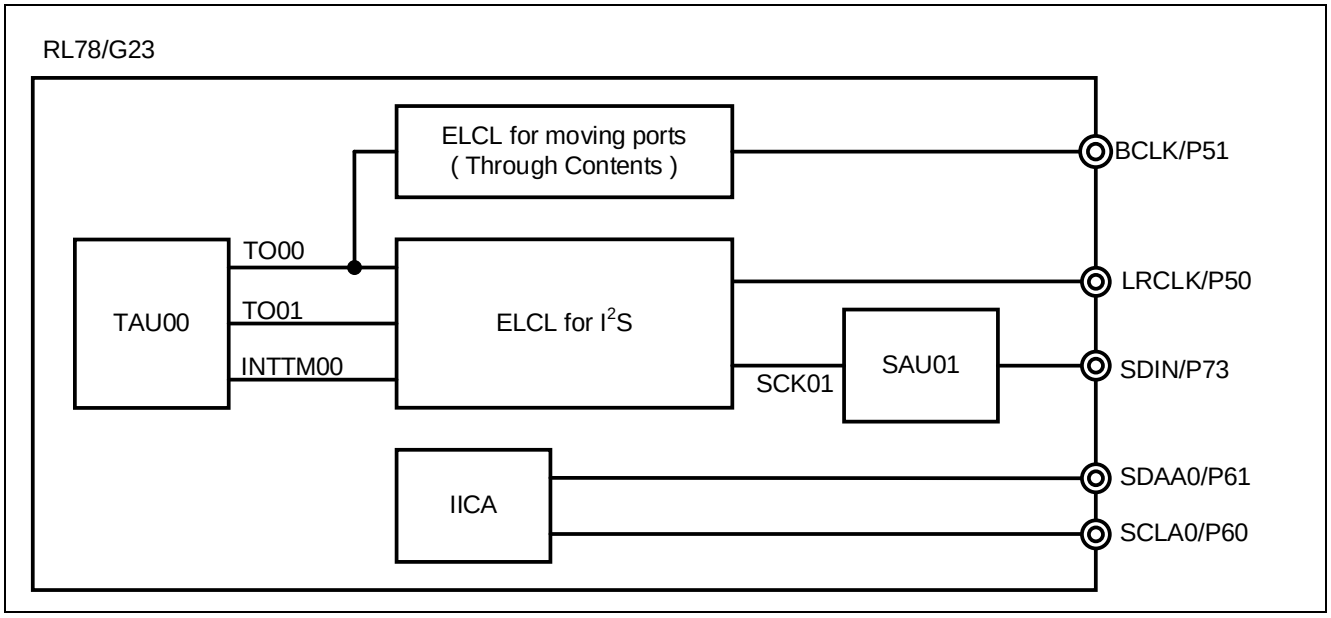
BCLK uses the Through content of ELCL and the output pin is changed to P51.

The sample code works as follows.

(1) IICA for I²C communication starts operation and sets registers in the audio CODEC.
(2) CSI01 for I²S communication starts operation (waiting for SCK01).
(3) TAU0 for BCLK and LRCLK starts operation and data transmission starts.
(4) After data transmission, turns on LED1.
(5) Stop CSI01 and TAU0.

Figure 4-1 shows the system configuration of the sample code.

**Figure 4-1 System configuration of the sample code**



### 4.1.2 Folder Configuration

Table 4-1, Table 4-2 shows folder configuration of source file and header files using by sample code except the files generated by integrated development environment and the files in the bsp environment.

**Table 4-1 Folder configuration (1/2)**

| Folder/File configuration | Outline | Created by Smart configurator |
|---|---|---|
| ¥r01an6420_elcl_i2s_master<DIR>Note 3 | Root folder of this sample code | |
| ¥src<DIR> | Folder for program source | |
| main.c | Sample code source file | |
| main.h | Sample code header file | |
| r_codec.cNote 4 | Source file for CODEC module settings | |
| r_codec.hNote 4 | Header file for CODEC module settings | |
| sound_data.c | Source file for sound data | |
| sound_data.h | Header file for sound data | |
| ¥csi01<DIR> | Folder for CSI01 program | |
| csi01.c | Source file for CSI01 | |
| csi01.h | Header file for CSI01 | |
| ¥elcl<DIR> | Folder for ELCL program | |
| elcl.c | Source file for ELCL | |
| elcl.h | Header file for ELCL | |
| ¥iica0<DIR>Note 4 | Folder for IICA0 program | |
| iica0.c | Source file for IICA0 | |
| iica0.h | Header file for IICA0 | |
| ¥smc_gen<DIR> | Folder created by Smart Configurator | √ |
| ¥Config_PORT<DIR> | Folder for PORT program | √ |
| Config_PORT.c | Source file for PORT | √ |
| Config_PORT.h | Header file for PORT | √ |
| Config_PORT_user.c | Interrupt source file for PORT | √Note 1 |
| ¥Config_TAU0_0<DIR> | Folder for TAU00 program | √ |
| Config_TAU0_0.c | Source file for TAU00 | √ |
| Config_TAU0_0.h | Header file for TAU00 | √ |
| Config_TAU0_0_user.c | Interrupt source file for TAU00 | √Note 2 |
| ¥Config_TAU0_1<DIR> | Folder for TAU01 program | √ |
| Config_TAU0_1.c | Source file for TAU01 | √ |
| Config_TAU0_1.h | Header file for TAU01 | √ |
| Config_TAU0_1_user.c | Interrupt source file for TAU00 | √Note 1 |

Note. <DIR> means directory

Note 1. Not used in this sample code.

Note 2. Added the interrupt handling routine to the file generated by the Smart Configurator.

Note 3. The IAR version of the sample code contains r01an6420_elcl_i2s_master.ipcf. For the ipcf file, refer to "RL78 Smart Configurator User Guide: IAR (R20AN0581)".

Note 4. This file is required to configure the RS Audio CODEC 754-1974 module.

**Table 4-2 Folder configuration (2/2)**

| Folder/File configuration | Outline | Created by Smart configurator |
|---|---|---|
| ¥Config_TAU0_7<DIR>Note 4 | Folder for TAU07 program | √ |
| Config_TAU0_7.c | Source file for TAU07 | √ |
| Config_TAU0_7.h | Header file for TAU07 | √ |
| Config_TAU0_7_user.c | Interrupt source file for TAU07 | √Note 5 |
| ¥Config_Through<DIR> | Folder for Through program | √ |
| Config_Through.c | Source file for Through | √ |
| Config_Through.h | Header file for Through | √ |
| Config_Through _user.c | Interrupt source file for Through | √ |
| ¥general<DIR> | Folder for initialize or common program | √ |
| ¥r_bsp<DIR> | Folder for BSP program | √ |
| ¥r_config<DIR> | Folder for program | √ |

Note. <DIR> means directory

Note 4. This file is required to configure the RS Audio CODEC 754-1974 module.

Note 5. Added the interrupt handling routine to the file generated by the Smart Configurator.

Since I²S communication uses CSI, it does not follow the original CSI standard and overrun errors occur. Error detection processing is removed from the code generated by the Smart configurator.

In addition, to enable multiple interrupts in IICA, the code generated by the Smart configurator is modified to remove unnecessary functions.

### 4.1.3 Option Byte Settings

Table 4-3 shows the option byte settings.

**Table 4-3 Option Byte Settings**

| Address | Setting Value | Contents |
|---|---|---|
| 000C0H/040C0H | 1110 1111B (EFH) | Operation of Watchdog timer is stopped (counting is stopped after reset) |
| 000C1H/040C1H | 1111 1110B (FEH) | LVD0 operating mode: reset mode<br>Detection voltage: Rising edge 1.90V<br>Falling edge 1.86V |
| 000C2H/040C2H | 1110 1000B (E8H) | Flash operating mode: HS mode<br>High-speed on-chip oscillator clock: 32MHz |
| 000C3H/040C3H | 1000 0101B (85H) | On-chip debugging is enabled |

### 4.1.4 Constants

Table 4-4, Table 4-5 shows the constants that are used in this sample code.

**Table 4-4 Constants used in the sample code (1/2)**

| Constant Name | Setting Value | Contents | File |
|---|---|---|---|
| LED1 | P5_bit.no3 | P53 | csi01.c |
| LED_ON | 0 | Setting value for turning on the LED | csi01.c |
| LED_OFF | 1 | Setting value for turning off the LED | csi01.c |
| DATA_48K_32B | 1 | Select the sound data to play. (1: Enable, 0: Disable) | sound_data.h |
| DATA_44K_24B | 0 | Select the sound data to play. (1: Enable, 0: Disable) | sound_data.h |
| DATA_8K_16B | 0 | Select the sound data to play. (1: Enable, 0: Disable) | sound_data.h |
| DECODE_PCM_SIZE | 32000 (DATA_48K_32B) | The number of sound data. The setting value changes depending on the sound data to be played. | sound_data.h |
| | 22791 (DATA_44K_24B) | | |
| | 20481 (DATA_8K_16B) | | |
| g_tx_buf[] | See sample code | Sound data | sound_data.c |
| SENSOR_ADD | 0x34 | Sensor address | r_codec.h |
| WAIT_TIME | 100 | IICA0 communication wait time | r_codec.h |
| g_cmd_l_vol[2] | { 0x00, 0x17 } | Left line input channel volume control command | r_codec.c |
| g_cmd_r_vol[2] | { 0x02, 0x17 } | Right line input channel volume control command | r_codec.c |
| g_cmd_bypass[2] | { 0x08, 0x12 } | Analog audio path control command | r_codec.c |
| g_cmd_deempha[2] | { 0x0A, 0x00 } | Digital audio path control command | r_codec.c |
| g_cmd_line_adc[2] | { 0x0C, 0x42 } | Power down control command | r_codec.c |

**Table 4-5 Constants used in the sample code (2/2)**

| Constant Name | Setting Value | Contents | File |
|---|---|---|---|
| g_cmd_set_rate[2] | {0x10, 0x00} (48kHz) | Sampling frequency control command | r_codec.c |
| | { 0x10, 0x20 } (44kHz) | | |
| | {0x10, 0x0C} (8kHz) | | |
| g_cmd_set_format[2] | {0x0E, 0x0E} (32bits) | Digital audio interface format command | r_codec.c |
| | {0x10, 0x0A} (24bits) | | |
| | {0x0E, 0x02} (16bits) | | |
| g_cmd_activate[2] | { 0x12, 0x01 } | CODEC module active command | r_codec.c |
| g_cmd_inactivate[2] | { 0x12, 0x00 } | CODEC module inactive command | r_codec.c |
| g_cmd_reset[2] | { 0x1E, 0x00 } | Register reset command | r_codec.c |

### 4.1.5 Variables

Table 4-6 shows the global variables used in this sample code.

**Table 4-6 Global variables used in the sample code**

| Type | Variable name | contents | Functions used in |
|---|---|---|---|
| volatile uint16_t | g_ms_timer | Count value of the wait process | r_ms_delay, r_Config_TAU0_7_interrupt |
| volatile uint8_t | g_tx_done_flag | Transmission Completion Flag | main r_csi01_callback_sendend |
| volatile uint8_t | g_sample_mode | I²C communication status | r_codec_init r_codec_start r_codec_stop r_iica0_callback_master_sende nd |

### 4.1.6 Functions

Table 4-7 shows the functions used in the sample code. However, the unchanged functions generated by the Smart Configurator are excluded.

**Table 4-7 Functions**

| Function name | Outline | Source file |
|---|---|---|
| main | Main process | main.c |
| r_csi01_create | CSI01 initialize process | csi.c |
| r_csi01_start | CSI01 operation start process | csi.c |
| r_csi01_stop | CSI01 operation stop process | csi.c |
| r_csi01_send | CSI01 start sending process | csi.c |
| r_csi01_callback_sendend | CSI01 callback function for end of transmission process | csi.c |
| r_csi01_interrupt | CSI01 interrupt process | csi.c |
| r_elcl_create | ELCL initialize process | elcl.c |
| r_elcl_start | ELCL operation start process | elcl.c |
| r_elcl_stop | ELCL operation stop process | elcl.c |
| r_elcl_reset_flipflop | ELCL flip-flop reset process | elcl.c |
| r_elcl_start_port_move | ELCL terminal output destination change start process | elcl.c |
| r_elcl_stop_port_move | ELCL terminal output destination change stop process | elcl.c |
| r_iica0_create | IICA0 initialize process | Iica0.c |
| r_iica0_stop | IICA0 operation stop process | Iica0.c |
| r_iica0_stopcondition | IICA0 stop condition process | Iica0.c |
| r_iica0_master_send | IICA0 master send start process | Iica0.c |
| r_iica0_callback_master_sendend | IICA0 callback function for end of transmission process | Iica0.c |
| r_iica0_master_handler | IICA0 interrupt handler | Iica0.c |
| r_iica0_interrupt | IICA0 interrupt process | Iica0.c |
| r_codec_init | CODEC module setup process | r_codec.c |
| r_codec_start | CODEC module start process | r_codec.c |
| r_codec_stop | CODEC module stop process | r_codec.c |
| r_tau0_start | TAU0 start process | r_cg_tau_common.c |
| R_Config_TAU0_0_Create_UserInit | TAU0 initialization user code addition process | Config_TAU0_0_user.c |
| r_ms_delay | Wait process | Config_TAU0_7_user.c |
| r_Config_TAU0_7_interrupt | TAU0 channel 7 interrupt process | Config_TAU0_7_user.c |

### 4.1.7 Function Specifications

This part describes function specifications of the sample code.

[Function name] main

| | |
|---|---|
| **Outline** | Main process |
| **Header** | r_smc_entry.h, main.h, elcl.h, csi01.h, r_codec.h, sound_data.h |
| **Declaration** | void main (void); |
| **Description** | This function initializes ELCL and CSI01, sets CODEC module, and performs I²S communication. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_create

| | |
|---|---|
| **Outline** | CSI01 initialize process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | void r_csi01_create (void); |
| **Description** | This function initializes CSI01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_start

| | |
|---|---|
| **Outline** | CSI01 operation start process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | void r_csi01_start (void); |
| **Description** | This function enables the CSI01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_stop

| | |
|---|---|
| **Outline** | CSI01 operation stop process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | void r_csi01_stop (void); |
| **Description** | This function stops the CSI01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_send

| | |
|---|---|
| **Outline** | CSI01 start sending process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | MD_STATUS r_csi01_send (uint8_t *const tx_buf, uint16_t tx_num); |
| **Description** | This function sends the number of data specified by the argument tx_num from the address specified by the argument tx_buf.<br>This function does not perform error detection. |
| **Arguments** | tx_buf, tx_num |
| **Return value** | status |
| **Remarks** | None |

[Function name] r_csi01_callback_sendend

| | |
|---|---|
| **Outline** | CSI01 callback function for end of transmission process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | static void r_csi01_callback_sendend (void); |
| **Description** | This function stops the output of ELCL, sets g_tx_done_flag and turns on LED1. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_interrupt

| | |
|---|---|
| **Outline** | CSI01 interrupt process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | #pragma interrupt r_csi01_interrupt (vect=INTCSI01) |
| **Description** | This function sets the next transmission data to SIO01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_create

| | |
|---|---|
| **Outline** | ELCL initialize process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_create (void); |
| **Description** | This function initializes ELCL. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_start

| | |
|---|---|
| **Outline** | ELCL operation start process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_start (void); |
| **Description** | This function enables the ELCL output. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_stop

| | |
|---|---|
| **Outline** | ELCL operation stop process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_stop (void); |
| **Description** | This function stops the ELCL output. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_reset_flipflop

| | |
|---|---|
| **Outline** | ELCL flip-flop reset process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_reset_flipflop (void); |
| **Description** | This function resets the flip-flops to reset the ELOMONI flag. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | Flip-flops are reset when bit 6 and bit 7 of ELLnCTL are set to 0. |

[Function name] r_elcl_start_port_move

| | |
|---|---|
| **Outline** | ELCL terminal output destination change start process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_start_port_move (void); |
| **Description** | This function executes R_Config_Through_Stop and starts the terminal output destination change processing. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_stop_port_move

| | |
|---|---|
| **Outline** | ELCL terminal output destination change stop process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_stop_port_move (void); |
| **Description** | This function executes R_Config_Through_Stop and stops the terminal output destination change processing. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_create

| | |
|---|---|
| **Outline** | IICA0 initialize process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_csi01_create (void); |
| **Description** | This function initializes IICA0. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_stop

| | |
|---|---|
| **Outline** | IICA0 operation stop process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_iica0_stop (void); |
| **Description** | This function stops IICA0 operation. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_stopcondition

| | |
|---|---|
| **Outline** | IICA0 stop condition process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_iica0_stopcondition (void); |
| **Description** | This function generates stop conditions. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_master_send

| | |
|---|---|
| **Outline** | IICA0 master send start process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | MD_STATUS r_iica0_master_send (uint8_t adr, uint8_t *const tx_buf, uint16_t tx_num, uint8_t wait); |
| **Description** | (1) When the I²C bus is released, the start condition is generated.<br>(2) After the start condition is generated, the slave address is set to transmit mode and data transmission is started.<br>(3) Waits for communication completion. |
| **Arguments** | adr, tx_buf, tx_num, wait |
| **Return value** | status |
| **Remarks** | None |

[Function name] r_iica0_callback_master_sendend

| | |
|---|---|
| **Outline** | IICA0 callback function for end of transmission process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | static void r_iica0_callback_master_sendend (void); |
| **Description** | This function generates stop condition and sets IIC communication status to FINISH. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_master_handler

| | |
|---|---|
| **Outline** | IICA0 interrupt handler |
| **Header** | iica.h, r_codec.h |
| **Declaration** | static void r_iica0_master_handler (void); |
| **Description** | This function sends data if it detects an ACK for a slave address. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_interrupt

| | |
|---|---|
| **Outline** | IICA0 interrupt process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | #pragma interrupt r_iica0_interrupt (vect=INTIICA0,enable=true) |
| **Description** | This function accepts and processes IICA0 interrupt requests.<br>Permits multiple interrupts. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_codec_init

| | |
|---|---|
| **Outline** | CODEC module setup process |
| **Header** | sound_data.h, iica0.h, r_codec.h, Config_TAU0_7.h |
| **Declaration** | void r_codec_init (void); |
| **Description** | (1) Performs the initial setup process for IICA0.<br>(2) Setup the CODEC module.<br>(3) Waits for setup completion (100ms). |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_codec_start

| | |
|---|---|
| **Outline** | CODEC module start process |
| **Header** | sound_data.h, iica0.h, r_codec.h, Config_TAU0_7.h |
| **Declaration** | void r_codec_start (void); |
| **Description** | This function starts the operation of the CODEC module. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_codec_stop

| | |
|---|---|
| **Outline** | CODEC module stop process |
| **Header** | sound_data.h, iica0.h, r_codec.h, Config_TAU0_7.h |
| **Declaration** | void r_codec_stop (void); |
| **Description** | This function stops the operation of the CODEC module. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_tau0_start

| | |
|---|---|
| **Outline** | TAU0 start process |
| **Header** | r_cg_macrodriver.h, r_cg_userdefine.h, Config_TAU0_0.h, Config_TAU0_1.h<br>Config_TAU0_7.h,r_cg_tau_common.h |
| **Declaration** | void r_tau0_start (void); |
| **Description** | This function start counters for TAU00 and TAU01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] R_Config_TAU0_0_Create_UserInit

| | |
|---|---|
| **Outline** | TAU0 initialization user code addition process |
| **Header** | r_cg_macrodriver.h, r_cg_userdefine.h, Config_TAU0_0.h |
| **Declaration** | void R_Config_TAU0_0_Create_UserInit (void); |
| **Description** | This function sets the initial output of TAU00 to 1 and the initial output of TAU01 to 0. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_ms_delay

| | |
|---|---|
| **Outline** | Wait process |
| **Header** | r_cg_macrodriver.h, r_cg_userdefine.h, Config_TAU0_7.h |
| **Declaration** | void r_ms_delay (uint16_t msec); |
| **Description** | This function waits for the time (ms) specified by the argument msec.<br>This function counts using channel 7.Polls if g_ms_timer is less than msec, completes wait process if more than msec. |
| **Arguments** | msec |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_Config_TAU0_7_interrupt

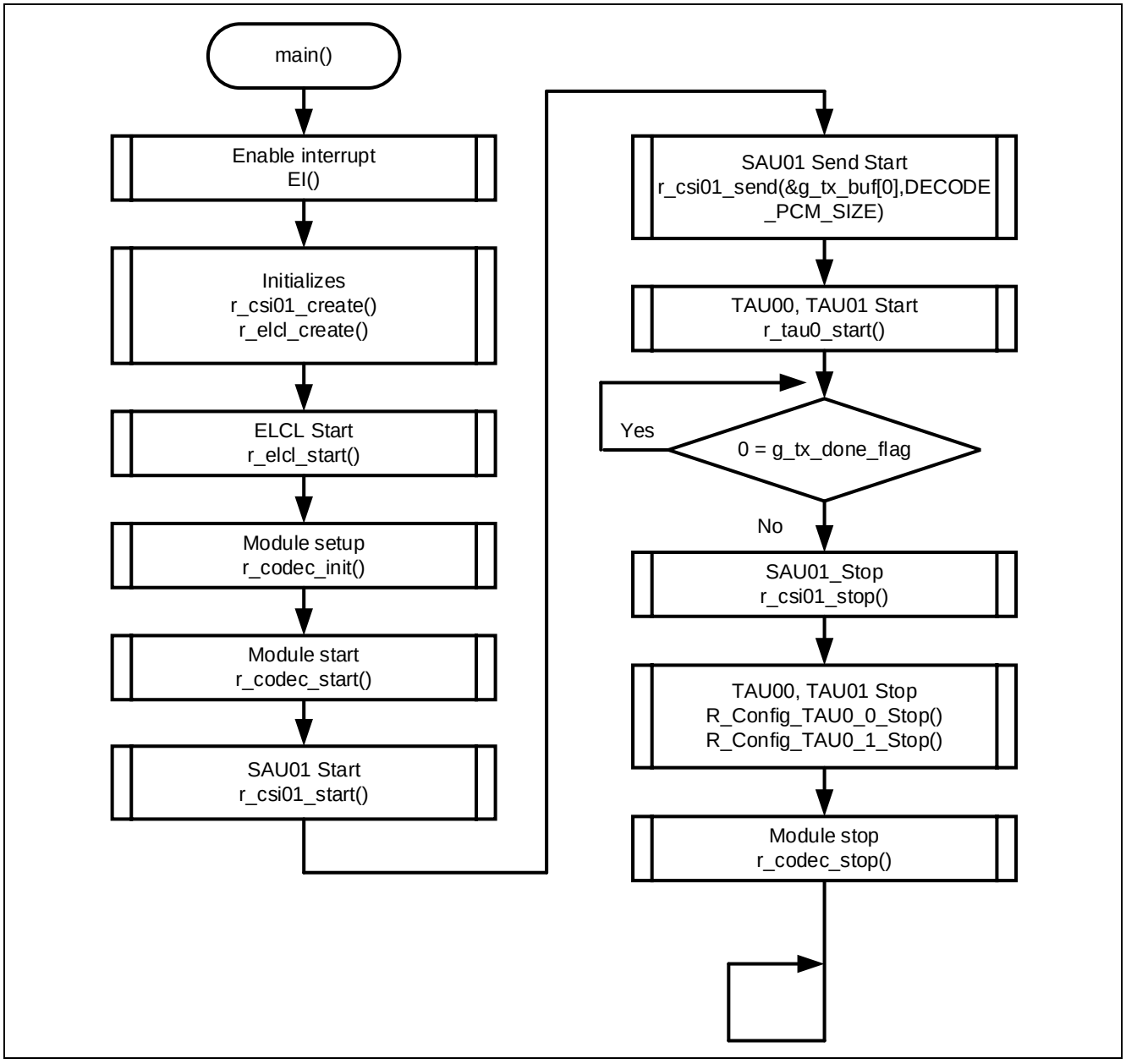
| | |
|---|---|
| **Outline** | TAU0 channel 7 interrupt process |
| **Header** | r_cg_macrodriver.h, r_cg_userdefine.h, Config_TAU0_7.h |
| **Declaration** | #pragma interrupt r_Config_TAU0_7_interrupt (vect=INTTM07) |
| **Description** | This function is an interrupt process by INTTM07 on TAU0 channel 7.<br>Counts up g_ms_timer. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

### 4.1.8 Flow Charts

#### 4.1.8.1 Main Process

Figure 4-2 shows flowchart of main process.

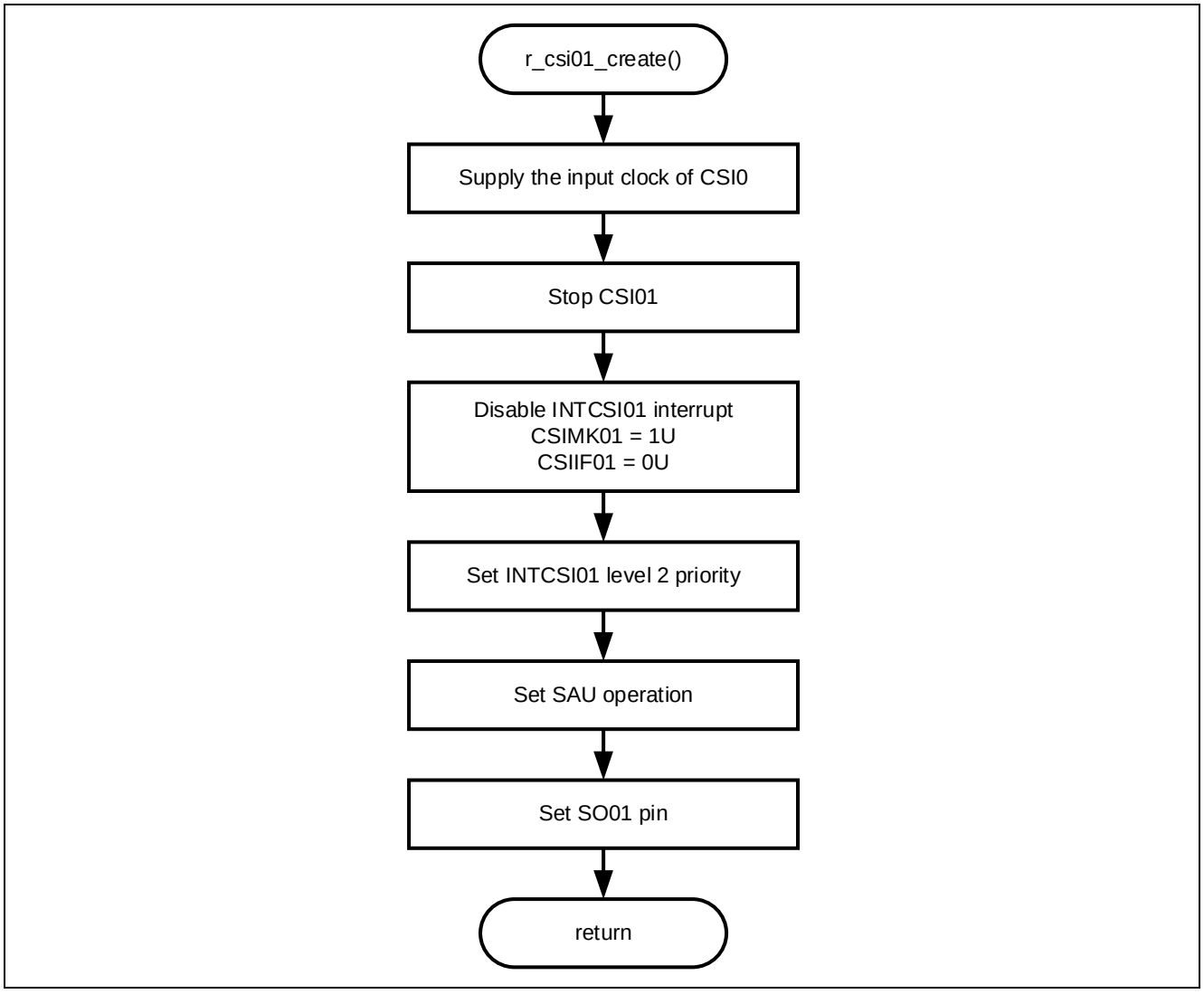
**Figure 4-2 Main process**



#### 4.1.8.2 CSI01 initialize process

Figure 4-3 shows flowchart of CSI01 initialize process.

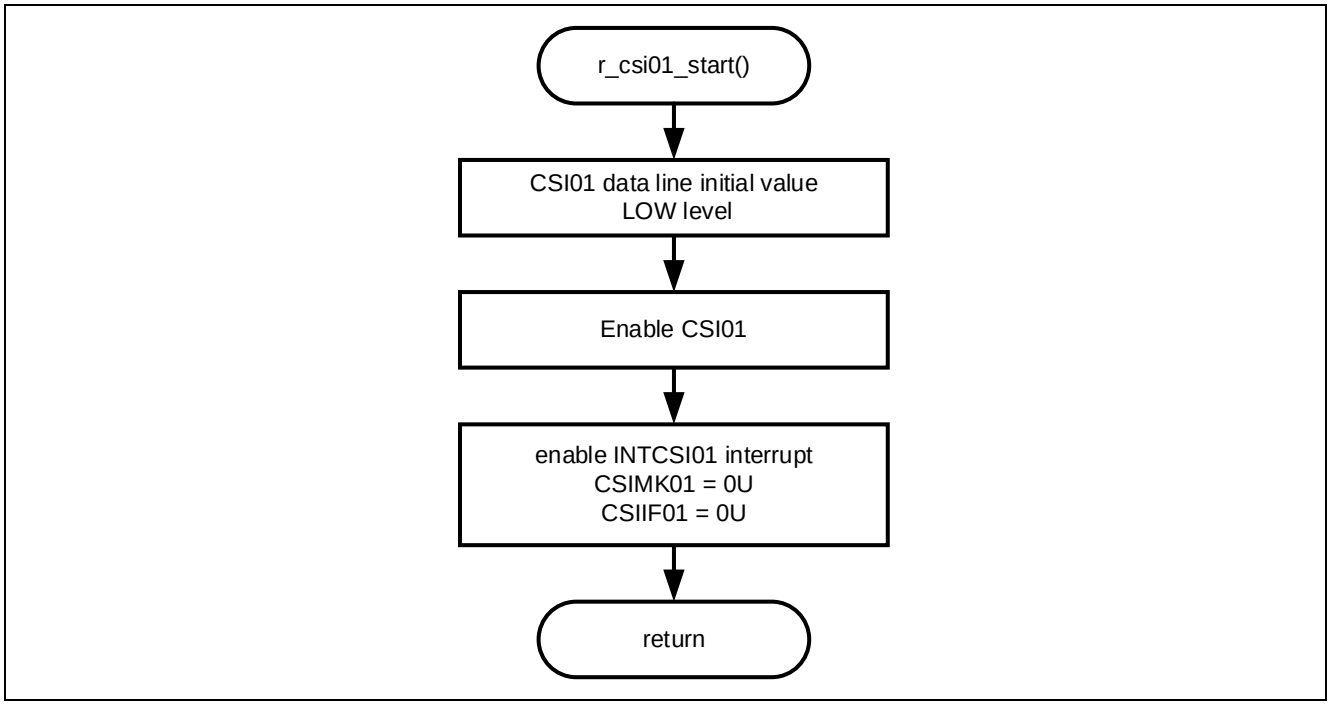
**Figure 4-3 CSI01 initialize process**



#### 4.1.8.3 CSI01 operation start process

Figure 4-4 shows flowchart of CSI01 operation start process.

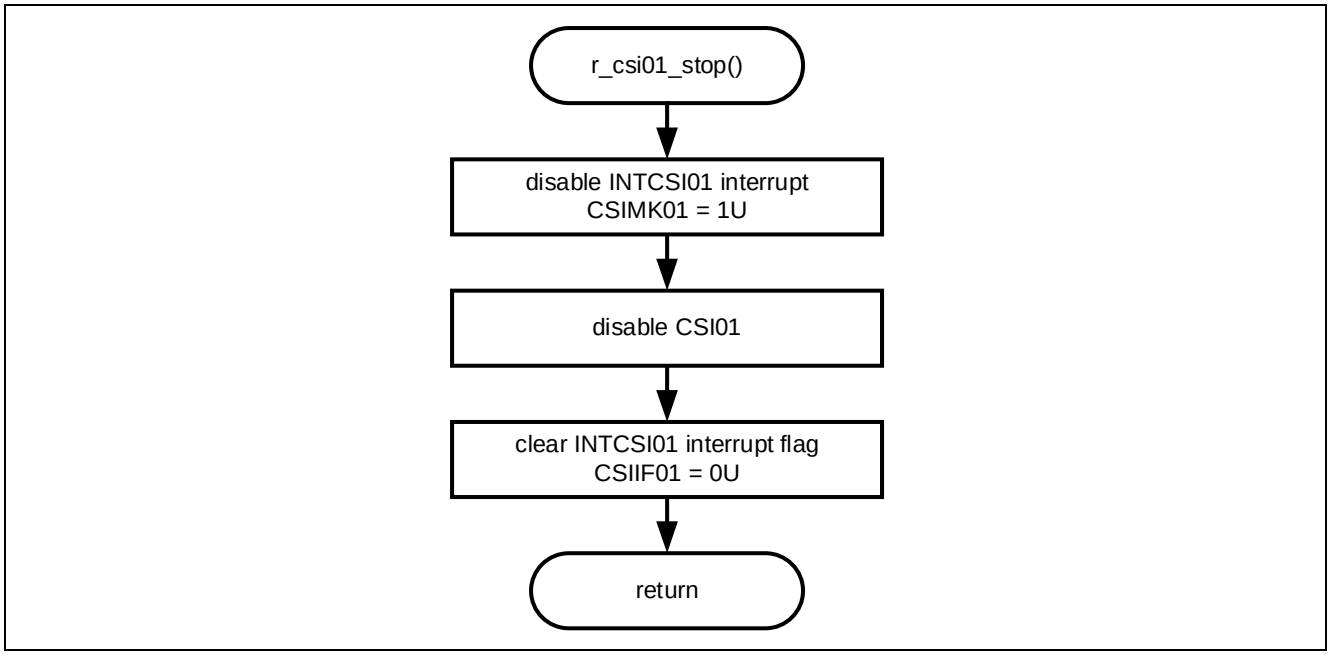
**Figure 4-4 CSI01 operation start process**




#### 4.1.8.4 CSI01 operation stop process

Figure 4-5 shows flowchart of CSI01 operation stop process.

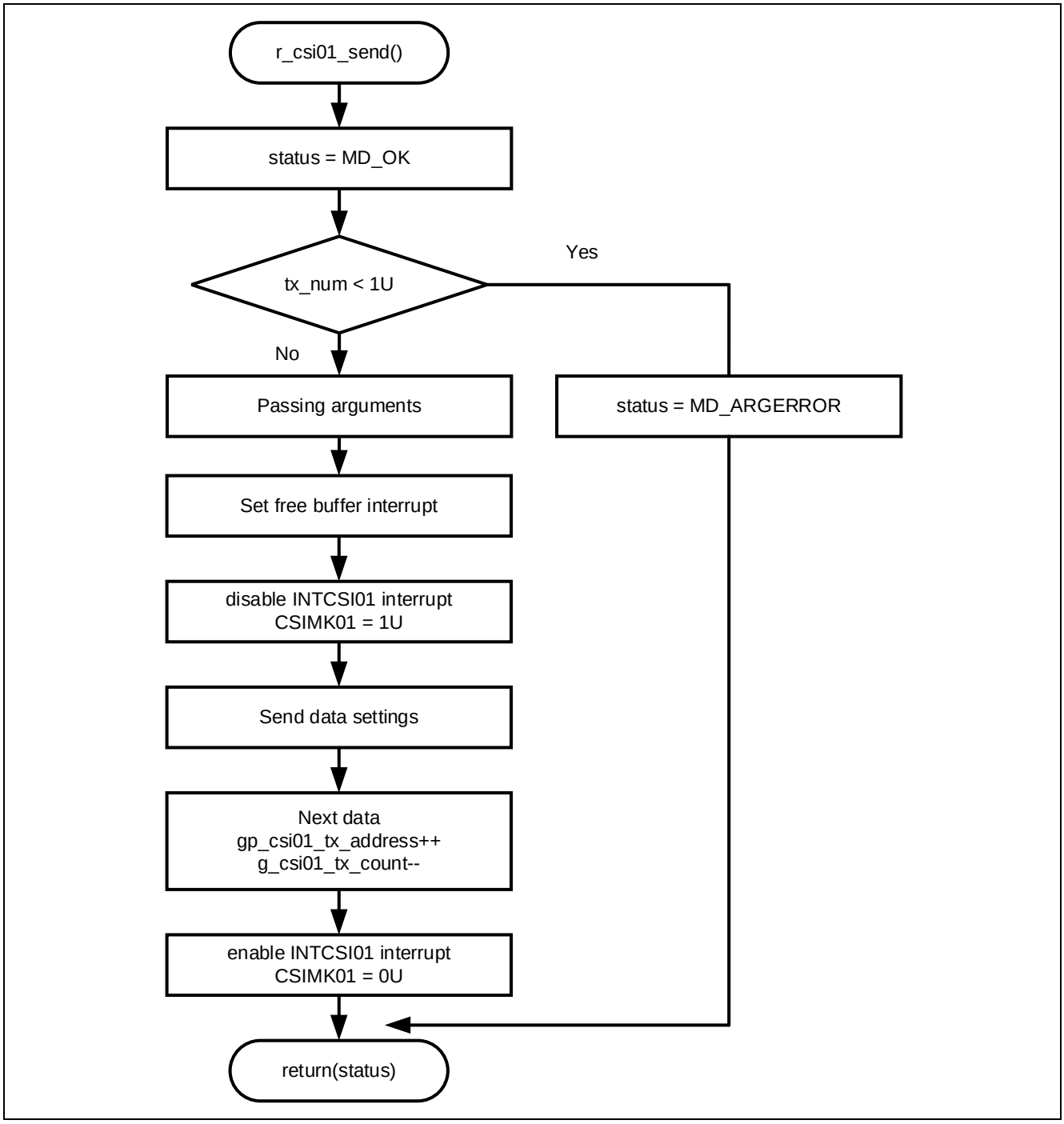
**Figure 4-5 CSI01 operation stop process**



#### 4.1.8.5 CSI01 send start process

Figure 4-6 shows flowchart of CSI01 start sending process.

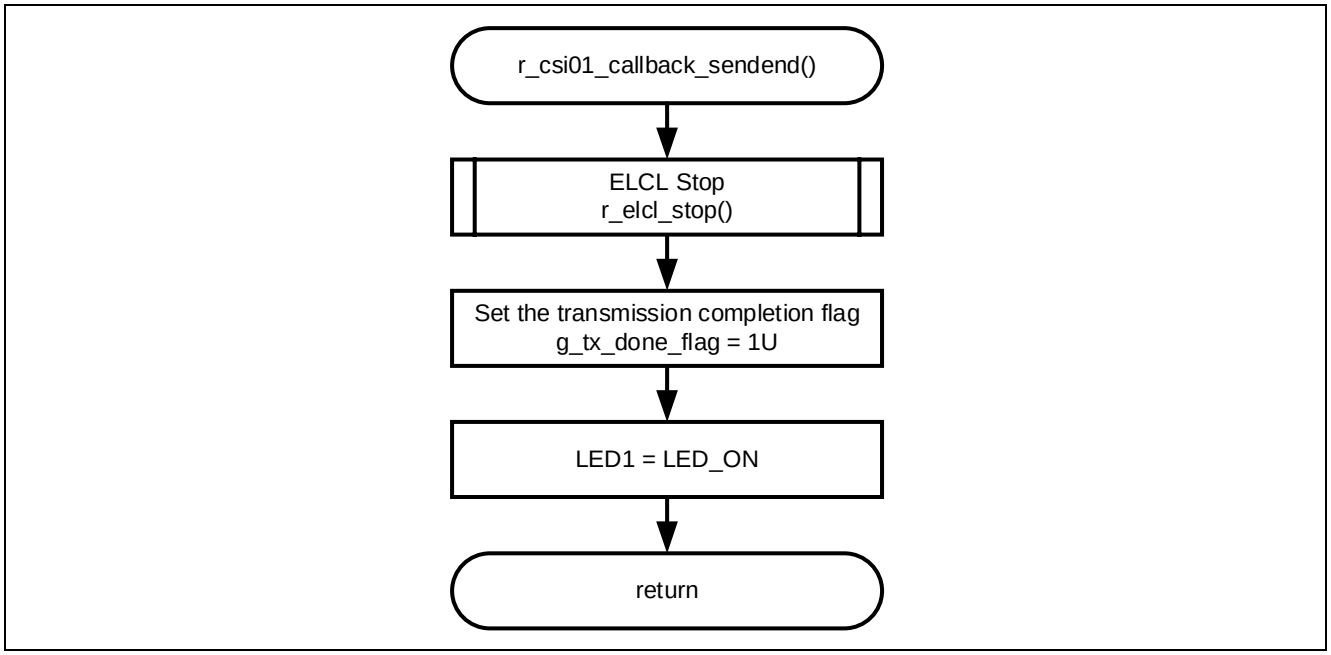
**Figure 4-6 CSI01 start sending process**



#### 4.1.8.6 CSI01 callback function for end of transmission process

Figure 4-7 shows flowchart of CSI01 callback function for end of transmission process.

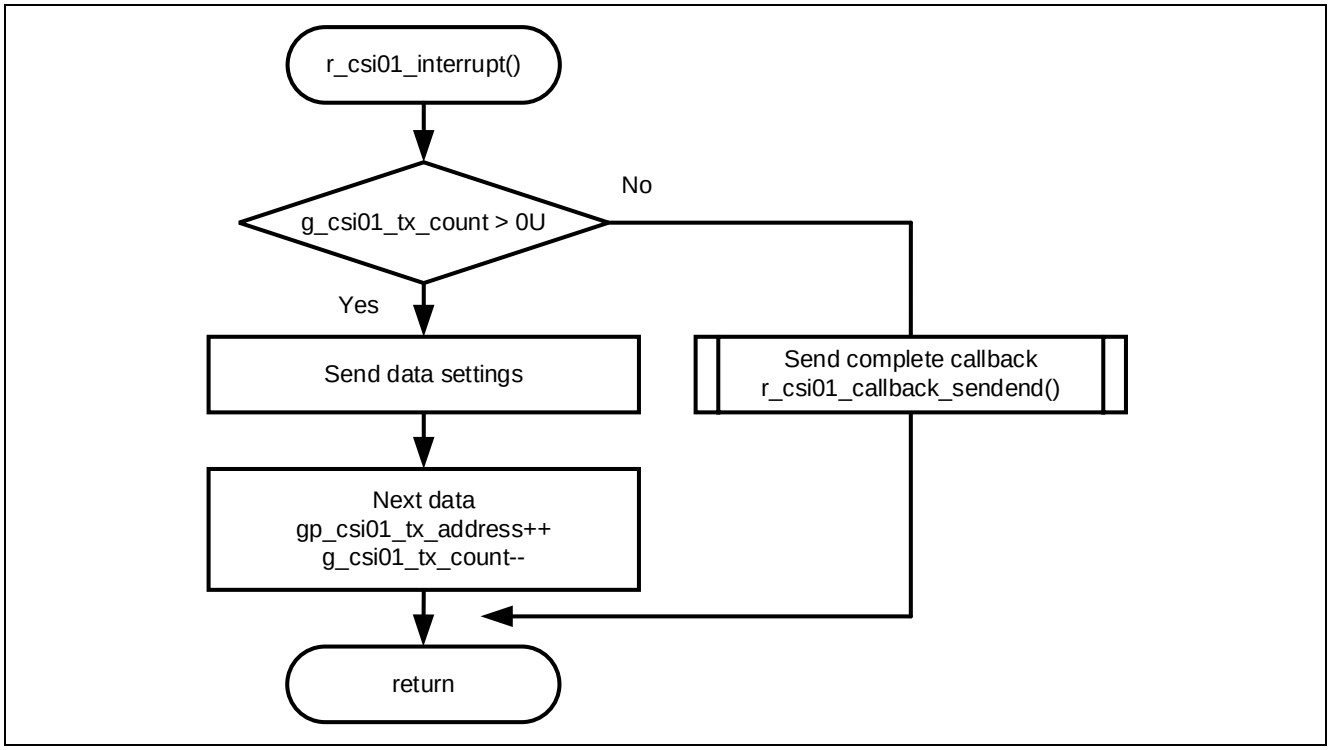
**Figure 4-7 CSI01 callback function for end of transmission process**



#### 4.1.8.7 CSI01 interrupt process

Figure 4-8 shows flowchart of CSI01 interrupt process.

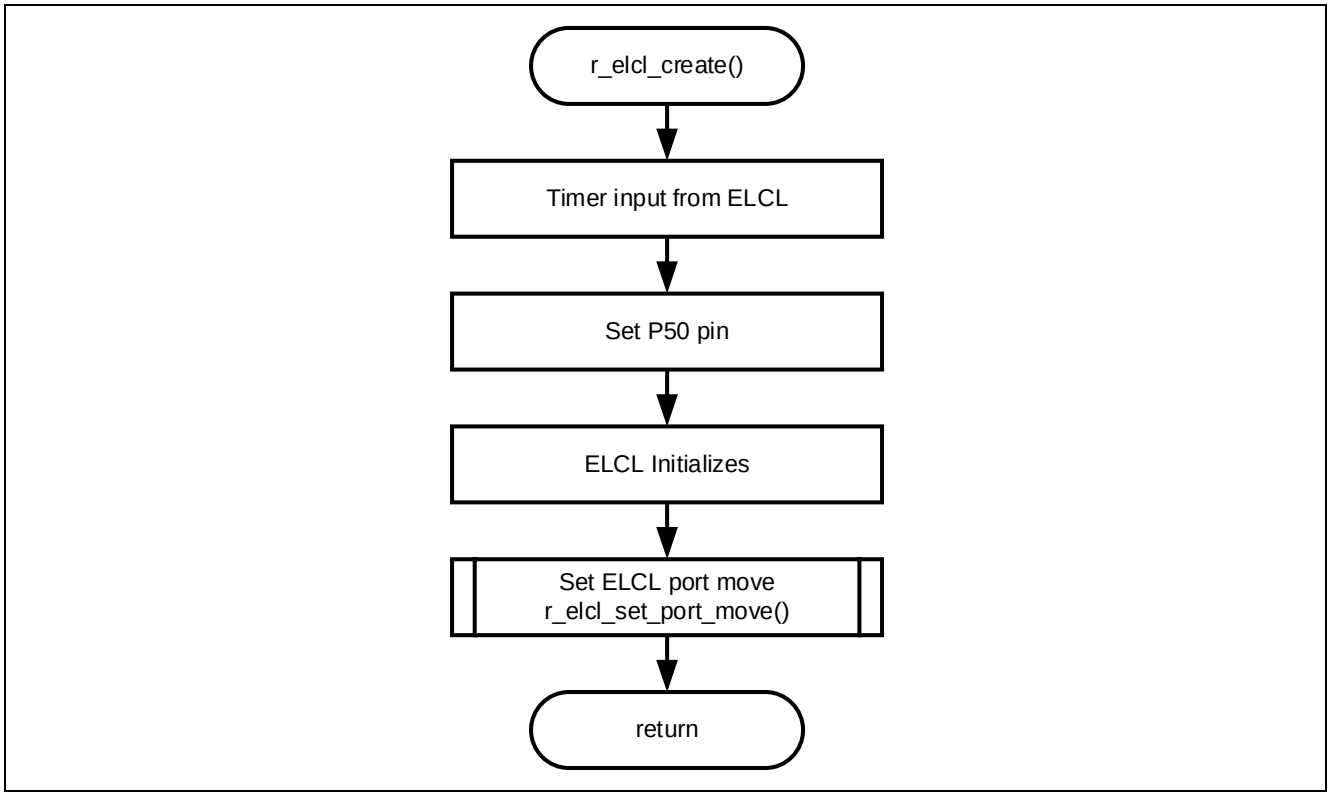
**Figure 4-8 CSI01 interrupt process**



#### 4.1.8.8 ELCL initialize process

Figure 4-9 shows flowchart of ELCL initialize process.

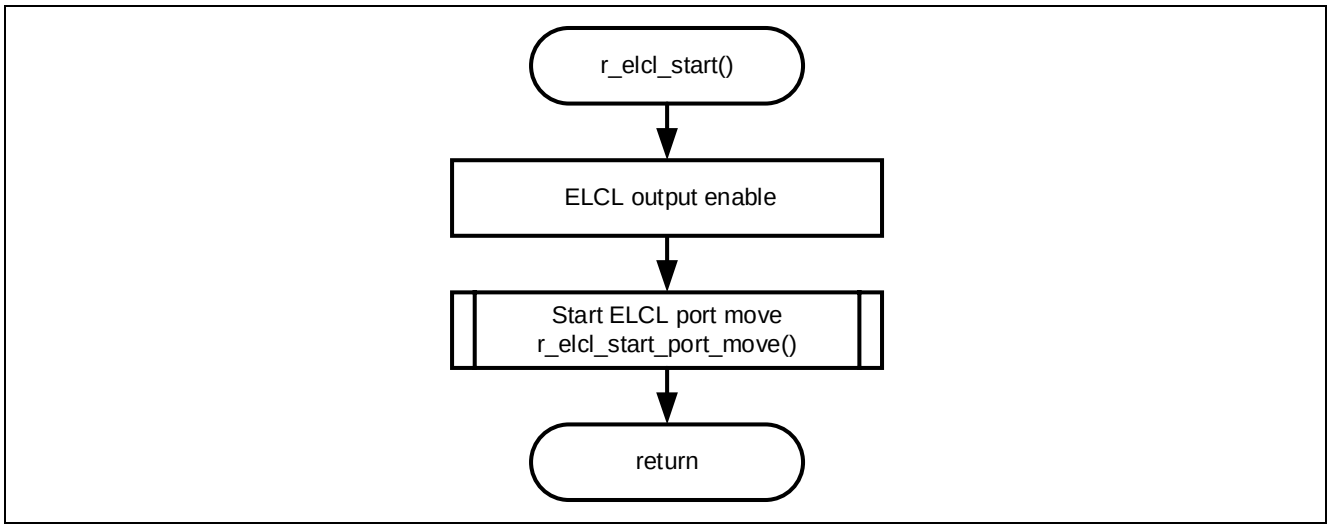
**Figure 4-9 ELCL initialize process**




#### 4.1.8.9 ELCL operation start process

Figure 4-10 shows flowchart of ELCL operation start process.

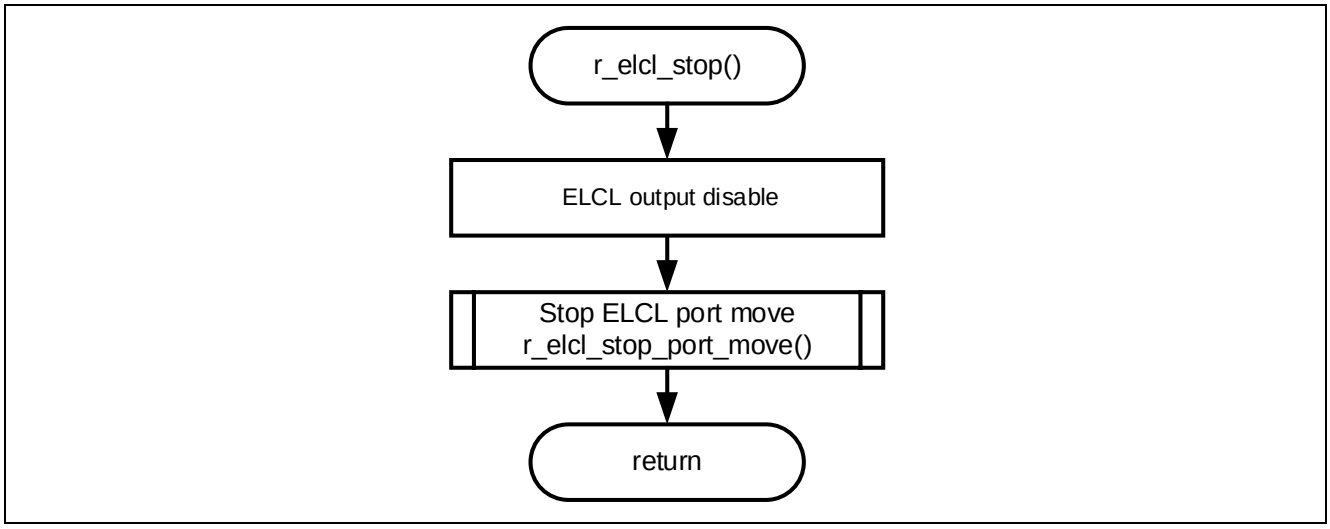
**Figure 4-10 ELCL operation start process**



### 4.1.8.10 ELCL operation stop process

Figure 4-11 shows flowchart of ELCL operation stop process.

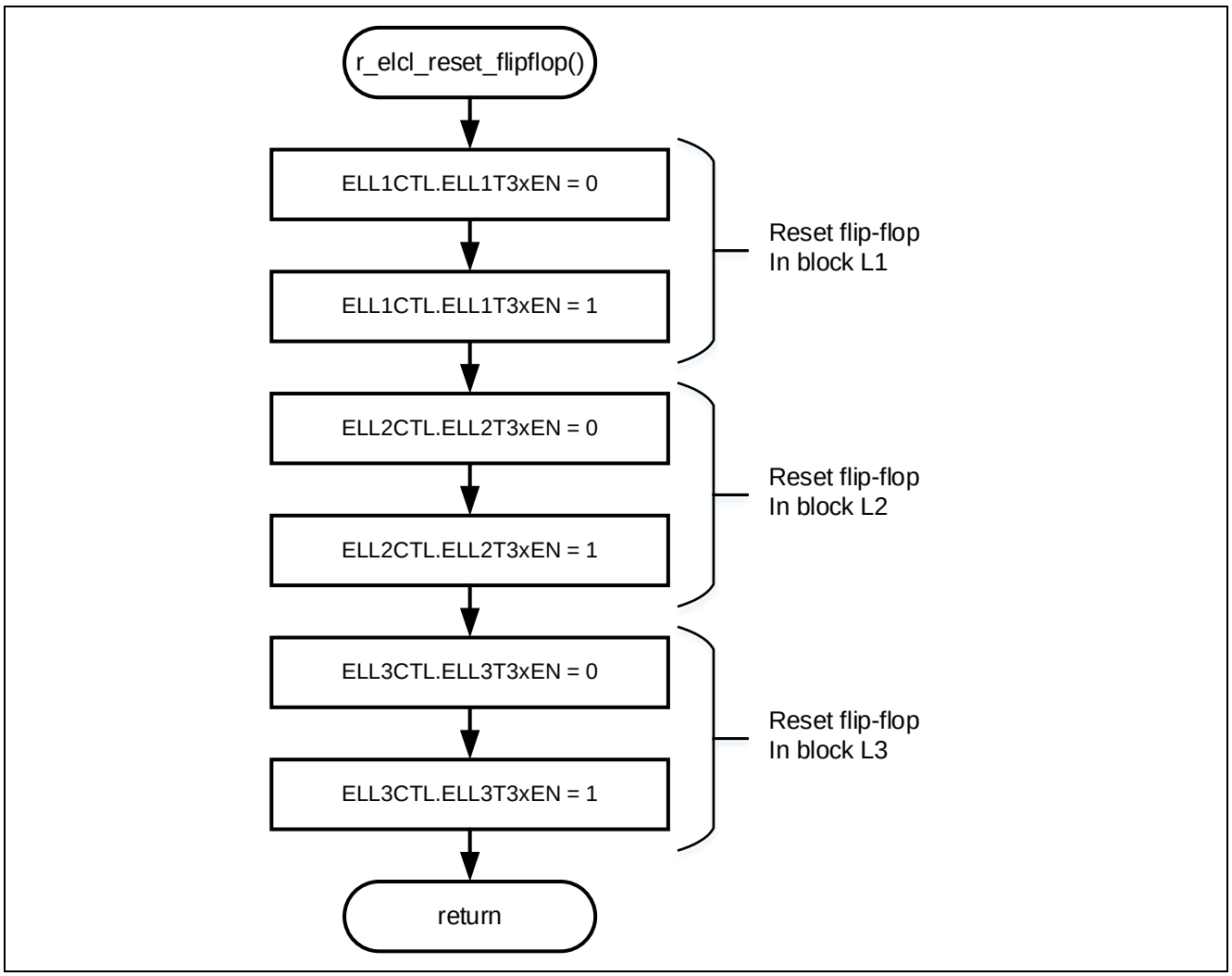
**Figure 4-11 ELCL operation stop process**



### 4.1.8.11 ELCL flip-flop reset process

Figure 4-12 shows flowchart of ELCL flip-flop reset process.

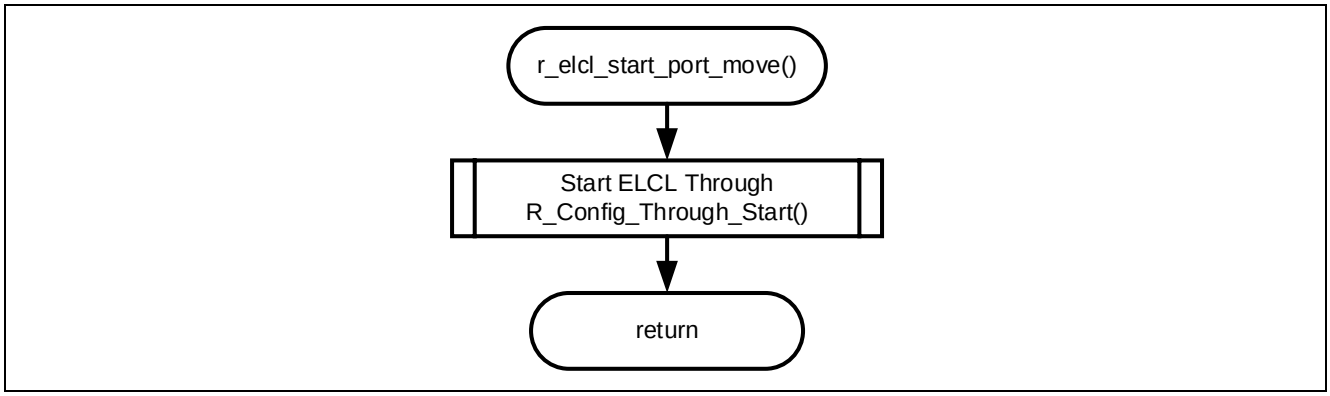
**Figure 4-12 ELCL flip-flop reset process**



#### 4.1.8.12 ELCL terminal output destination change start process

Figure 4-13 shows flowchart of ELCL terminal output destination change start process.

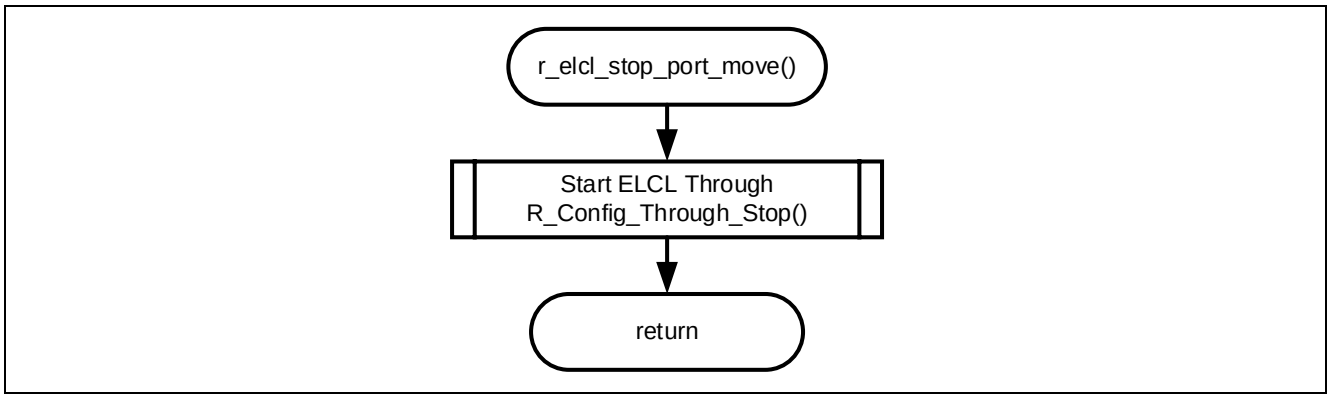
**Figure 4-13 ELCL terminal output destination change start process**




#### 4.1.8.13 ELCL terminal output destination change stop process

Figure 4-14 shows flowchart of ELCL terminal output destination change stop process.

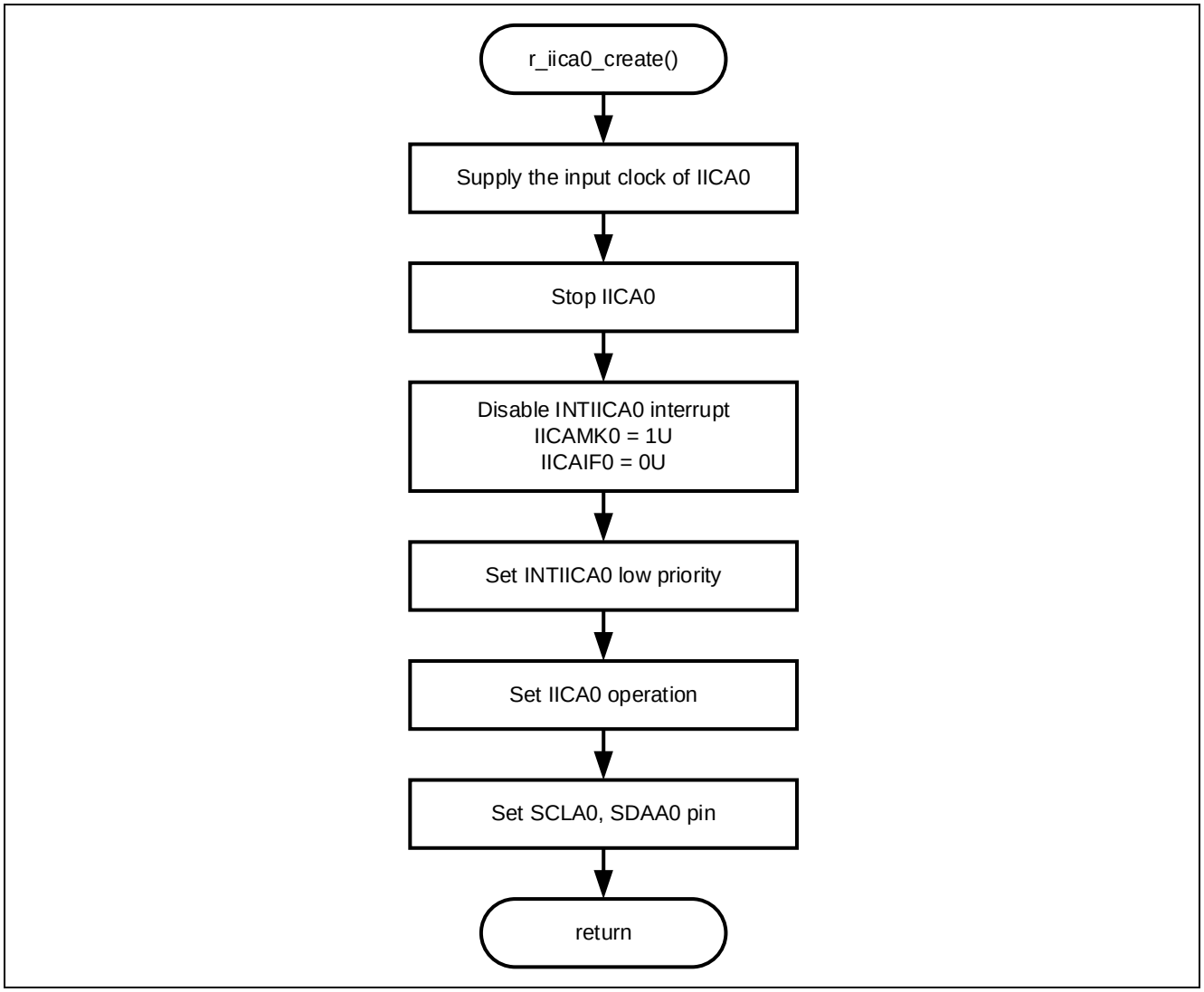
**Figure 4-14 ELCL terminal output destination change stop process**



### 4.1.8.14 IICA0 initialize process

Figure 4-15 shows flowchart of IICA0 initialize process.

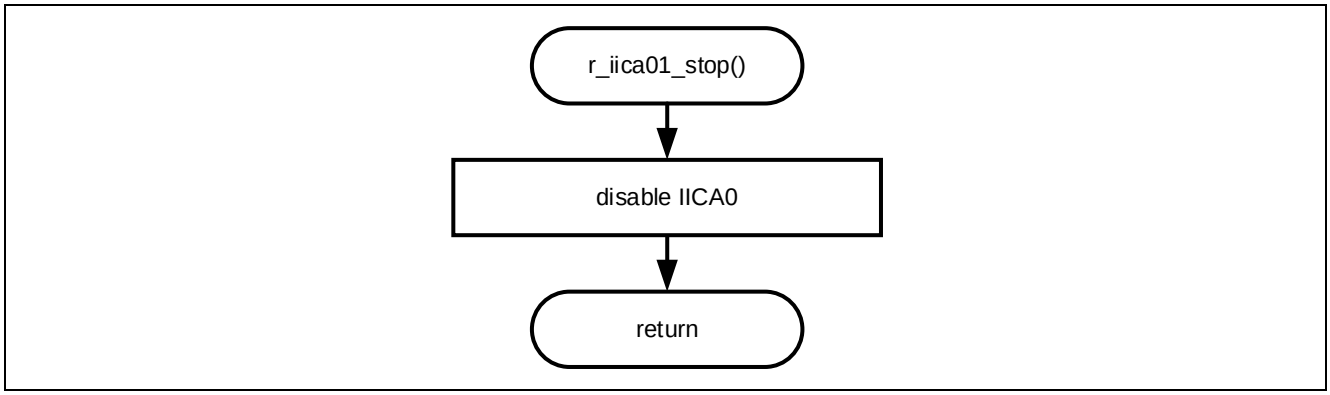
**Figure 4-15 IICA0 initialize process**



### 4.1.8.15 IICA0 operation stop process

Figure 4-16 shows flowchart of IICA0 operation stop process.

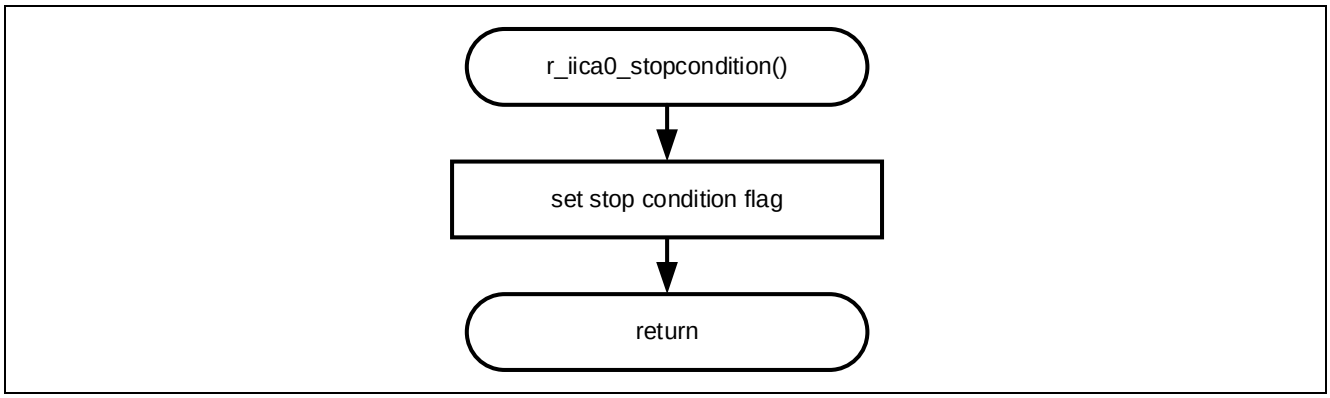
**Figure 4-16 IICA0 operation stop process**




### 4.1.8.16 IICA0 stop condition process

Figure 4-17 shows flowchart of IICA0 stop condition process.

**Figure 4-17 IICA0 stop condition process**



### 4.1.8.17 IICA0 master send start process

Figure 4-18, Figure 4-19 shows flowchart of IICA0 Send start process.

**Figure 4-18 IICA0 master send start process (1/2)**

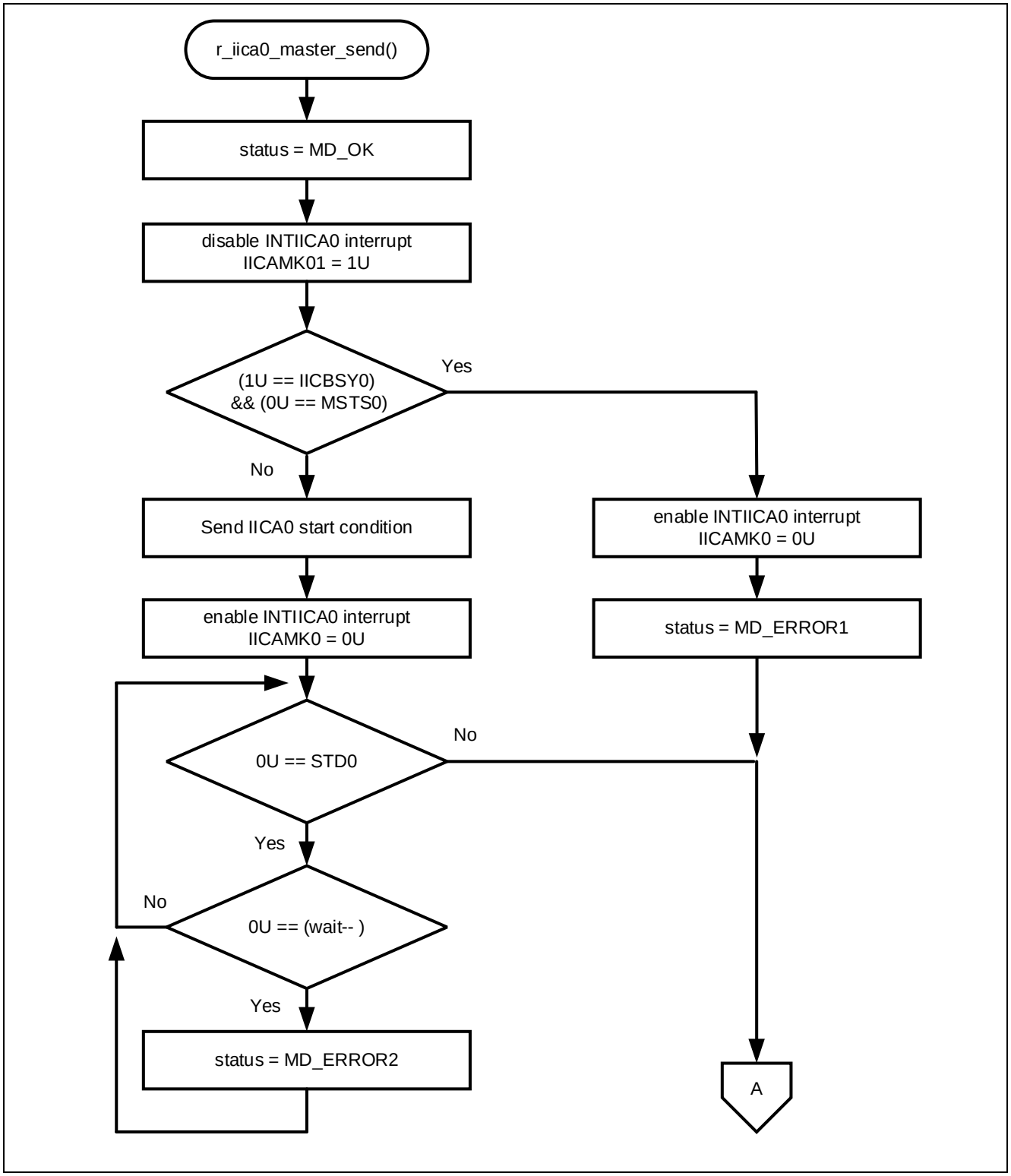
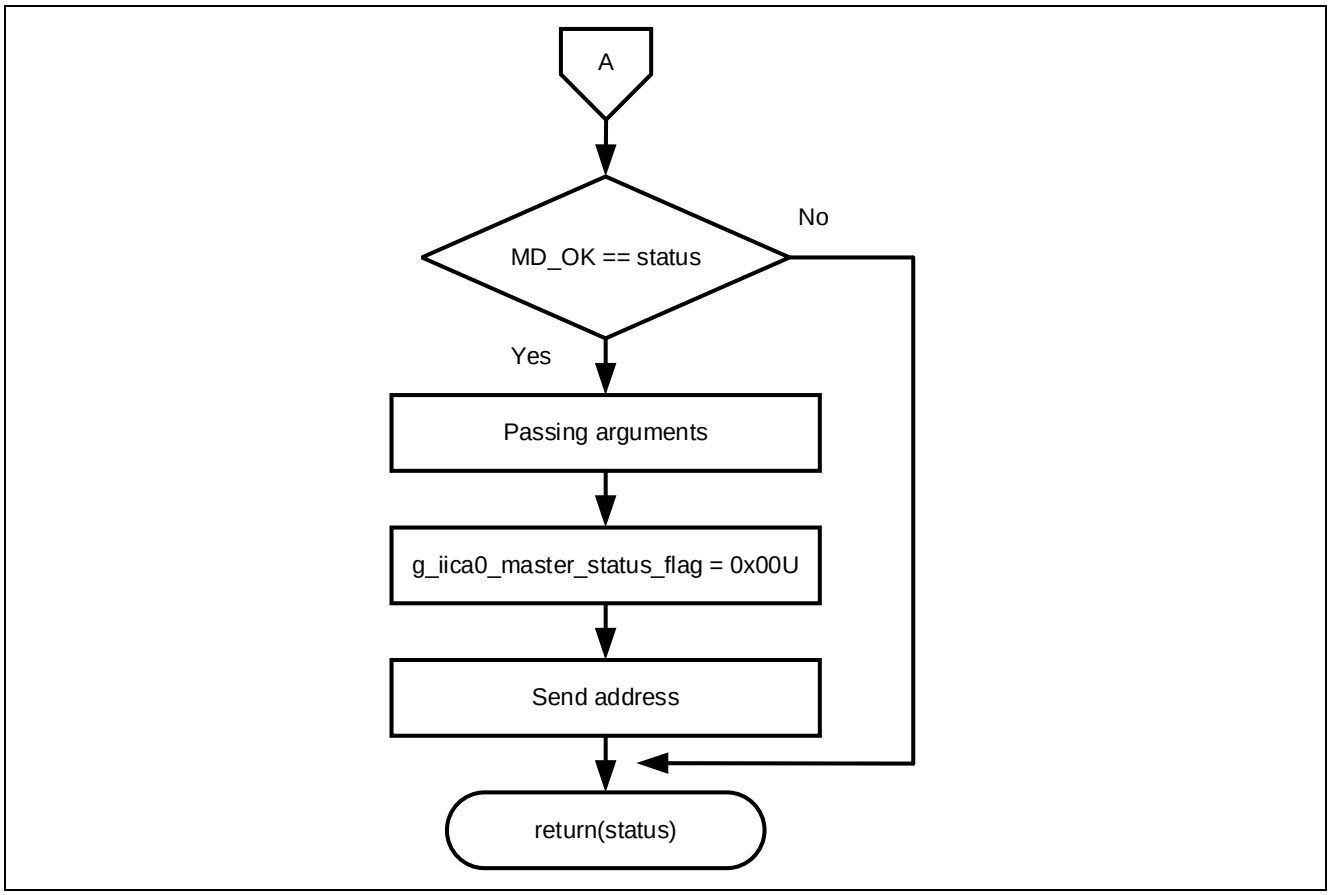


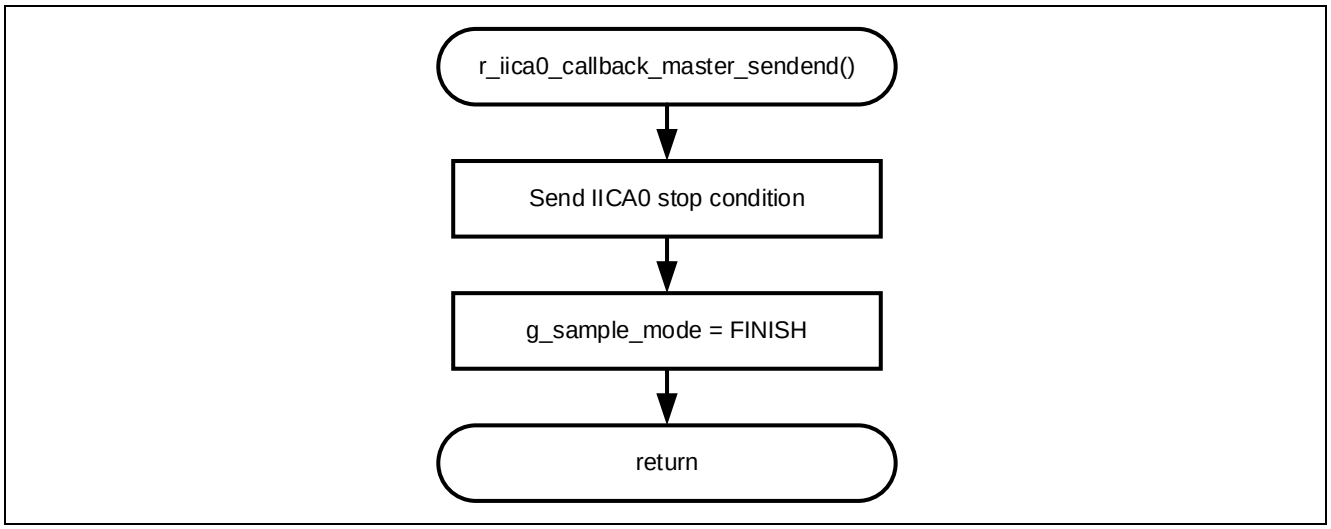
**Figure 4-19 IICA0 master send start process (2/2)**




### 4.1.8.18 IICA0 callback function for end of transmission process

Figure 4-20 shows flowchart of IICA0 callback function for end of transmission process.

**Figure 4-20 IICA0 callback function for end of transmission process**



### 4.1.8.19 IICA0 interrupt handler

Figure 4-21, Figure 4-22 shows flowchart of IICA0 interrupt handler.

**Figure 4-21 IICA0 interrupt handler (1/2)**

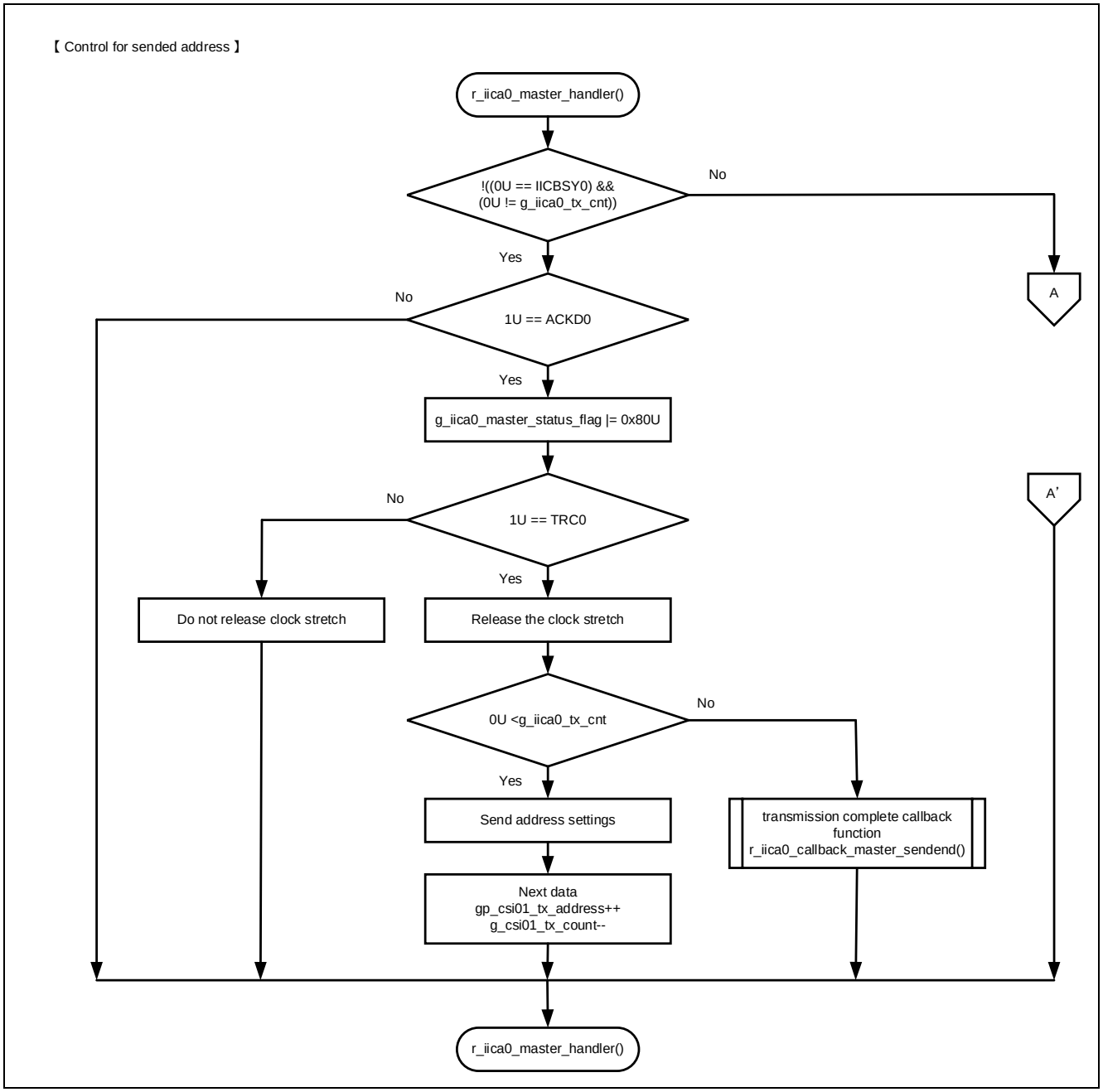
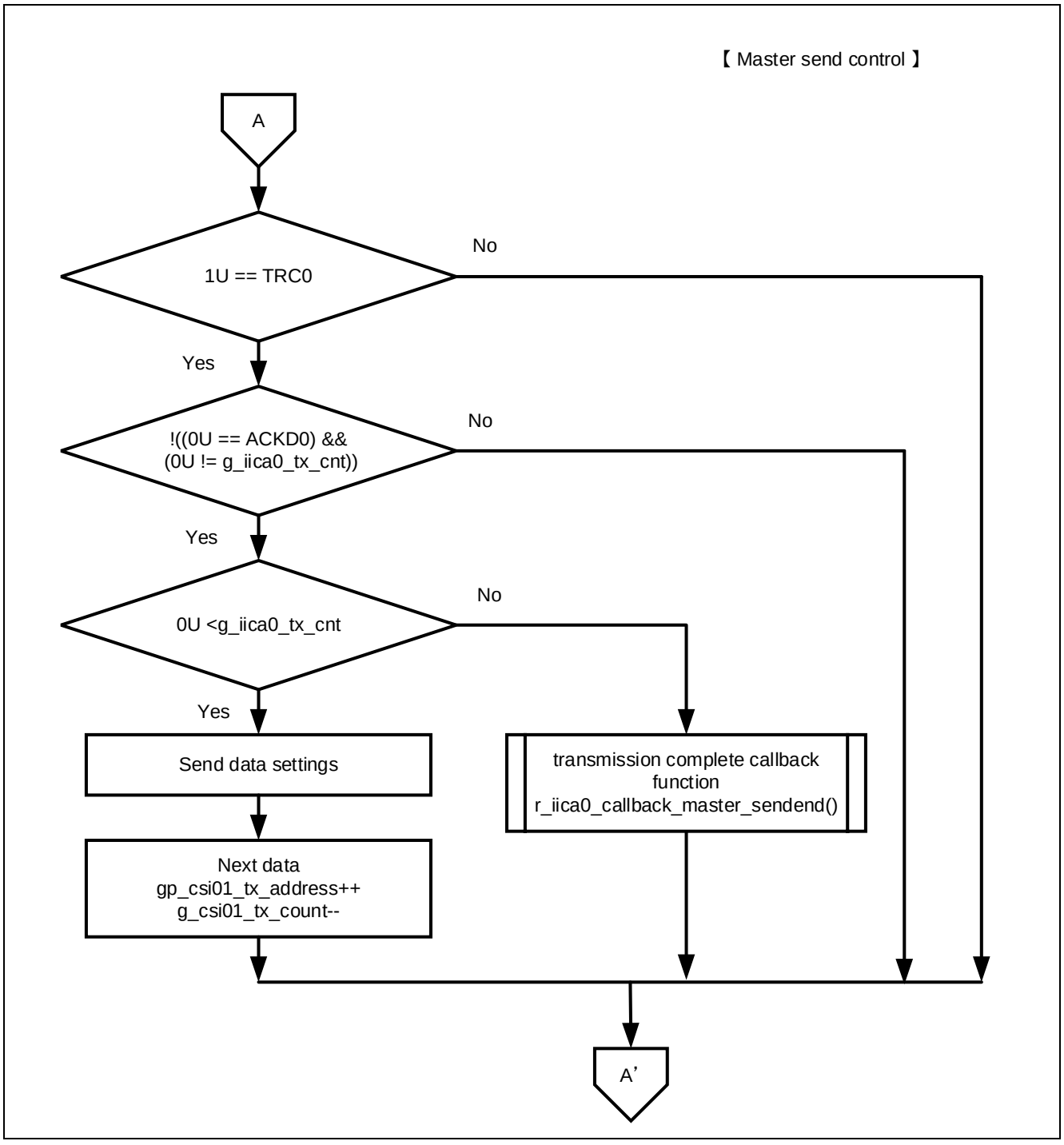


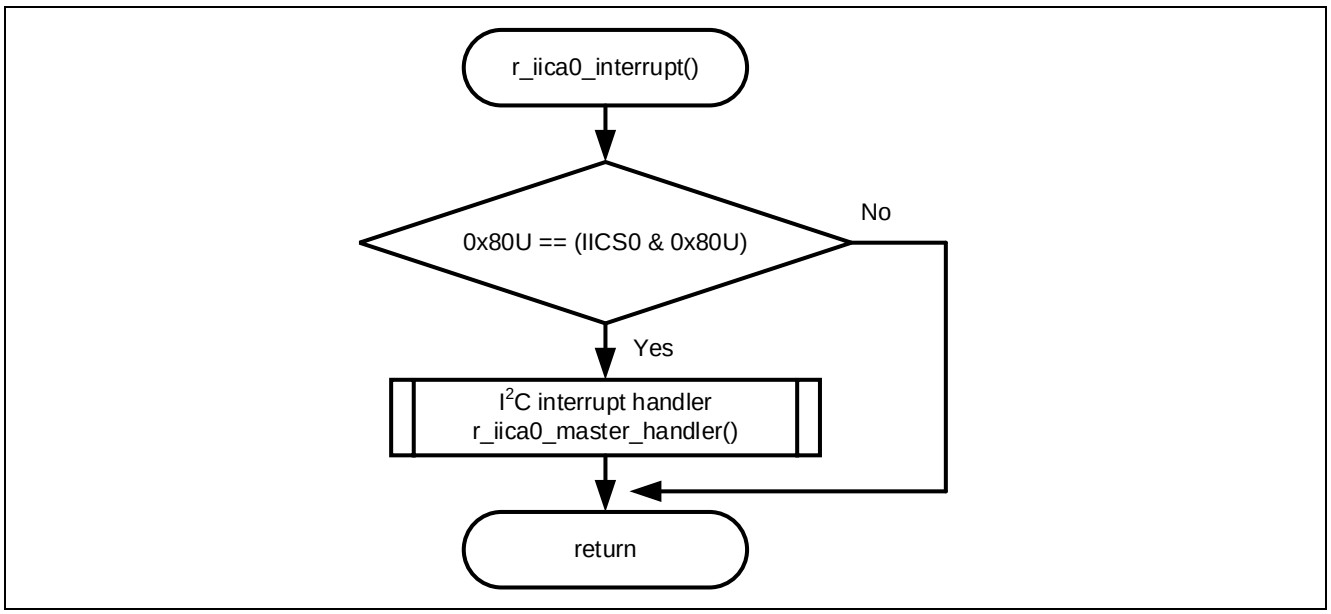
**Figure 4-22 IICA0 interrupt handler (2/2)**



#### 4.1.8.20 IICA0 interrupt process

Figure 4-23 shows flowchart of IICA0 interrupt process.

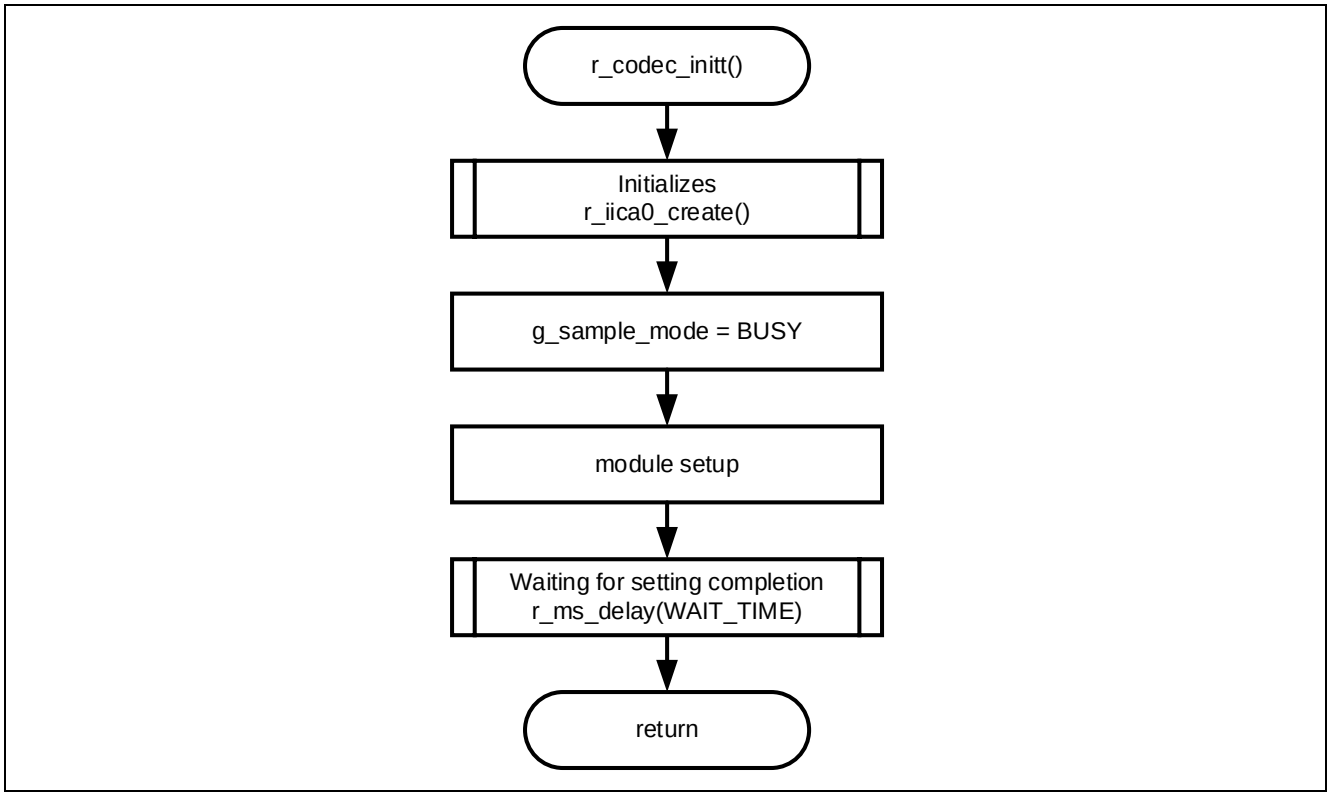
**Figure 4-23 IICA0 interrupt process**



### 4.1.8.21 CODEC module setup process

Figure 4-24 shows flowchart of CODEC module setup process.

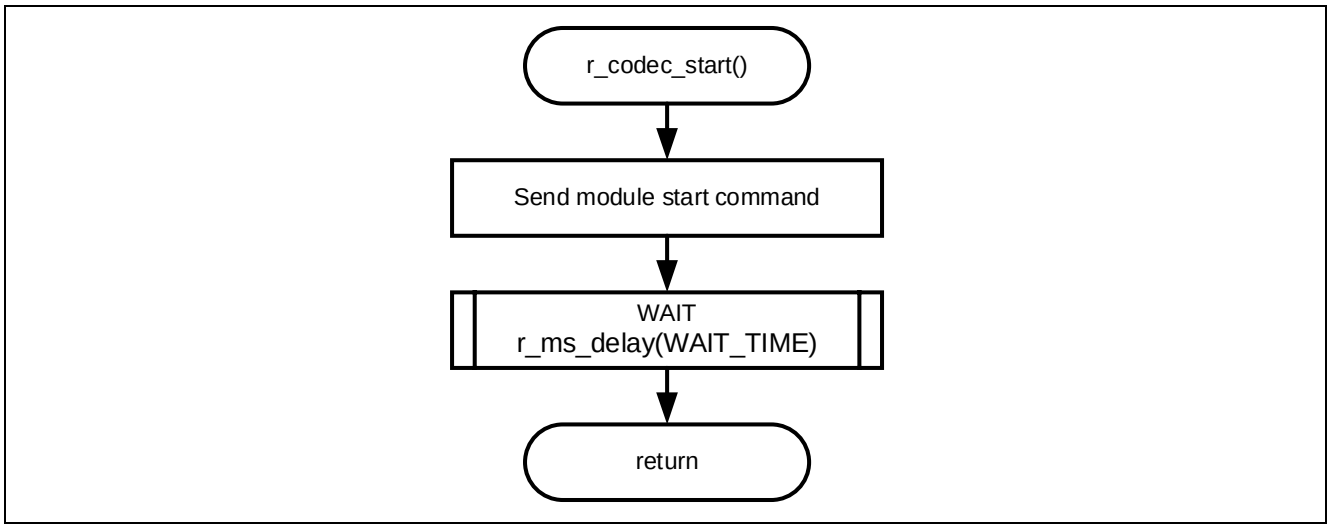
**Figure 4-24 CODEC module setup process**




### 4.1.8.22 CODEC module start process

Figure 4-25 shows flowchart of CODEC module start process.

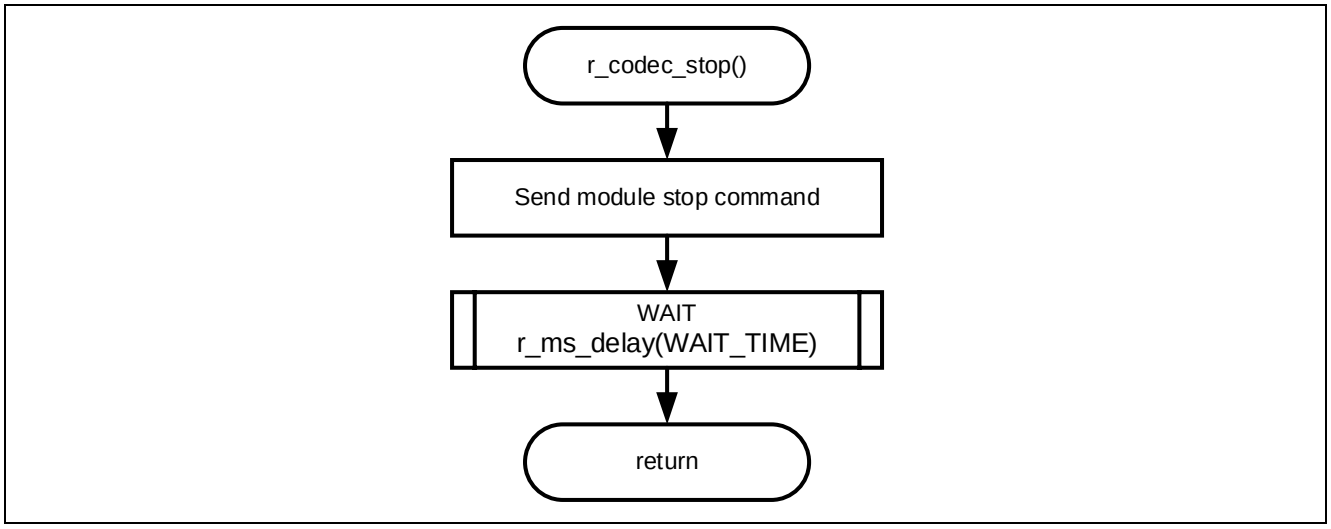
**Figure 4-25 CODEC module start process**



### 4.1.8.23 CODEC module stop process

Figure 4-26 shows flowchart of CODEC module stop process.

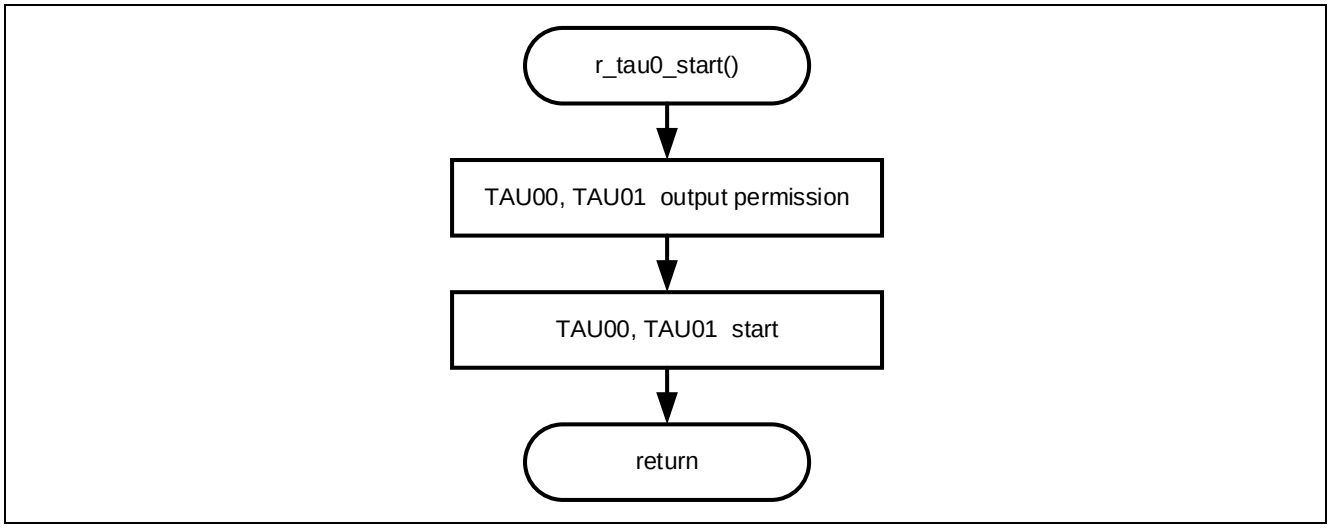
**Figure 4-26 CODEC module stop process**



#### 4.1.8.24 TAU0 start process

Figure 4-27 shows flowchart of TAU0 start process.

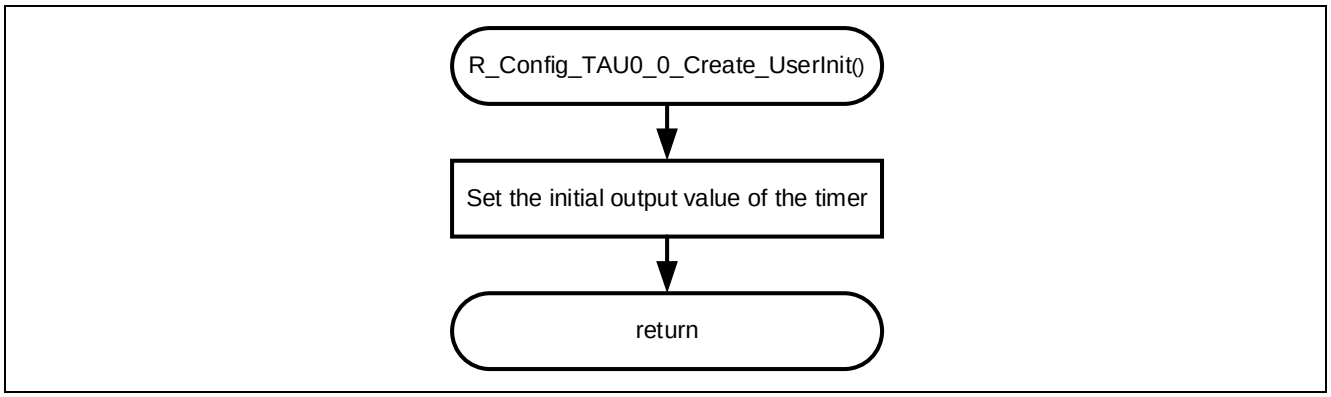
**Figure 4-27 TAU0 start process**




#### 4.1.8.25 TAU0 initialization user code addition process

Figure 4-28 shows flowchart of TAU0 initialization user code addition process.

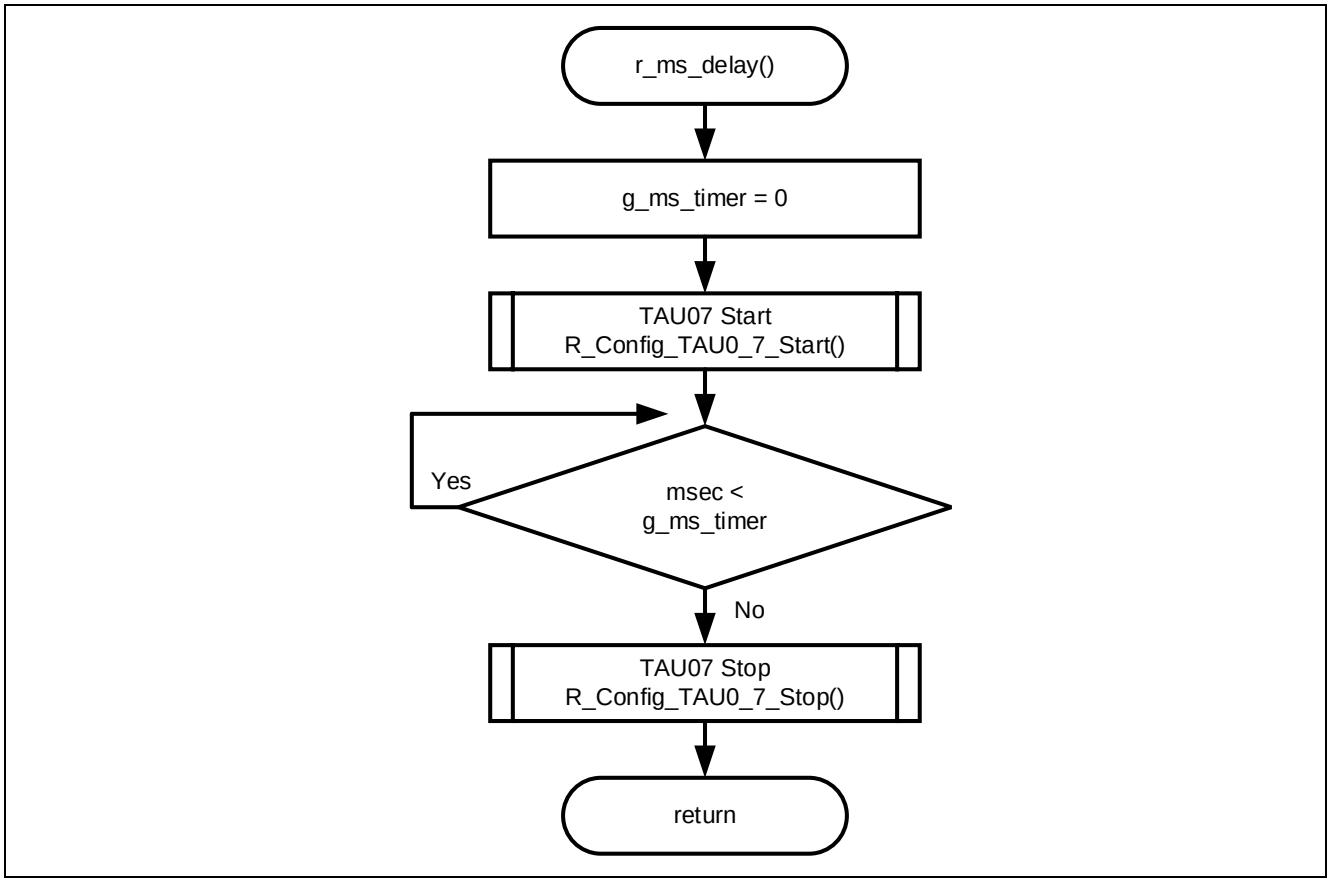
**Figure 4-28 TAU0 initialization user code addition process**



### 4.1.8.26 Wait process

Figure 4-29 shows flowchart of Wait process.

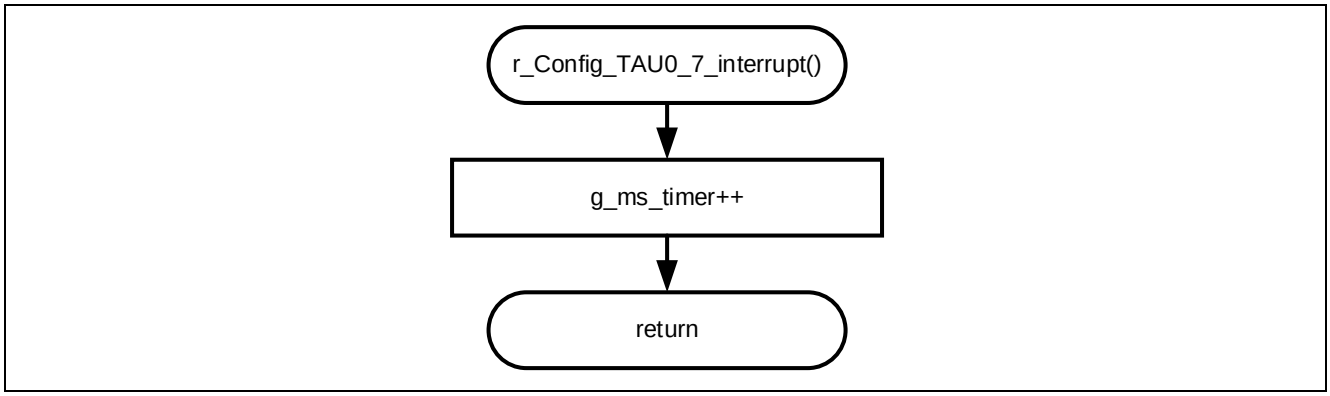
**Figure 4-29 Wait process**




### 4.1.8.27 TAU0 channel 7 interrupt process

Figure 4-30 shows flowchart of TAU0 channel 7 interrupt process

**Figure 4-30 TAU0 channel 7 interrupt process**



## 4.2 Application example

In addition to the sample code, this application note contains the following Smart Configurator configuration files

r01an6420_elcl_i2s_master.scfg

The following is a description of the file and examples of settings and notes for use.

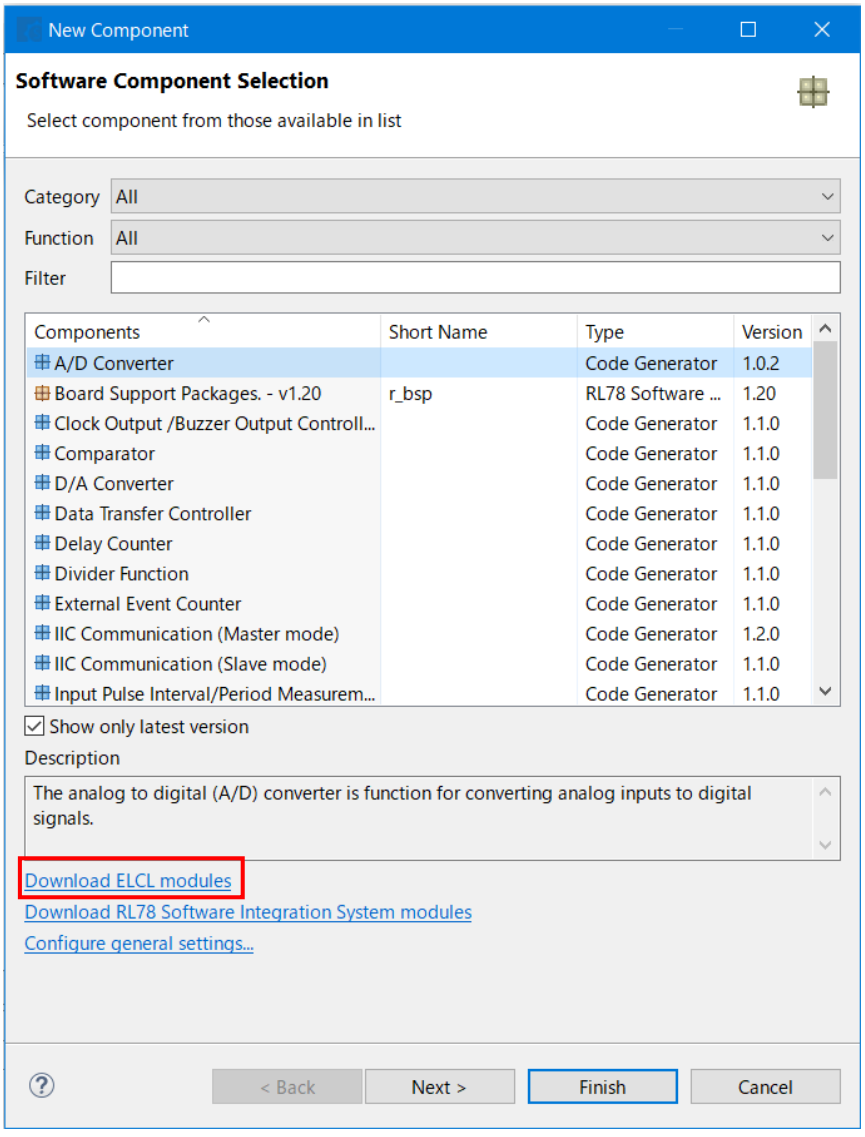
### 4.2.1 Setting up the ELCL components

To use the ELCL component, you need to install the ELCL content file.

The procedure is shown below.

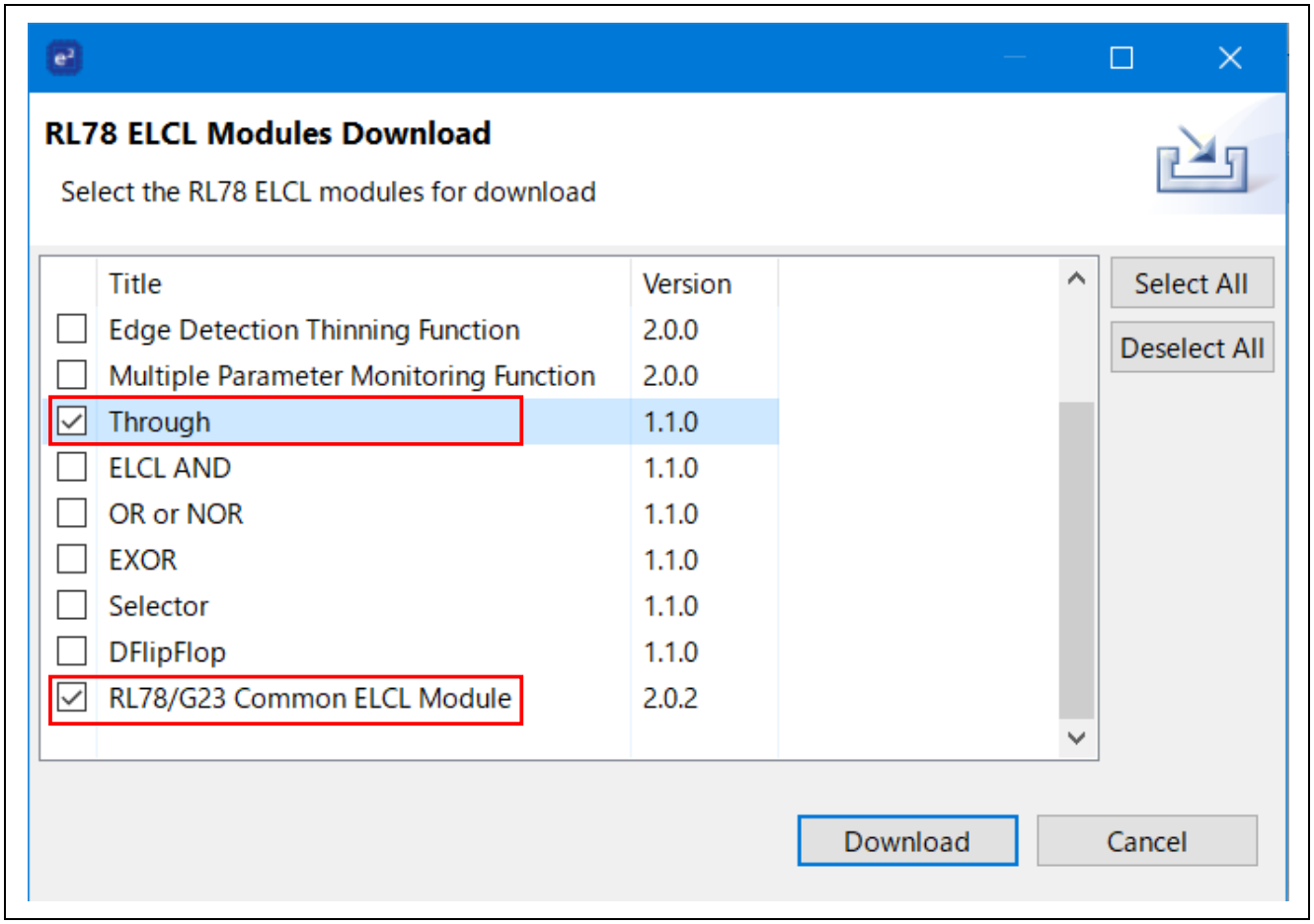
1. Start the Smart Configurator.
2. Click on the "Components" tag, and then click "Add component".
3. When the "New Component" window shown in Figure 4-31opens, click on "Download ELCL modules".

**Figure 4-31 Add component**



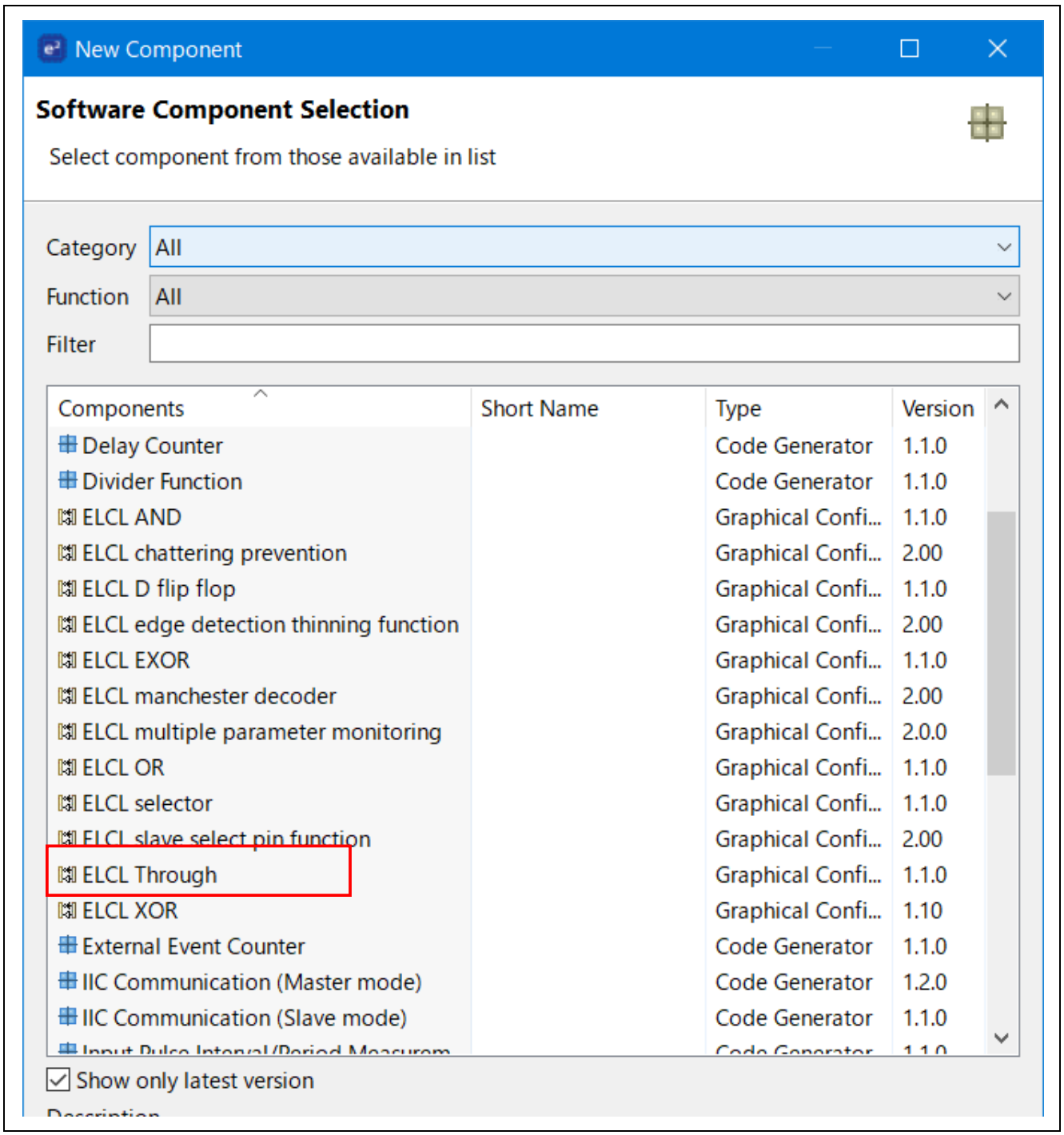
4. Select "Through" and download it. Please download the common setting file "RL78/G23 Common ELCL Module" as well.

**Figure 4-32 Download the module**



5. After the download is complete, make sure that "ELCL Through" is available for selection.

**Figure 4-33 Select the module**



### 4.2.2 r01an6420_elcl_i2s_master.scfg

This is the Smart Configurator configuration file used in the sample code. It contains all the features configured in the Smart Configurator. The sample code settings are as follows.

**Table 4-8 Parameters of Smart Configurator (1/2)**

| Tag name | Component | Contents |
|---|---|---|
| Clocks | - | Operation mod: High-speed main mode 2.4 (V) ~ 5.5 (V)<br>$EV_{DD}$ setting: 1.8V≦$EV_{DD0}$＜5.5V<br>High-speed on-chip oscillator: 32MHz<br>$f_{IHP}$: 32MHz<br>$f_{CLK}$: 32000kHz (High-speed on-chip oscillator)<br>$f_{SXP}$: 32.768kHz (Low-speed on-chip oscillator) |
| System | - | On-chip debug operation setting: COM port Note 1<br>Pseudo-RRM/DMM function setting: Used<br>Start/Stop function setting: Unused<br>Trace function setting: Used<br>Security ID setting: Use security ID<br>Security ID: 0x00000000000000000000<br>Security ID authentication failure setting: Do not erase flash memory data |
| Components | r_bsp | Start up select: Enable (use BSP startup)<br>Control of invalid memory access detection: Disable<br>RAM guard space (GRAM0-1): Disabled<br>Guard of control registers of port function (GPORT): Disabled<br>Guard of registers of interrupt function (GINT): Disabled<br>Guard of control registers of clock control function, voltage detector, and RAM parity error detection function (GCSC): Disabled<br>Data flash access control (DFLEN): Disables<br>Initialization of peripheral functions by Code Generator/Smart Configurator: Enable<br>API functions disable: Enable<br>Parameter check enable: Enable<br>Setting for starting the high-speed on-chip oscillator at the times of release from STOP mode and of transitions to SNOOZE mode: High-speed<br>Enable user warm start callback (PRE): Unused<br>Enable user warm start callback (POST): Unused<br>Watchdog Timer refresh enable: Unused |
| | Config_LVD0 | Operation mode setting: Reset mode<br>Voltage detection setting: Reset generation level ($V_{LVD0}$): 1.86 (V) |

**Table 4-9 Parameters of Smart Configurator (2/2)**

| Tag name | Component | Contents |
|---|---|---|
| Components | Config_TAU0_0 | Components: Interval timer<br>Operating mode: 16-bit count mode<br>Resource: TAU0_0<br>Operation clock: CK00<br>Clock source: $f_{CLK}$<br>Interval value: 0.163us<br>Generate INTTM00 interrupt at start of counting<br>Interrupt setting: Unuse |
| | Config_TAU0_1 | Components: External Event Counter<br>Resource: TAU0_1<br>Operation clock: CK00<br>Clock source: $f_{CLK}$<br>Input source setting: ELCL<br>Operation mode setting: 16-bit<br>External event edge select (TI01): Rising edge<br>Count value: 32<br>Interrupt setting: Unuse |
| | Config_TAU0_7 | Components: Interval timer<br>Operating mode: 16-bit count mode<br>Resource: TAU0_7<br>Operation clock: CK01<br>Clock source: $f_{CLK}/2^8$<br>Interval value: 1ms<br>Interrupt setting: use<br>Priority: Level 2 |
| | Config_Through | Components: ELCL Through<br>Common setting: L3L0<br><br>Detail setting: L3L0<br>Input signal selector: ELISEL_5 , TO00<br>Application: Through<br>Output signal selector: P51 |
| | Config_PORT | Components: Port<br>Port selection: PORT5<br>P53: Out (Output 1) |

#### 4.2.2.1 Clocks

Set the clock used in the sample code.

#### 4.2.2.2 System

Set the on-chip debug of the sample code.

"Control of on-chip debug operation" and "Security ID authentication failure setting" affect "On-chip debugging is enabled" in "Table 4-3 Option Byte Settings". Note that changing the settings.

#### 4.2.2.3 r_bsp

Set the startup of the sample code.

#### 4.2.2.4 Config_LVD0

Set the power management of the sample code.

Affects "Setting of LVD0" in "Table 4-3 Option Byte Settings". Note that changing the settings.

#### 4.2.2.5 Config_TAU00

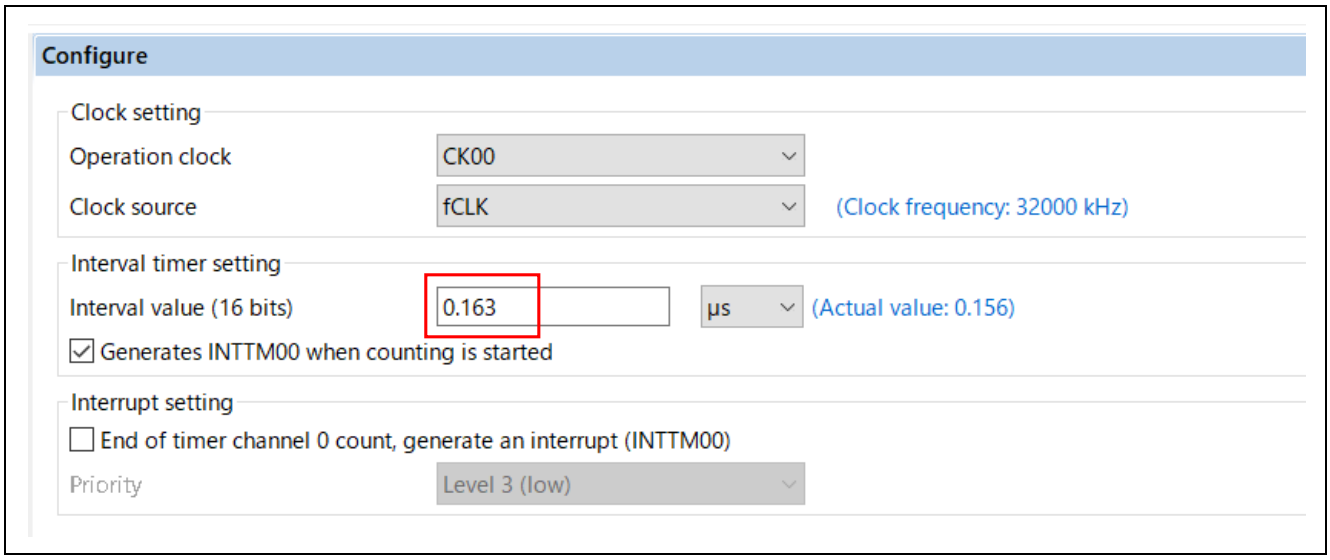
Set up TAU00 in the sample code.

The sample code uses it as an "interval timer" to generate BCLK. Set the values in Table 4-10 for the interval time in Figure 4-34 according to the sampling frequency and data size. Settings in red are those for which data is prepared in the sample code.

**Table 4-10 Set value of interval time**

| Data size | Sampling frequency（kHz） | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 8 | 11 | 12 | 16 | 22.1 | 24 | 32 | 44.1 | 48 | 96 |
| 16bits | 1.953 | 1.417 | 1.302 | 0.977 | 0.709 | 0.651 | 0.488 | 0.354 | 0.326 | 0.163 |
| 24bits | 1.302 | 0.945 | 0.868 | 0.651 | 0.472 | 0.434 | 0.326 | 0.236 | 0.217 | — |
| 32bits | 0.977 | 0.709 | 0.651 | 0.488 | 0.354 | 0.326 | 0.244 | 0.177 | 0.163 | — |

**Figure 4-34 TAU0 channel 0 setting (in the case of a sampling frequency of 48 kHz and a data size of 32 bits)**



#### 4.2.2.6 Config_TAU01

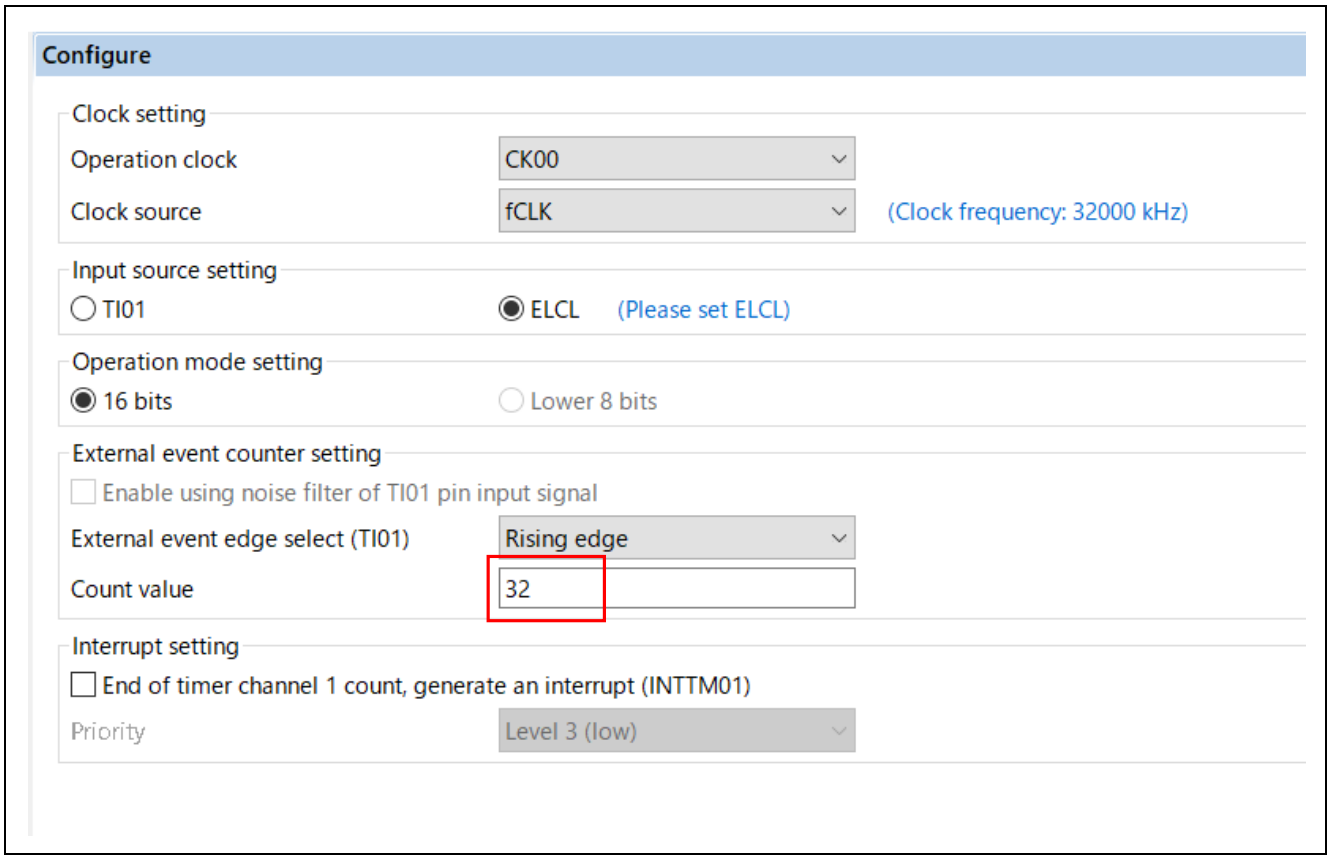
Set up TAU01 in the sample code.

The sample code uses it as an "external event count" and generates LRCLK. Set the value in Table 4-11 for the count value in Figure 4-35 to match the data size.

**Table 4-11 Count value**

| Data size | Count value |
|---|---|
| 16bits | 16 |
| 24bits | 24 |
| 32bits | 32 |

**Figure 4-35 TAU0 channel 1 setting (in case of 32 bits data size)**



#### 4.2.2.7 Config_TAU07

Set TAU07 in the sample code.

The sample code uses it as a 1 ms interval timer.

#### 4.2.2.8 Config_Through

Set to change the output terminal of BCLK in the sample code.

Set the output destination of TO00 used as BCLK to P51.

#### 4.2.2.9 Config_PORT

Set the port of the sample code.

In the sample code, P53 is used to control LED1.

### 4.2.3 How to change sound data

#### 4.2.3.1 To change the sound data in the sample code

This sample code is prepared with sound data for the following conditions.

Sampling frequency: 48kHz, Data size: 32bits

Sampling frequency: 44.1kHz, Data size: 24bits

Sampling frequency: 8kHz, Data size: 16bits

The sound data to be played can be changed in sound_data.h in the sample code.

The default settings are as follows: sampling frequency: 48 kHz, number of data: 32 bits of sound data.

```
#define DATA_48K_32B (1)
#define DATA_44K_24B (0)
#define DATA_8K_16B  (0)
```

The sound data can be changed by setting only the sound data to be played back to 1 and all others to 0.

Change TAU00 and TAU01 according to the set conditions, referring to Table 4-10 and Table 4-11.

The above settings can be used to change the sound data in the sample code.

#### 4.2.3.2 To add new sound data to the sample code

This sample code allows for the addition of new sound data.

(1) Please add new sound data to the following g_tx_buf[] in sound_data.c of the sample code.

```
#else    /* Not DATA_8K_16B_DATA */
/* For user setting */
const uint8_t g_tx_buf[] =
       {
              /* Add sound data here */
       };

#endif /* DATA_48K_32B */
```

(2) Set all sound data to be played in sound_data.h in the sample code to 0 as follows.

```
#define DATA_48K_32B (0)
#define DATA_44K_24B (0)
#define DATA_8K_16B  (0)
```

(3) Set the constant "DECODE_PCM_SIZE" for the number of data to be sent in the same file.

```
  #else /* For user setting */
  #define DECODE_PCM_SIZE   (0)

  #endif
```

(4) Please set the sampling frequency and the number of data in the following section of r_codec.c in the sample code.

```
#else /* For user setting */
uint8_t g_cmd_set_rate[2] =
       { 0x10, 0x00 }; /* Set sampling frequency */
uint8_t g_cmd_set_format[2] =
       { 0x0E, 0x0E }; /* Set input bit length */
#endif
```

**Table 4-12 Sampling frequency setting value**

| Sampling frequency | g_cmd_set_rate setting value |
|---|---|
| 96kHz | 0x10, 0x1C |
| 48kHz | 0x10, 0x00 |
| 44.1kHz | 0x10, 0x20 |
| 32kHz | 0x10, 0x18 |
| 8kHz | 0x10, 0x0C |

**Table 4-13 Setting value of the number of data**

| Number of data | g_cmd_set_format setting value |
|---|---|
| 16bits | 0x0E, 0x02 |
| 24bits | 0x0E, 0x0A |
| 32bits | 0x0E, 0x0E |

(5) Change TAU00 and TAU01 according to the set conditions, referring to Table 4-10 and Table 4-11.

The above settings allow you to add new sound data to the sample code.

### 4.2.4 When using other CODEC modules

When using a CODEC module other than RS Audio CODEC 754-1974, delete the files necessary for setting up the RS Audio CODEC 754-1974 module in Table 4-1 and Table 4-2. Also, delete the CODEC module setup process and CODEC module start process in the main function and add various settings according to the CODEC module to be used.

# 5. I²S communication (slave) using ELCL

## 5.1 Software

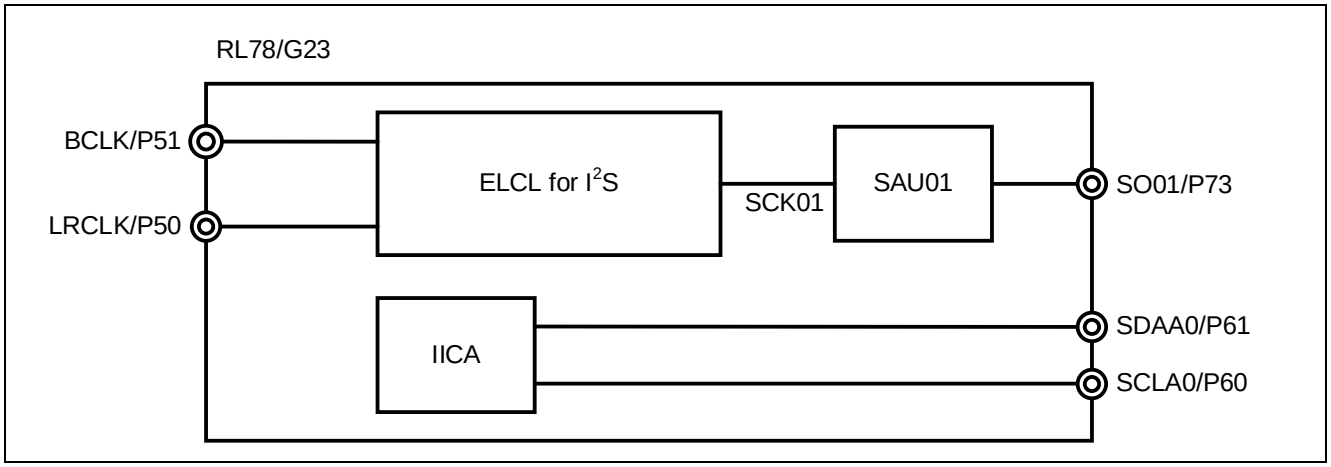
### 5.1.1 Overview of the sample program

This sample code uses RS Audio CODEC 754-1974. It also uses I²C communication for register settings in the audio CODEC and I²S communication for sound data transmission processing.

(1) IICA for I²C communication starts operation and sets registers in the audio CODEC.
(2) CSI01 for I²S communication starts operation (waiting for SCK01).
(3) BCLK and LRCLK are input from the audio CODEC and data transfer starts.
(4) After data transmission, LED1 is turned on.
(5) CSI01 and TAU0 stop.

Figure 5-1 shows the system configuration of the sample code.

**Figure 5-1 System configuration of the sample code**



### 5.1.2 Folder Configuration

Table 5-1 shows folder configuration of source file and header files using by sample code except the files generated by integrated development environment and the files in the bsp environment.

**Table 5-1 Folder configuration**

| Folder/File configuration | Outline | Created by Smart configurator |
|---|---|---|
| ¥r01an6420_elcl_slave<DIR>Note 3 | Root folder of this sample code | |
| ¥src<DIR> | Folder for program source | |
| main.c | Sample code source file | |
| main.h | Sample code header file | |
| r_codec.cNote 4 | Source file for CODEC module settings | |
| r_codec.hNote 4 | Header file for CODEC module settings | |
| sound_data.c | Source file for sound data | |
| sound_data.h | Header file for sound data | |
| ¥csi01<DIR> | Folder for CSI01 program | |
| csi01.c | Source file for CSI01 | |
| csi01.h | Header file for CSI01 | |
| ¥elcl<DIR> | Folder for ELCL program | |
| elcl.c | Source file for ELCL | |
| elcl.h | Header file for ELCL | |
| ¥iica0<DIR>Note 4 | Folder for IICA0 program | |
| iica0.c | Source file for IICA0 | |
| iica0.h | Header file for IICA0 | |
| ¥smc_gen<DIR> | Folder created by Smart Configurator | √ |
| ¥Config_PORT<DIR> | Folder for PORT program | √ |
| Config_PORT.c | Source file for PORT | √ |
| Config_PORT.h | Header file for PORT | √ |
| Config_PORT_user.c | Interrupt source file for PORT | √Note 1 |
| ¥Config_TAU0_7<DIR>Note 4 | Folder for TAU07 program | √ |
| Config_TAU0_7.c | Source file for TAU07 | √ |
| Config_TAU0_7.h | Header file for TAU07 | √ |
| Config_TAU0_7_user.c | Interrupt source file for TAU07 | √Note 2 |
| ¥general<DIR> | Folder for initialize or common program | √ |
| ¥r_bsp<DIR> | Folder for BSP program | √ |
| ¥r_config<DIR> | Folder for program | √ |

Note. <DIR> means directory

Note 1. Not used in this sample code.

Note 2. Added the interrupt handling routine to the file generated by the Smart Configurator.

Note 3. The IAR version of the sample code contains r01an6420_elcl_i2s_slave.ipcf. For the ipcf file, refer to "RL78 Smart Configurator User Guide: IAR (R20AN0581)".

Note 4. This file is required to configure the RS Audio CODEC 754-1974 module.

Since I²S communication uses CSI, it does not follow the original CSI standard and overrun errors occur. Error detection processing is removed from the code generated by the Smart configurator.

In addition, to enable multiple interrupts in IICA, the code generated by the Smart configurator is modified to remove unnecessary functions.

### 5.1.3 Option Byte Settings

Table 5-2 shows the option byte settings.

**Table 5-2 Option Byte Settings**

| Address | Setting Value | Contents |
|---|---|---|
| 000C0H/040C0H | 1110 1111B (EFH) | Operation of Watchdog timer is stopped (counting is stopped after reset) |
| 000C1H/040C1H | 1111 1110B (FEH) | LVD0 operating mode: reset mode<br>Detection voltage: Rising edge 1.90V<br>Falling edge 1.86V |
| 000C2H/040C2H | 1110 1000B (E8H) | Flash operating mode: HS mode<br>High-speed on-chip oscillator clock: 32MHz |
| 000C3H/040C3H | 1000 0101B (85H) | On-chip debugging is enabled |

### 5.1.4 Constants

Table 5-3 shows the constants that are used in this sample code.

**Table 5-3 Constants used in the sample code**

| Constant Name | Setting Value | Contents | File |
|---|---|---|---|
| LED1 | P5_bit.no3 | P53 | csi01.c |
| LED_ON | 0 | Setting value for turning on the LED | csi01.c |
| LED_OFF | 1 | Setting value for turning off the LED | csi01.c |
| DATA_48K_32B | 1 | Select the sound data to play.<br>(1: Enable, 0: Disable) | sound_data.h |
| DECODE_PCM_SIZE | 32000 | The number of sound data.<br>The setting value changes depending on the sound data to be played. | sound_data.h |
| g_tx_buf[] | See sample code | Sound data | sound_data.c |
| SENSOR_ADD | 0x34 | Sensor address | r_codec.h |
| WAIT_TIME | 100 | IICA0 communication wait time | r_codec.h |
| g_cmd_l_vol[2] | { 0x00, 0x17 } | Left line input channel volume control command | r_codec.c |
| g_cmd_r_vol[2] | { 0x02, 0x17 } | Right line input channel volume control command | r_codec.c |
| g_cmd_bypass[2] | { 0x08, 0x12 } | Analog audio path control command | r_codec.c |
| g_cmd_deempha[2] | { 0x0A, 0x00 } | Digital audio path control command | r_codec.c |
| g_cmd_line_adc[2] | { 0x0C, 0x42 } | Power down control command | r_codec.c |
| g_cmd_set_rate[2] | {0x10, 0x00} | Sampling frequency control command | r_codec.c |
| g_cmd_set_format[2] | {0x0E, 0x0E} | Digital audio interface format command | r_codec.c |
| g_cmd_activate[2] | { 0x12, 0x01 } | Module active command | r_codec.c |
| g_cmd_inactivate[2] | { 0x12, 0x00 } | Module inactive command | r_codec.c |
| g_cmd_reset[2] | { 0x1E, 0x00 } | Register reset command | r_codec.c |

### 5.1.5 Variables

Table 5-4 shows the global variables used in this sample code.

**Table 5-4 Global variables used in the sample code**

| Type | Variable name | contents | Functions used in |
|---|---|---|---|
| volatile uint16_t | g_ms_timer | Count value of the wait process | r_ms_delay,<br>r_Config_TAU0_7_interrupt |
| volatile uint8_t | g_tx_done_flag | Transmission Completion Flag | main<br>r_csi01_callback_sendend |
| volatile uint8_t | g_sample_mode | I²C communication status | r_codec_init<br>r_codec_start<br>r_codec_stop<br>r_iica0_callback_master_sende nd |

### 5.1.6 Functions

Table 5-5 shows the functions used in the sample code. However, the unchanged functions generated by the Smart Configurator are excluded.

**Table 5-5 Functions**

| Function name | Outline | Source file |
|---|---|---|
| main | Main process | main.c |
| r_csi01_create | CSI01 initialize process | csi.c |
| r_csi01_start | CSI01 operation start process | csi.c |
| r_csi01_stop | CSI01 operation stop process | csi.c |
| r_csi01_send | CSI01 start sending process | csi.c |
| r_csi01_callback_sendend | CSI01 callback function for end of transmission process | csi.c |
| r_csi01_interrupt | CSI01 interrupt process | csi.c |
| r_elcl_create | ELCL initialize process | elcl.c |
| r_elcl_start | ELCL operation start process | elcl.c |
| r_elcl_stop | ELCL operation stop process | elcl.c |
| r_elcl_reset_flipflop | ELCL flip-flop reset process | elcl.c |
| r_iica0_create | IICA0 initialize process | Iica0.c |
| r_iica0_stop | IICA0 operation stop process | Iica0.c |
| r_iica0_stopcondition | IICA0 stop condition process | Iica0.c |
| r_iica0_master_send | IICA0 Send start process | Iica0.c |
| r_iica0_callback_master_sendend | IICA0 callback function for end of transmission process | Iica0.c |
| r_iica0_master_handler | IICA0 interrupt handler | Iica0.c |
| r_iica0_interrupt | IICA0 interrupt process | Iica0.c |
| r_codec_init | CODEC module setup process | r_codec.c |
| r_codec_start | CODEC module start process | r_codec.c |
| r_codec_stop | CODEC module stop process | r_codec.c |
| r_ms_delay | Wait process | Config_TAU0_7_user.c |
| r_Config_TAU0_7_interrupt | TAU0 channel 7 interrupt process | Config_TAU0_7_user.c |

### 5.1.7 Function Specifications

This part describes function specifications of the sample code.

[Function name] main

| | |
|---|---|
| **Outline** | Main process |
| **Header** | r_smc_entry.h, main.h, elcl.h, csi01.h, r_codec.h, sound_data.h |
| **Declaration** | void main (void); |
| **Description** | This function initializes ELCL and CSI01, sets CODEC module, and performs I²S communication. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_create

| | |
|---|---|
| **Outline** | CSI01 initialize process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | void r_csi01_create (void); |
| **Description** | This function initializes CSI01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_start

| | |
|---|---|
| **Outline** | CSI01 operation start process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | void r_csi01_start (void); |
| **Description** | This function enables the CSI0 |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_stop

| | |
|---|---|
| **Outline** | CSI01 operation stop process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | void r_csi01_stop (void); |
| **Description** | This function stops the CSI01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_send

| | |
|---|---|
| **Outline** | CSI01 start sending process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | MD_STATUS r_csi01_send (uint8_t *const tx_buf, uint16_t tx_num); |
| **Description** | This function sends the number of data specified by the argument tx_num from the address specified by the argument tx_buf.<br>This function does not perform error detection. |
| **Arguments** | tx_buf, tx_num |
| **Return value** | status |
| **Remarks** | None |

[Function name] r_csi01_callback_sendend

| | |
|---|---|
| **Outline** | CSI01 callback function for end of transmission process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | static void r_csi01_callback_sendend (void); |
| **Description** | This function stops the output of ELCL, sets g_tx_done_flag and turns on LED1. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_csi01_interrupt

| | |
|---|---|
| **Outline** | CSI01 interrupt process |
| **Header** | csi01.h, elcl.h |
| **Declaration** | #pragma interrupt r_csi01_interrupt (vect=INTCSI01) |
| **Description** | This function sets the next transmission data to SIO01. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_create

| | |
|---|---|
| **Outline** | ELCL initialize process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_create (void); |
| **Description** | This function initializes ELCL. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_start

| | |
|---|---|
| **Outline** | ELCL operation start process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_start (void); |
| **Description** | This function enables the ELCL output. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_stop

| | |
|---|---|
| **Outline** | ELCL operation stop process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_stop (void); |
| **Description** | This function stops the ELCL output. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_elcl_reset_flipflop

| | |
|---|---|
| **Outline** | ELCL flip-flop reset process |
| **Header** | elcl.h, Config_Throgh.h, platform.h |
| **Declaration** | void r_elcl_reset_flipflop (void); |
| **Description** | This function resets the flip-flops to reset the ELOMONI flag. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | Flip-flops are reset when bit 6 and bit 7 of ELLnCTL are set to 0. |

[Function name] r_iica0_create

| | |
|---|---|
| **Outline** | IICA0 initialize process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_csi01_create (void); |
| **Description** | This function initializes IICA0. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_stop

| | |
|---|---|
| **Outline** | IICA0 operation stop process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_iica0_stop (void); |
| **Description** | This function stops IICA0 operation. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_stopcondition

| | |
|---|---|
| **Outline** | IICA0 stop condition process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_iica0_stopcondition (void); |
| **Description** | This function generates stop conditions. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_master_send

| | |
|---|---|
| **Outline** | IICA0 Send start process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_iica0_master_send (uint8_t adr, uint8_t *const tx_buf, uint16_t tx_num, uint8_t wait); |
| **Description** | (1) When the I²C bus is released, the start condition is generated.<br>(2) After the start condition is generated, the slave address is set to transmit mode and data transmission is started.<br>(3) Waits for communication completion. |
| **Arguments** | adr, tx_buf, tx_num, wait |
| **Return value** | status |
| **Remarks** | None |

[Function name] r_iica0_callback_master_sendend

| | |
|---|---|
| **Outline** | IICA0 callback function for end of transmission process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | static void r_iica0_callback_master_sendend (void); |
| **Description** | This function generates stop condition and sets IIC communication status to FINISH. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_master_handler

| | |
|---|---|
| **Outline** | IICA0 interrupt handler |
| **Header** | iica.h, r_codec.h |
| **Declaration** | void r_iica0_master_handler (void); |
| **Description** | This function sends data if it detects an ACK for a slave address. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_iica0_interrupt

| | |
|---|---|
| **Outline** | IICA0 interrupt process |
| **Header** | iica.h, r_codec.h |
| **Declaration** | #pragma interrupt r_iica0_interrupt (vect=INTIICA0,enable=true) |
| **Description** | This function accepts and processes IICA0 interrupt requests.<br>Permits multiple interrupts. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_codec_init

| | |
|---|---|
| **Outline** | CODEC module setup process |
| **Header** | sound_data.h, iica0.h, r_codec.h, Config_TAU0_7.h |
| **Declaration** | void r_codec_init (void); |
| **Description** | (1) Performs the initial setup process for IICA0.<br>(2) Setup the CODEC module.<br>(3) Waits for setup completion (100ms). |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_codec_start

| | |
|---|---|
| **Outline** | CODEC module start process |
| **Header** | sound_data.h, iica0.h, r_codec.h, Config_TAU0_7.h |
| **Declaration** | void r_codec_start (void); |
| **Description** | This function starts the operation of the CODEC module. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_codec_stop

| | |
|---|---|
| **Outline** | CODEC module stop process |
| **Header** | sound_data.h, iica0.h, r_codec.h, Config_TAU0_7.h |
| **Declaration** | void r_codec_stop (void); |
| **Description** | This function stops the operation of the CODEC module. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_ms_delay

| | |
|---|---|
| **Outline** | Wait process |
| **Header** | r_cg_macrodriver.h, r_cg_userdefine.h, Config_TAU0_7.h |
| **Declaration** | void r_ms_delay (uint16_t msec); |
| **Description** | This function waits for the time (ms) specified by the argument msec.<br>This function counts using channel 7.Polls if g_ms_timer is less than msec, completes wait process if more than msec. |
| **Arguments** | msec |
| **Return value** | None |
| **Remarks** | None |

[Function name] r_Config_TAU0_7_interrupt

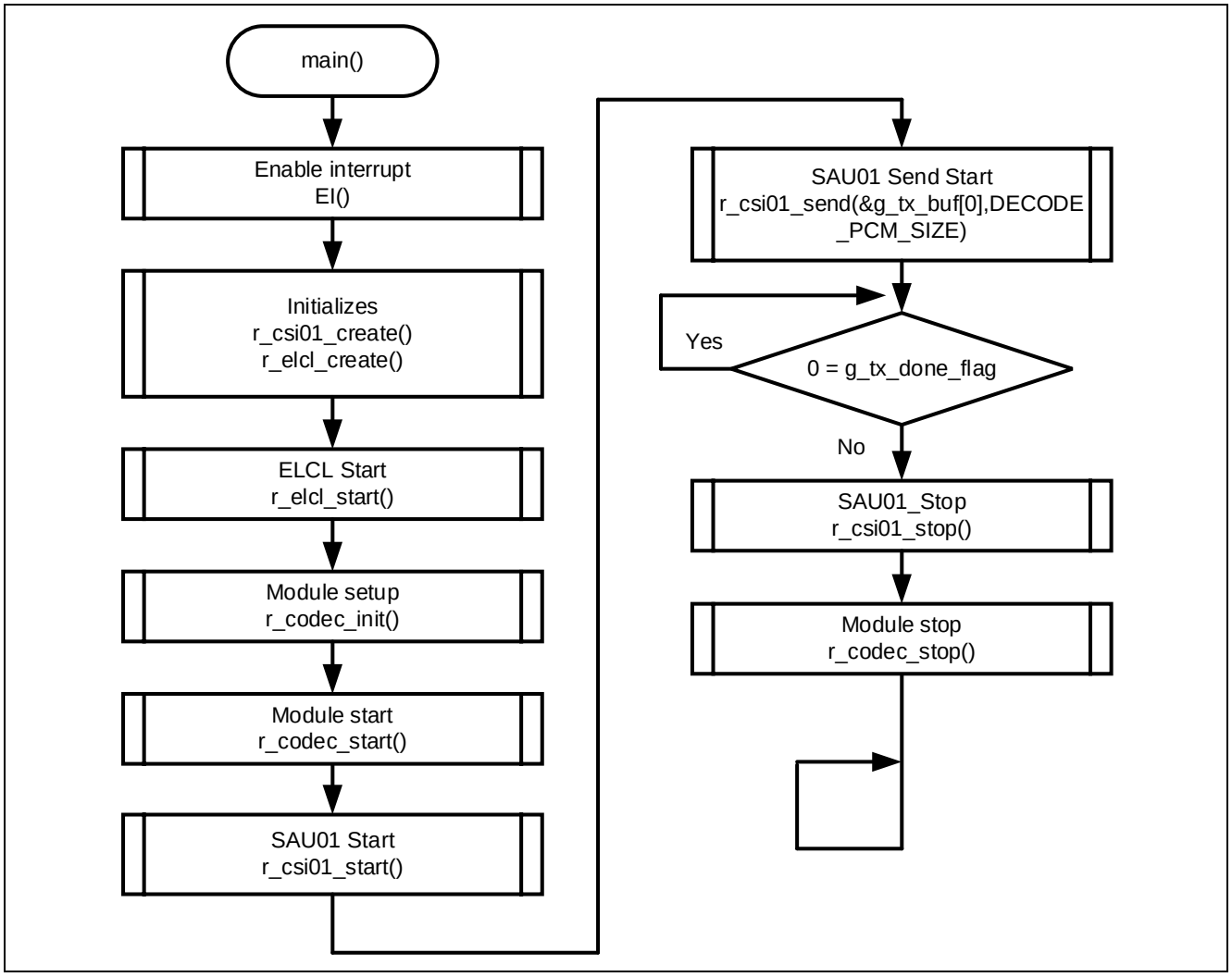
| | |
|---|---|
| **Outline** | TAU0 channel 7 interrupt process |
| **Header** | r_cg_macrodriver.h, r_cg_userdefine.h, Config_TAU0_7.h |
| **Declaration** | #pragma interrupt r_Config_TAU0_7_interrupt (vect=INTTM07) |
| **Description** | This function is an interrupt process by INTTM07 on TAU0 channel 7.<br>Counts up g_ms_timer. |
| **Arguments** | None |
| **Return value** | None |
| **Remarks** | None |

### 5.1.8 Flow Charts

#### 5.1.8.1 Main Process

Figure 5-2 shows flowchart of main process

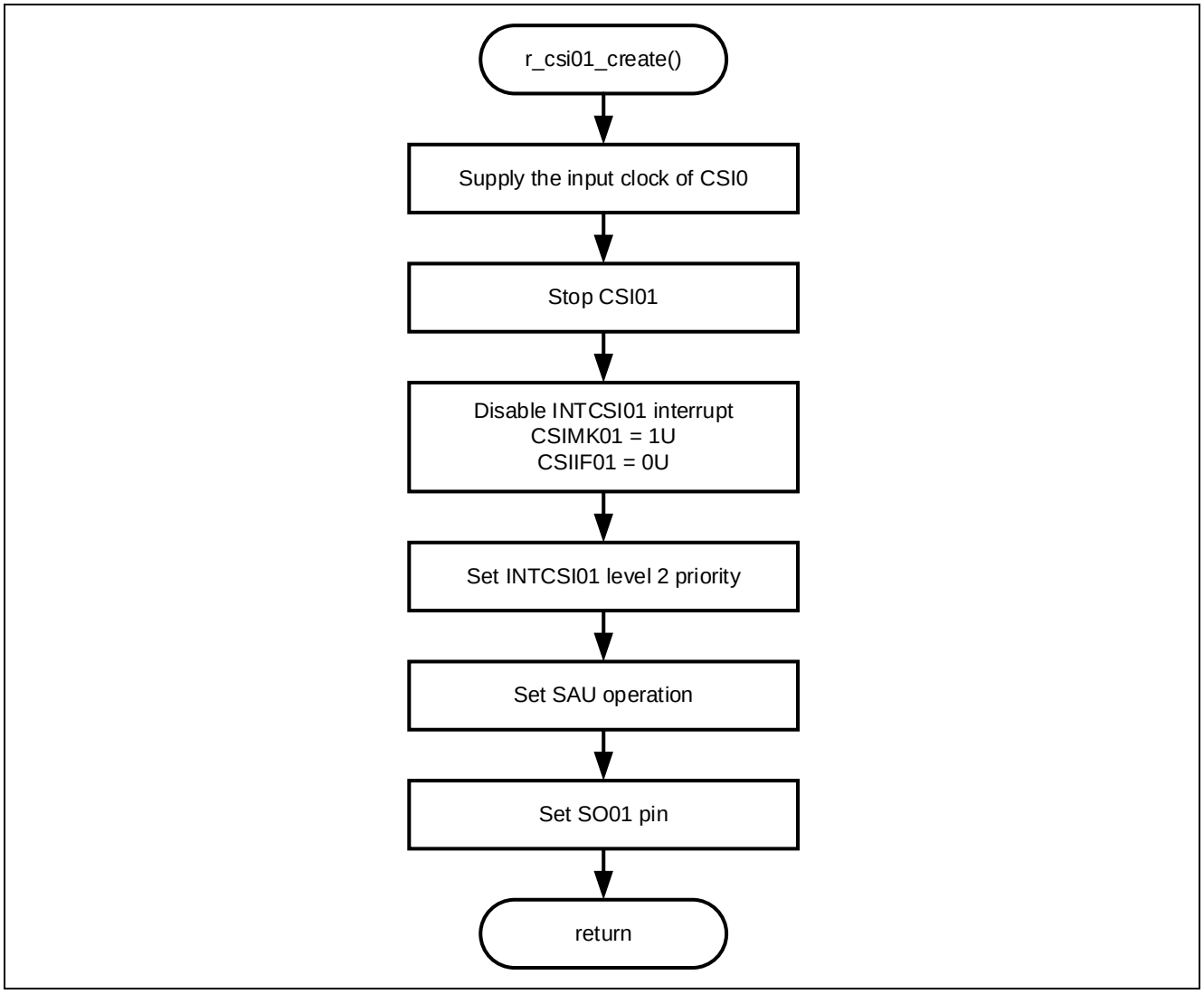
**Figure 5-2 Main process**



### 5.1.8.2 CSI01 initialize process

Figure 5-3 shows flowchart of CSI01 initialize process.

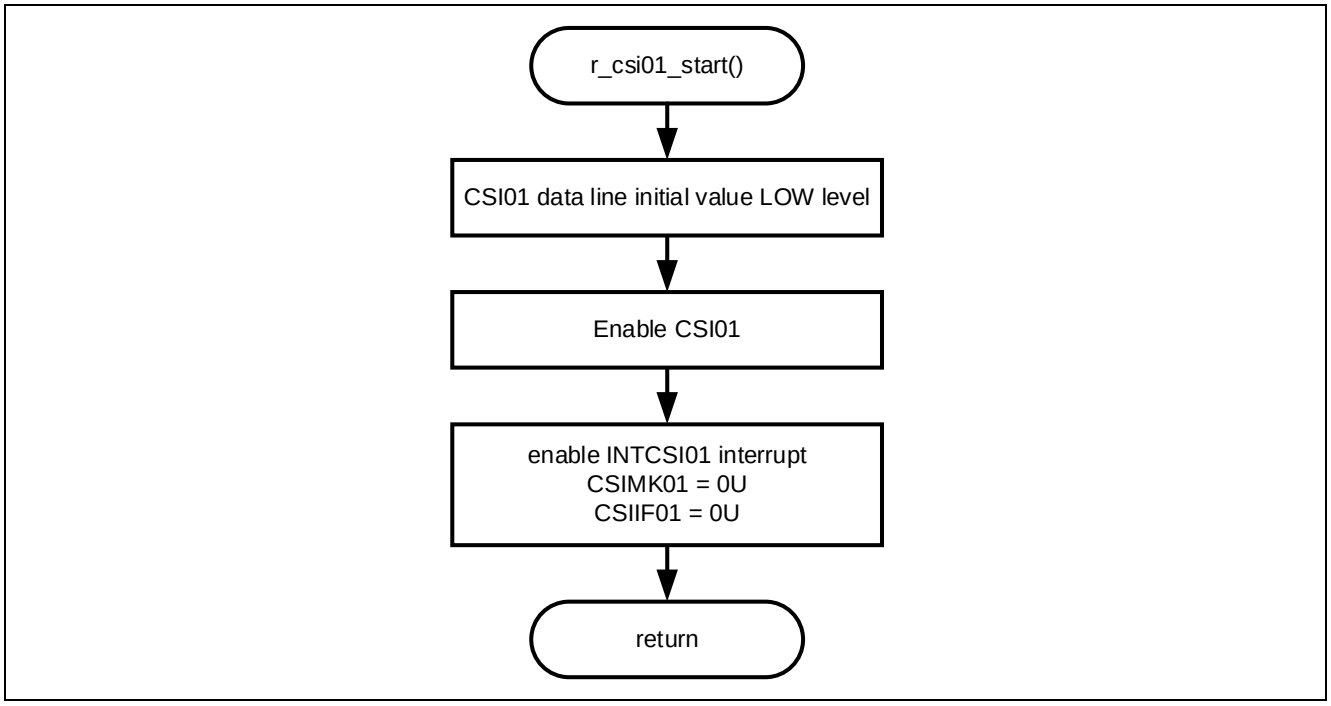
**Figure 5-3 CSI01 initialize process**



### 5.1.8.3 CSI01 operation start process

Figure 5-4 shows flowchart of CSI01 operation start process.

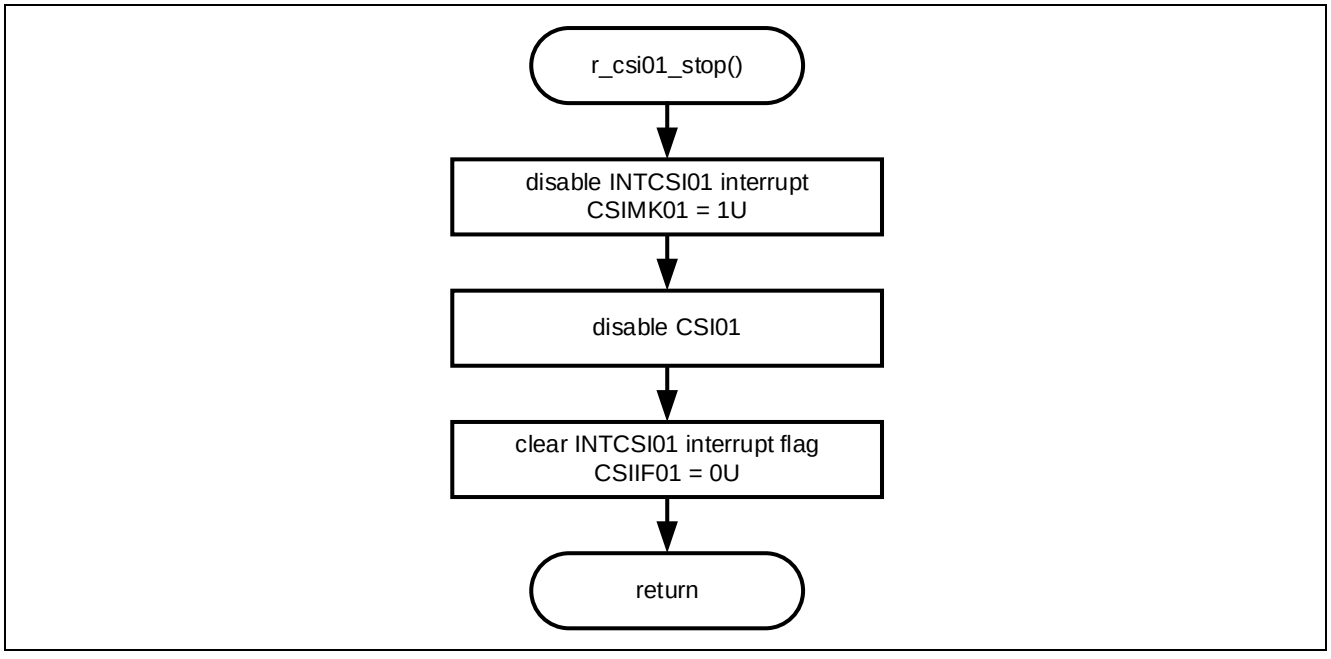
**Figure 5-4 CSI01 operation start process**




### 5.1.8.4 CSI01 operation stop process

Figure 5-5 shows flowchart of CSI01 operation stop process.

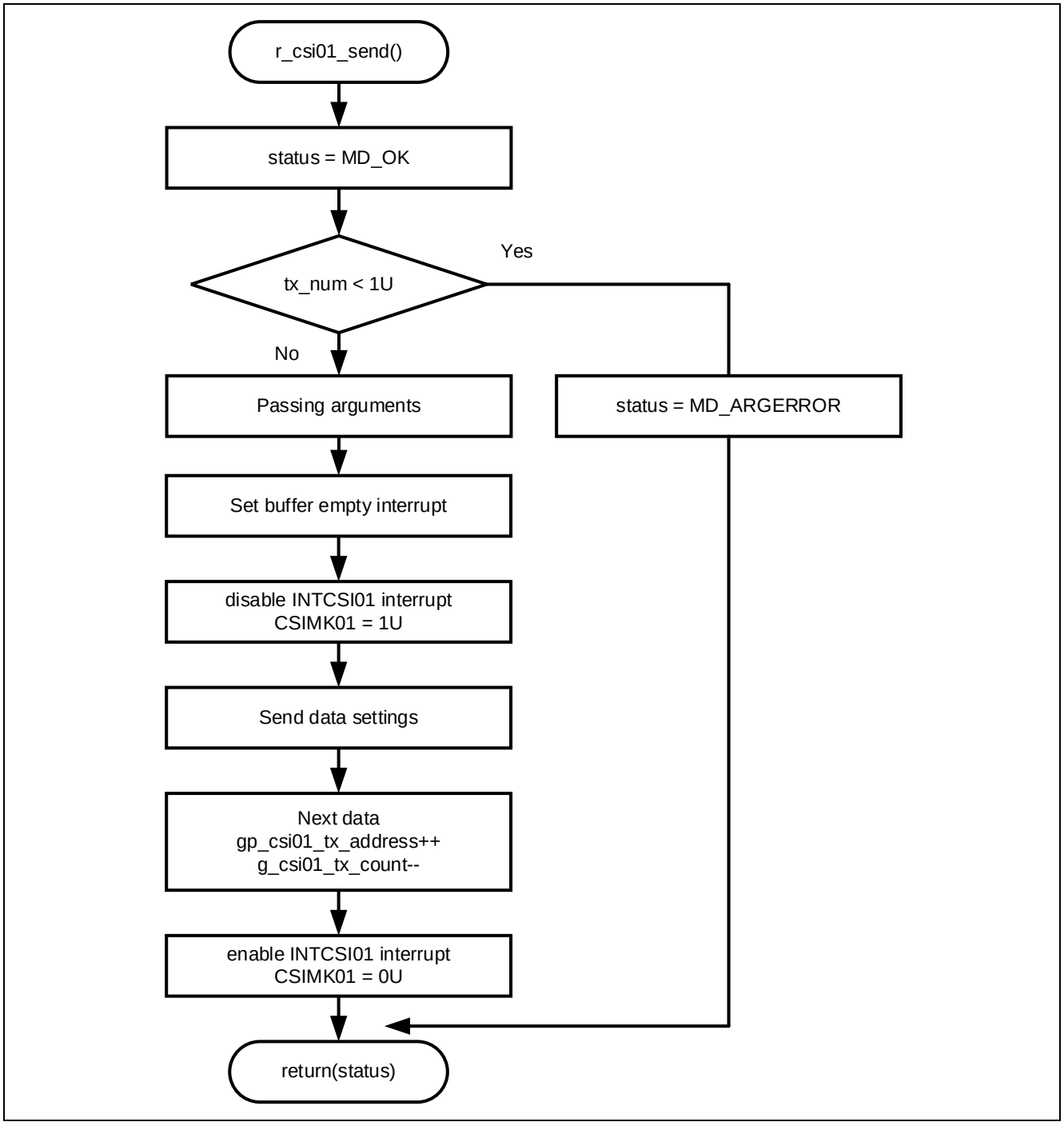
**Figure 5-5 CSI01 operation stop process**



### 5.1.8.5 CSI01 send start process

Figure 5-6 shows flowchart of CSI01 start sending process.

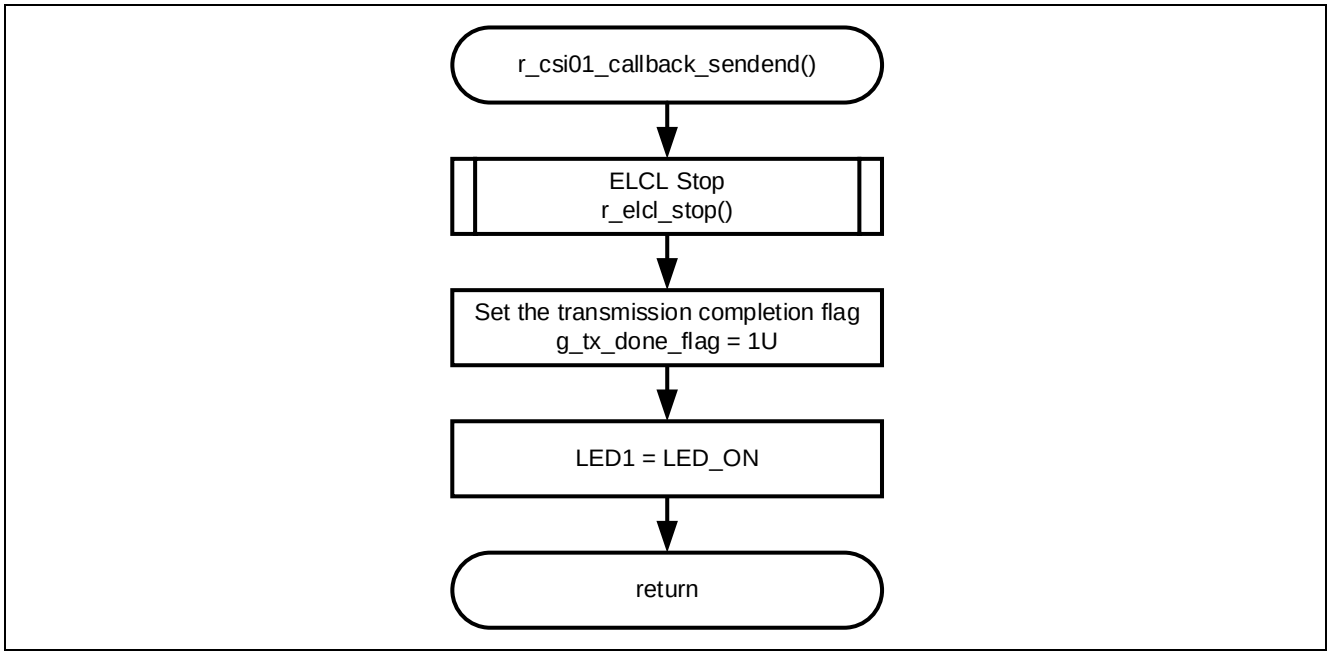
**Figure 5-6 CSI01 start sending process**



#### 5.1.8.6 CSI01 callback function for end of transmission process

Figure 5-7 shows flowchart of CSI01 callback function for end of transmission process.

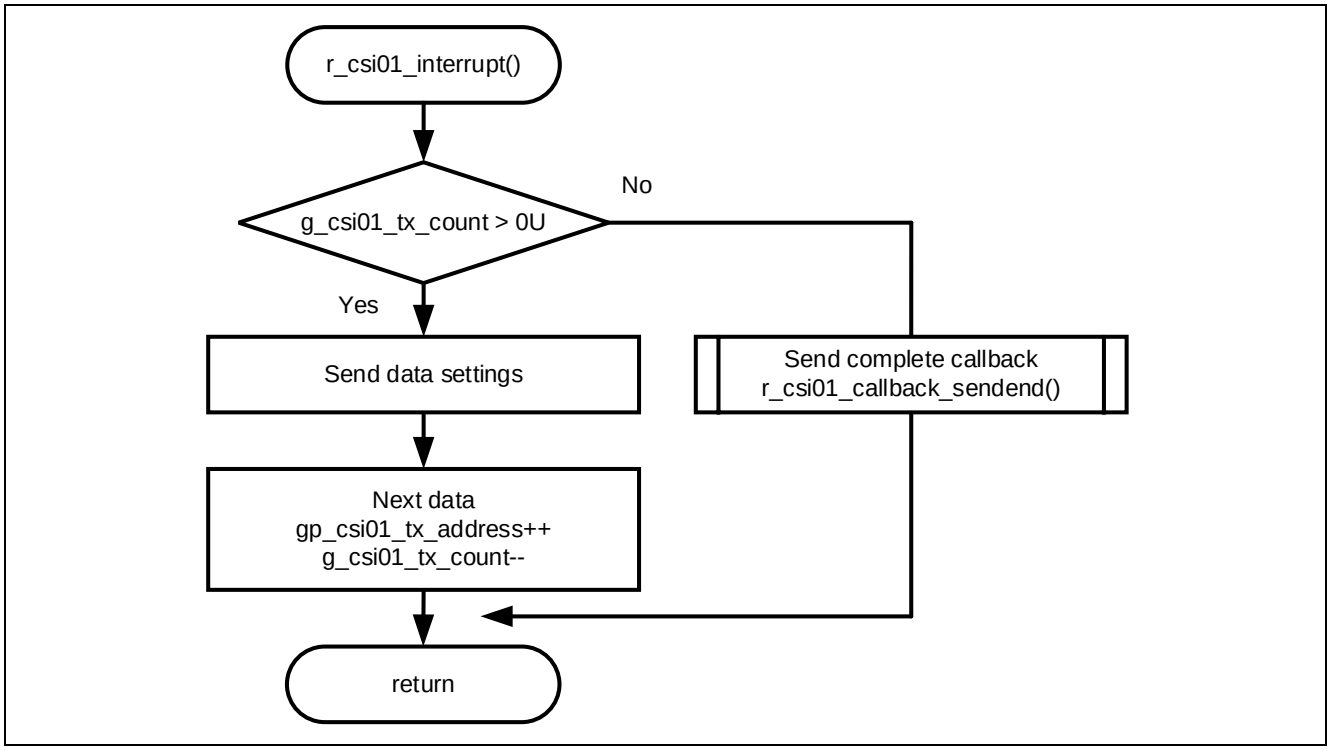
**Figure 5-7 CSI01 callback function for end of transmission process**



### 5.1.8.7 CSI01 interrupt process

Figure 5-8 shows flowchart of CSI01 interrupt process.

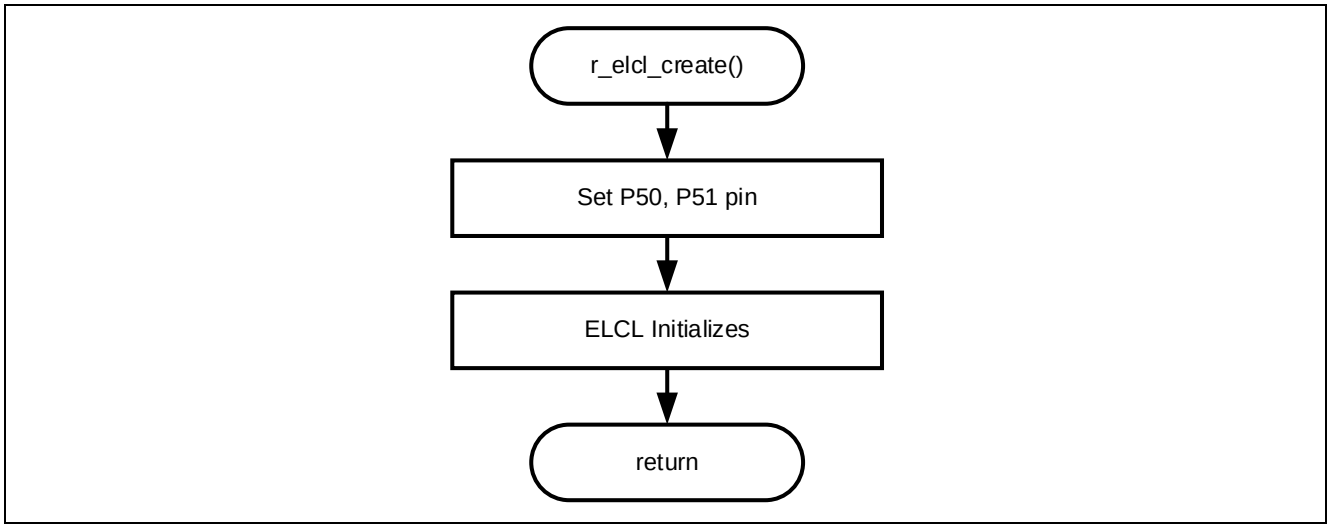
**Figure 5-8 CSI01 interrupt process**



#### 5.1.8.8 ELCL initialize process

Figure 5-9 shows flowchart of ELCL initialize process.

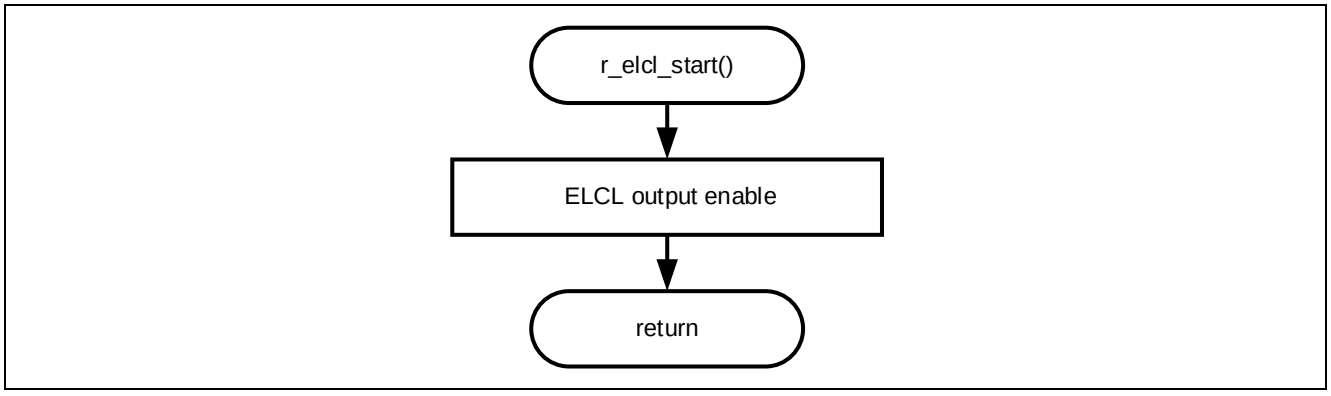
**Figure 5-9 ELCL initialize process**




#### 5.1.8.9 ELCL operation start process

Figure 5-10 shows flowchart of ELCL operation start process.

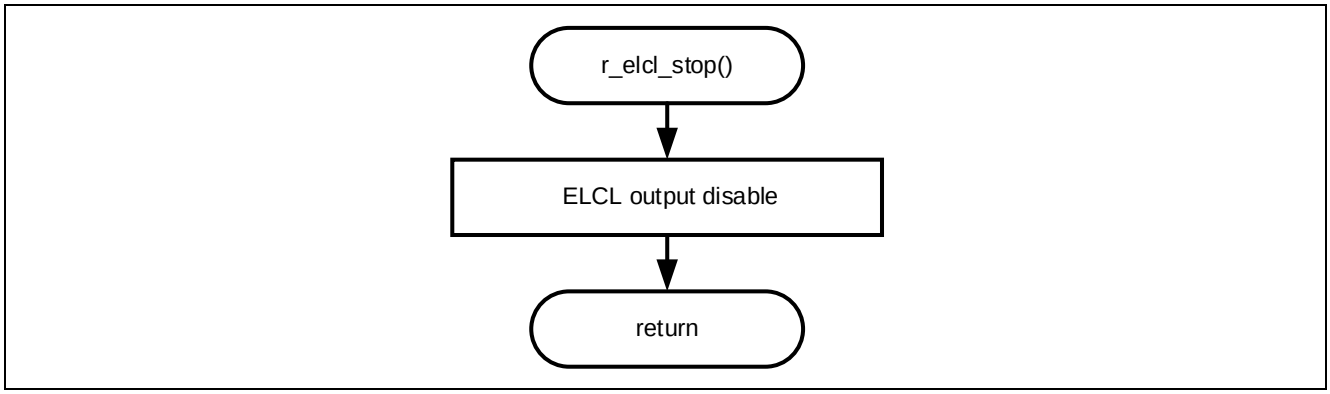
**Figure 5-10 ELCL operation start process**



### 5.1.8.10 ELCL operation stop process

Figure 5-11 shows flowchart of ELCL operation stop process.

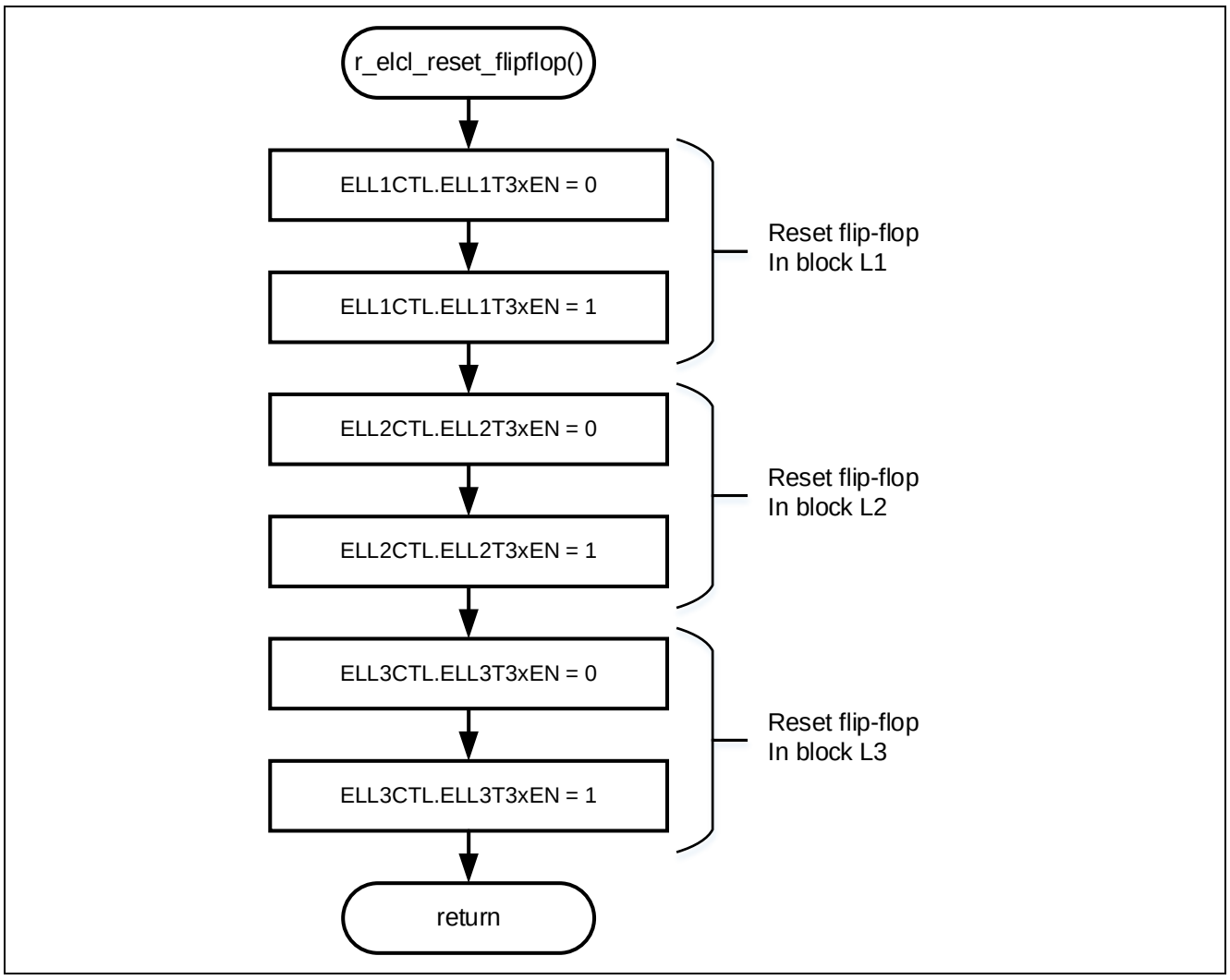
**Figure 5-11 ELCL operation stop process**



### 5.1.8.11 ELCL flip-flop reset process

Figure 5-12 shows flowchart of ELCL flip-flop reset process.

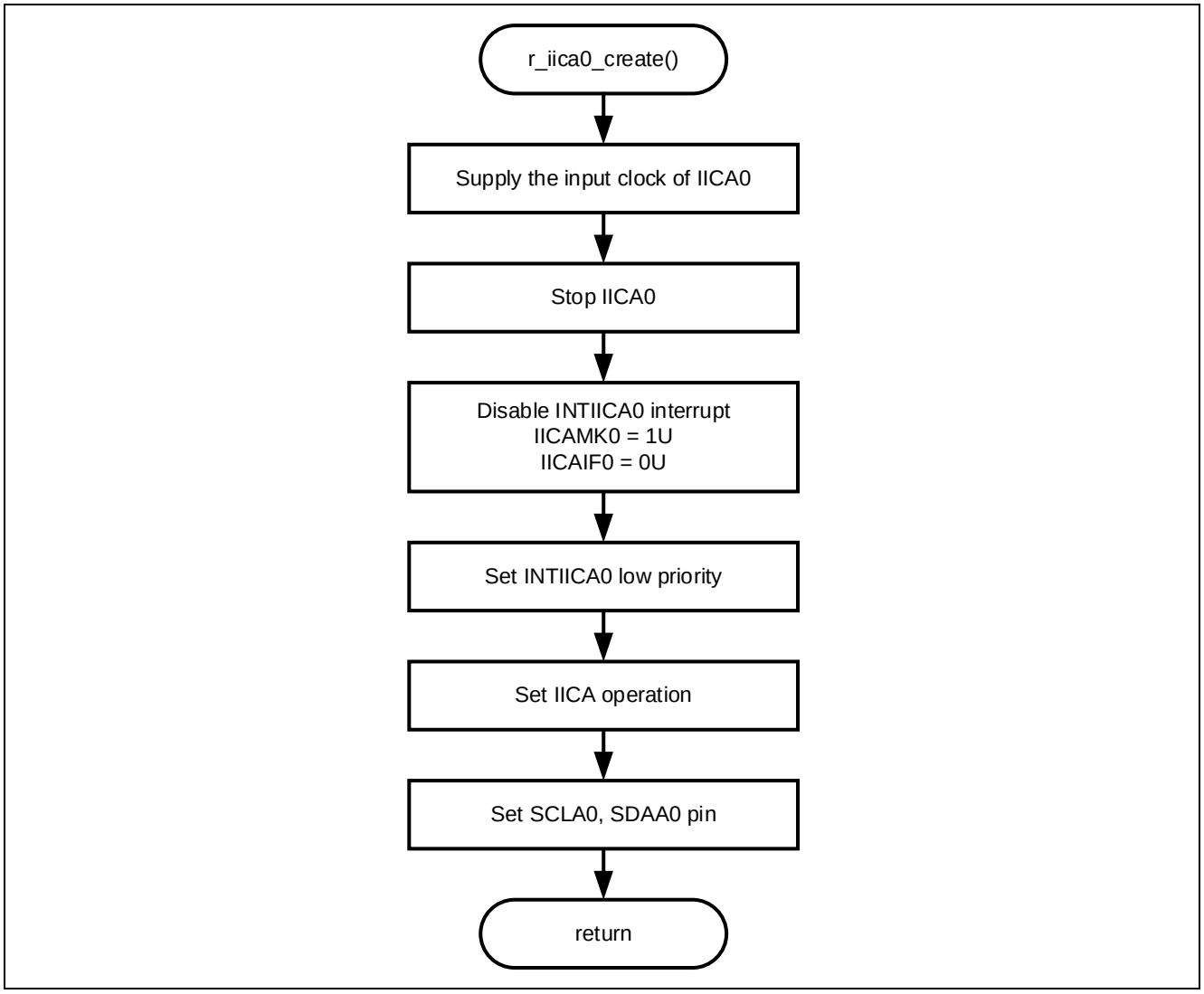
**Figure 5-12 ELCL flip-flop reset process**



### 5.1.8.12 IICA0 initialize process

Figure 5-13 shows flowchart of IICA0 initialize process.

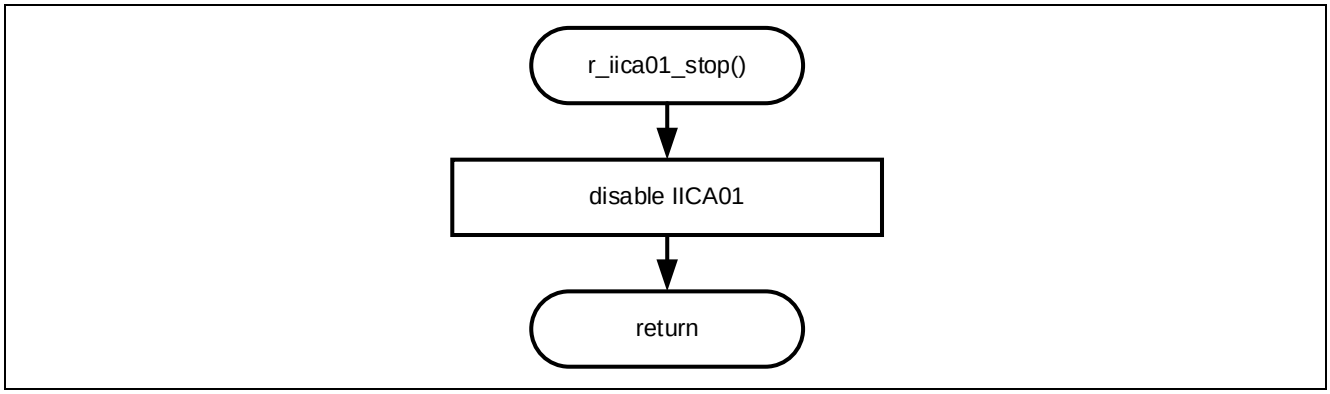
**Figure 5-13 IICA0 initialize process**



### 5.1.8.13 IICA0 operation stop process

Figure 5-14 shows flowchart of IICA0 operation stop process.

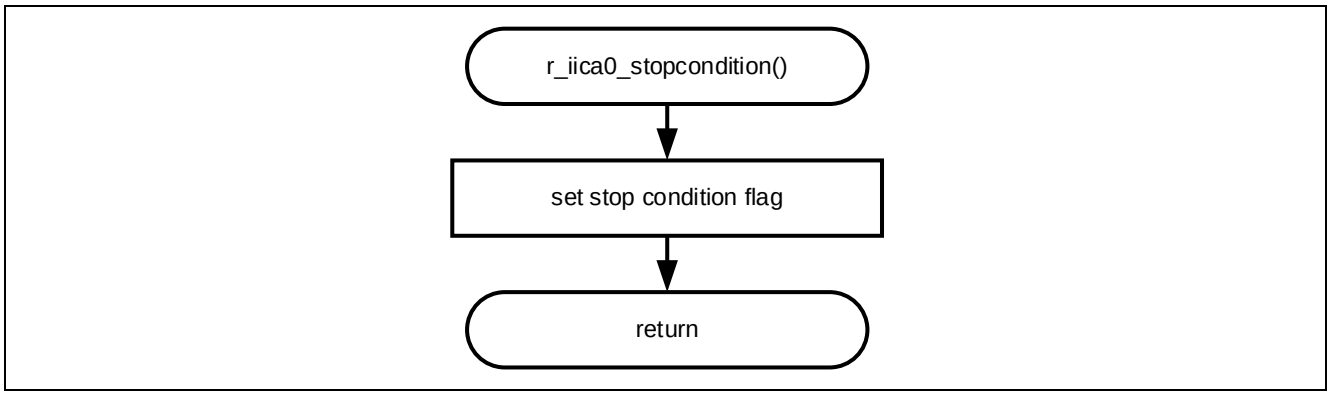
**Figure 5-14 IICA0 operation stop process**




### 5.1.8.14 IICA0 stop condition process

Figure 5-15 shows flowchart of IICA0 stop condition process.

**Figure 5-15 IICA0 stop condition process**



### 5.1.8.15 IICA0 Send start process

Figure 5-16, Figure 5-17 shows flowchart of IICA0 Send start process.

**Figure 5-16 IICA0 Send start process (1/2)**

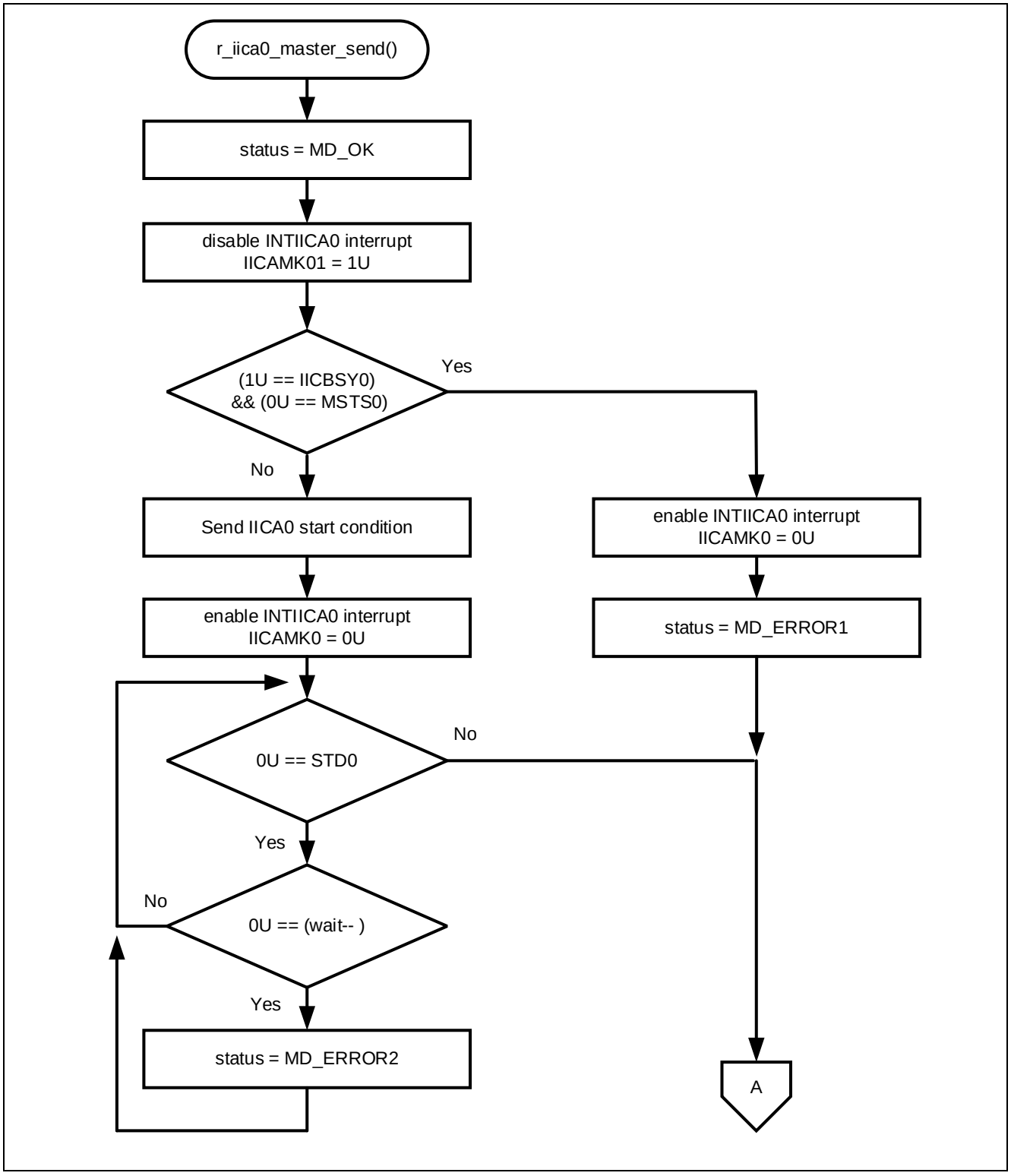
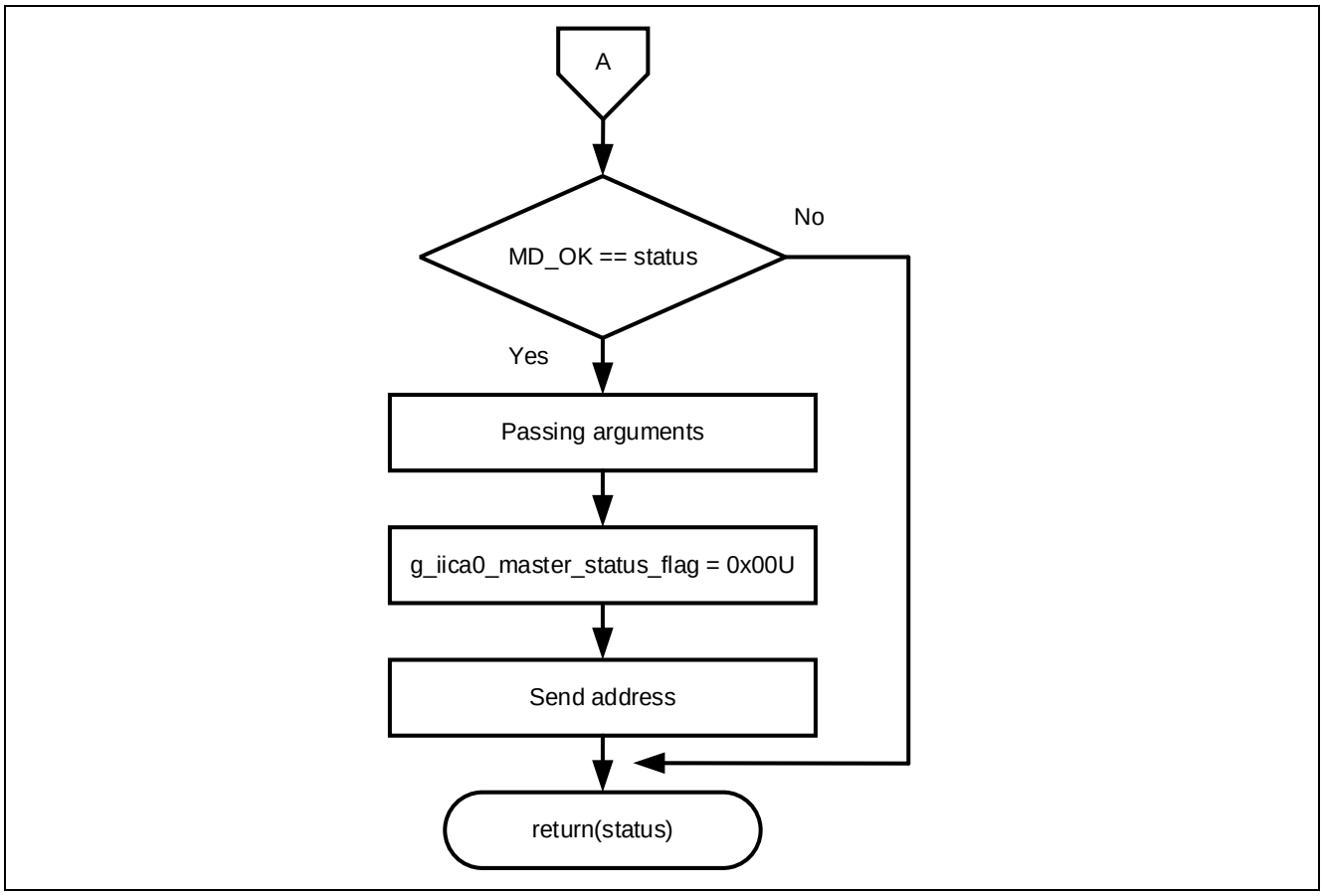


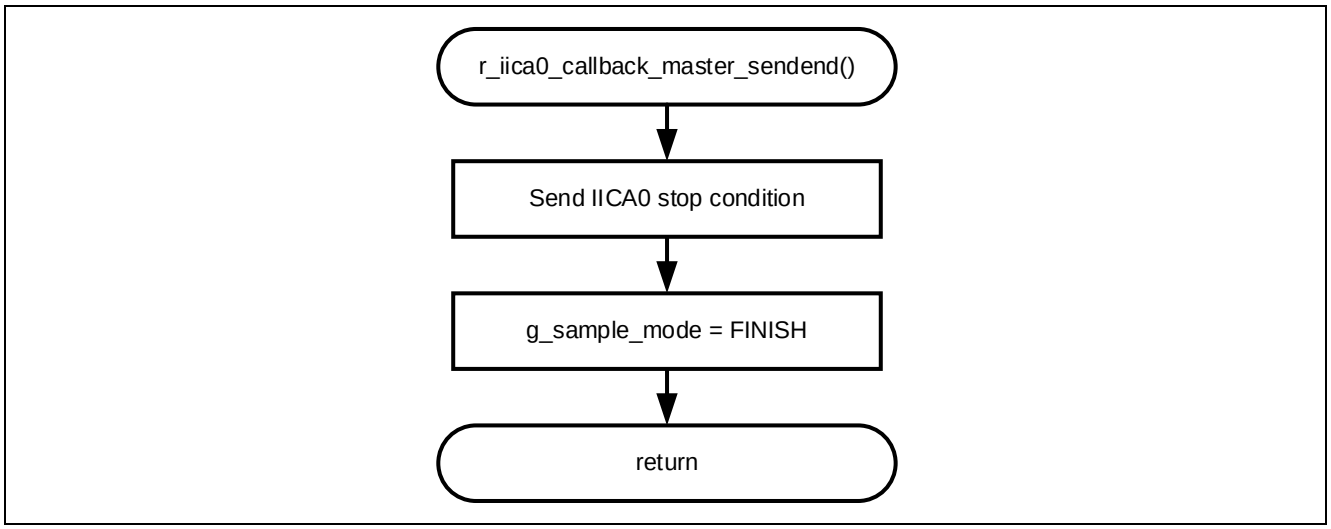
**Figure 5-17 IICA0 send start process (2/2)**




### 5.1.8.16 IICA0 callback function for end of transmission process

Figure 5-18 shows flowchart of IICA0 callback function for end of transmission process.

**Figure 5-18 IICA0 callback function for end of transmission process**



### 5.1.8.17 IICA0 interrupt handler

Figure 5-19, Figure 5-20 shows flowchart of IICA0 interrupt handler.

**Figure 5-19 IICA0 interrupt handler (1/2)**

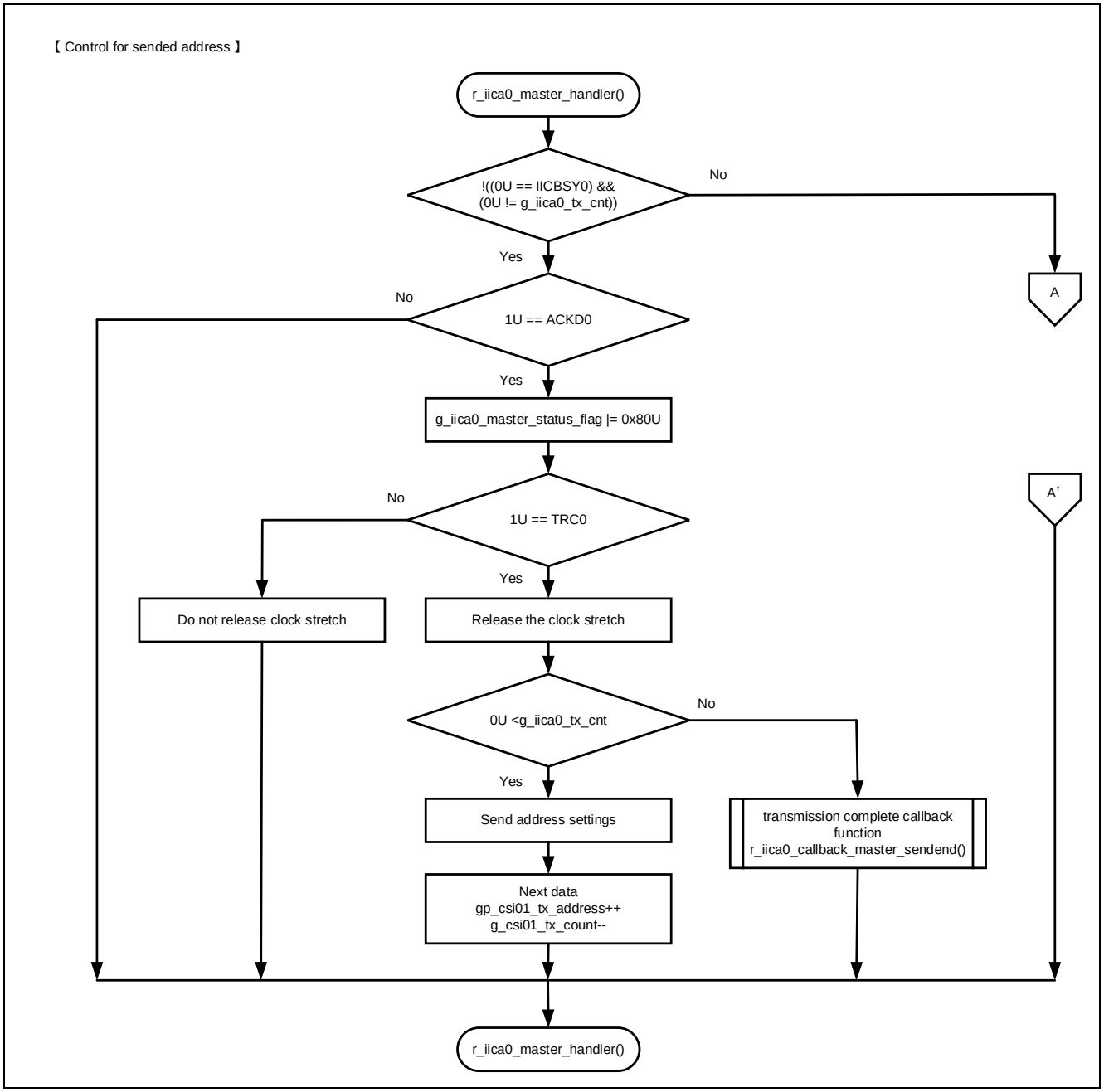
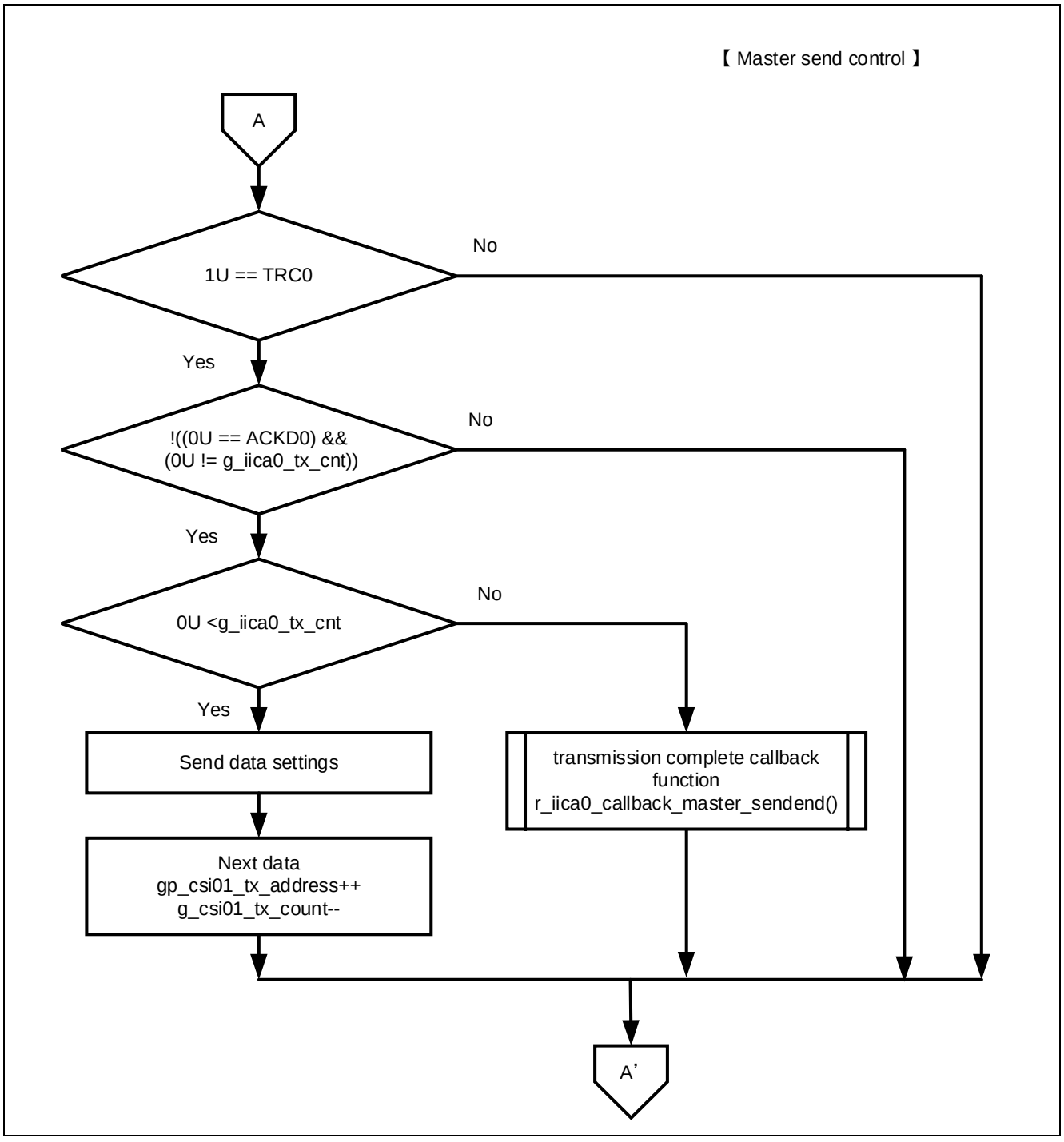


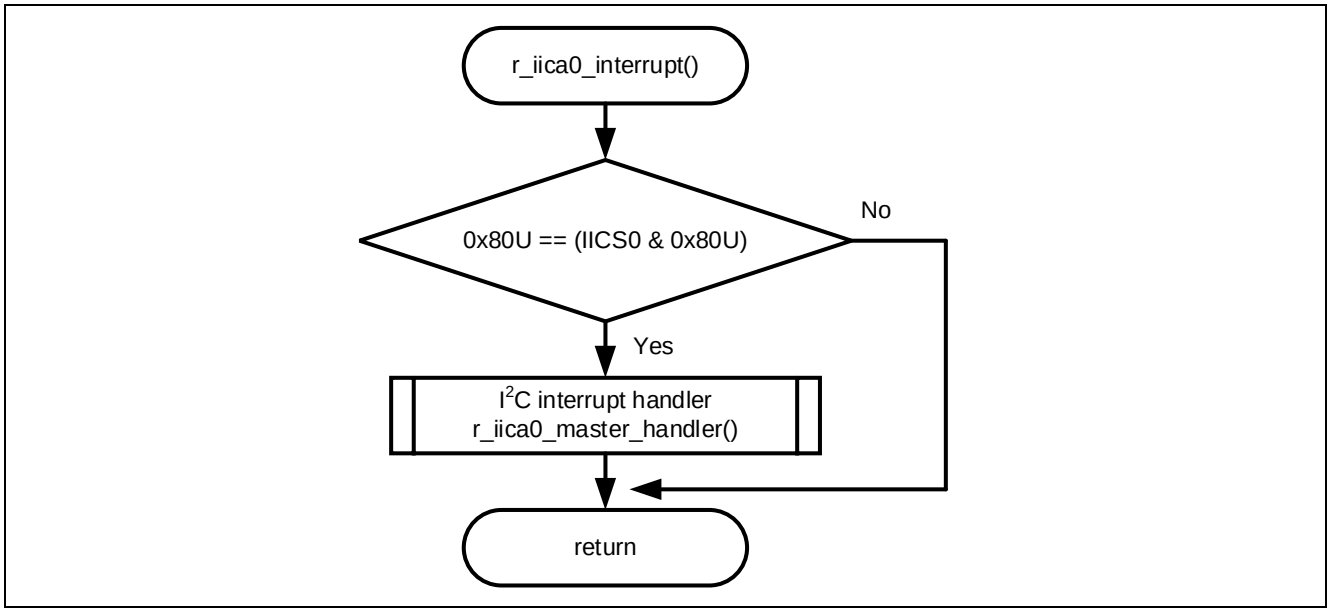
**Figure 5-20 IICA0 interrupt handler (2/2)**



### 5.1.8.18 IICA0 interrupt process

Figure 5-21 shows flowchart of IICA0 interrupt process.

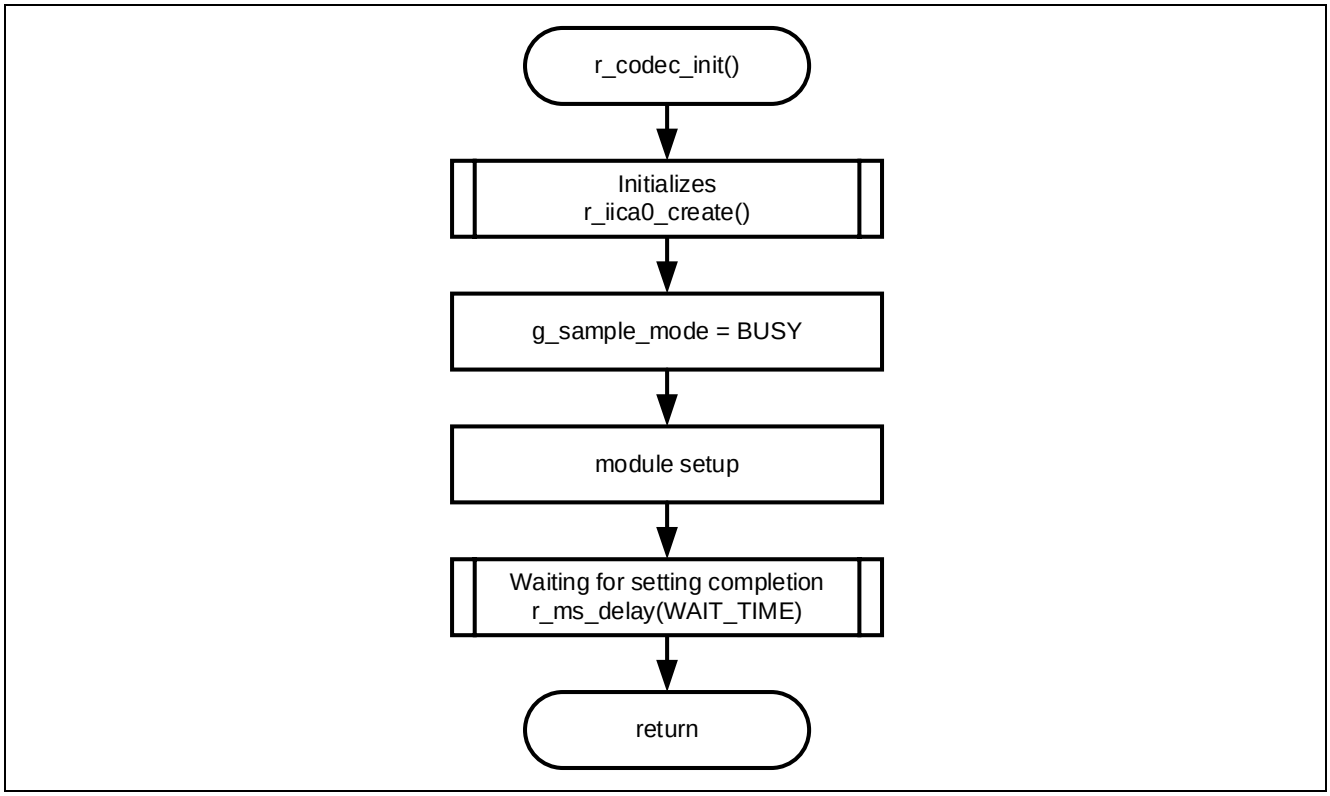
**Figure 5-21 IICA0 interrupt process**



### 5.1.8.19 CODEC module setup process

Figure 5-22 shows flowchart of CODEC module setup process.

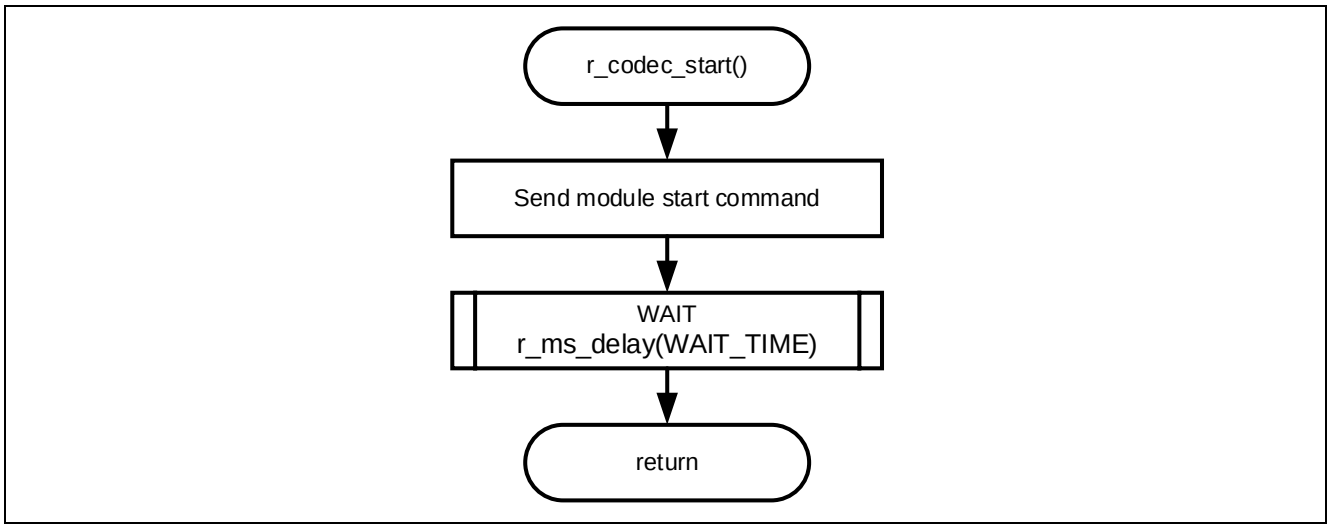
**Figure 5-22 CODEC module setup process**




### 5.1.8.20 CODEC module start process

Figure 5-23 shows flowchart of CODEC module start process.

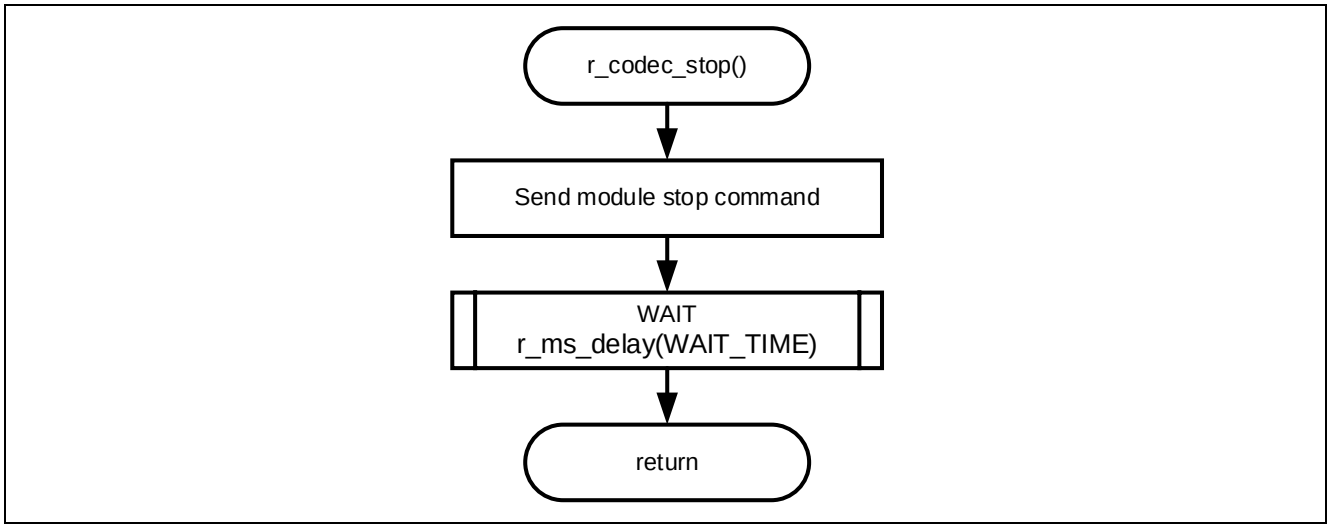
**Figure 5-23 CODEC module start process**



### 5.1.8.21 CODEC module stop process

Figure 5-24 shows flowchart of CODEC module stop process.

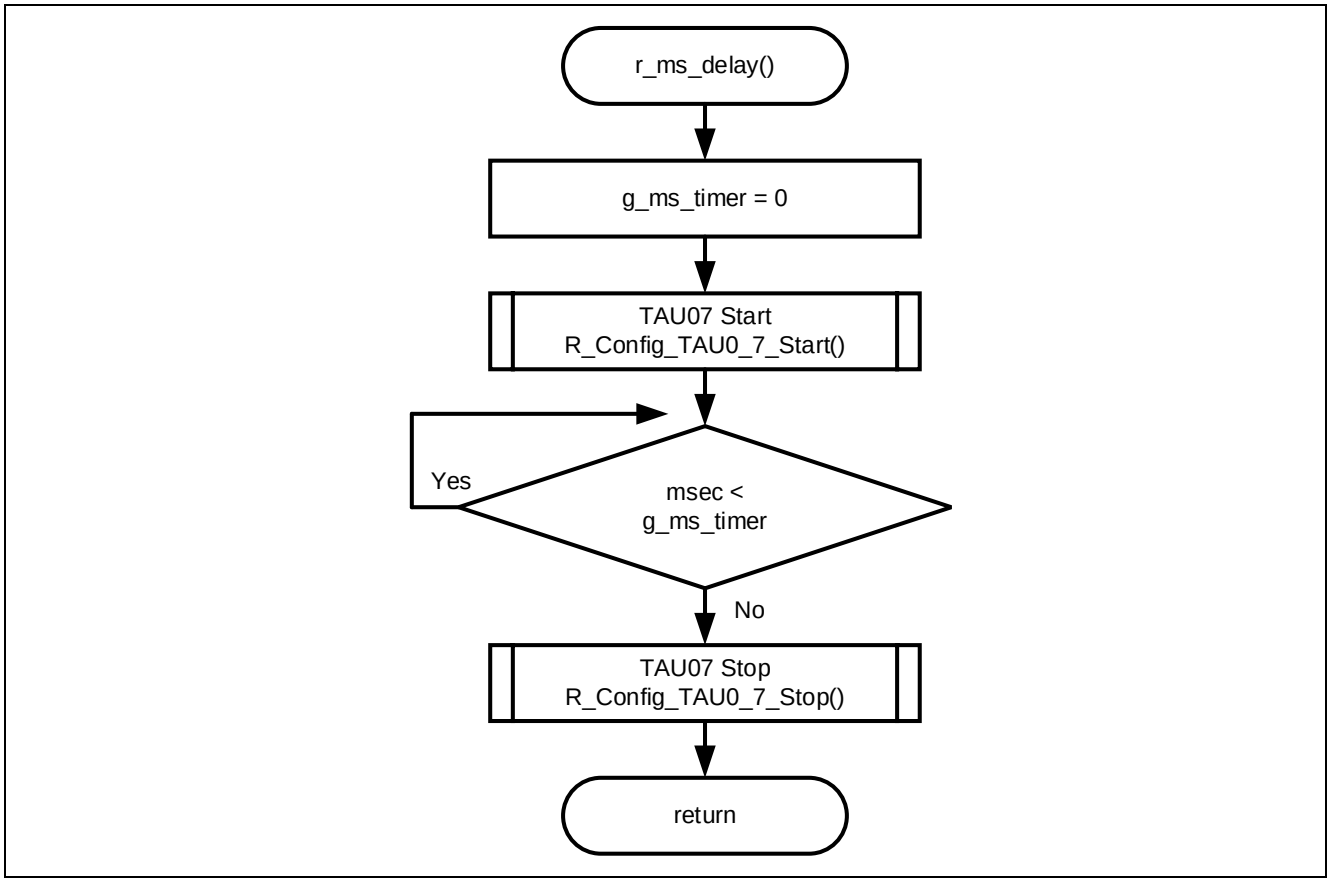
**Figure 5-24 CODEC module stop process**



### 5.1.8.22 Wait process

Figure 5-25 shows flowchart of Wait process.

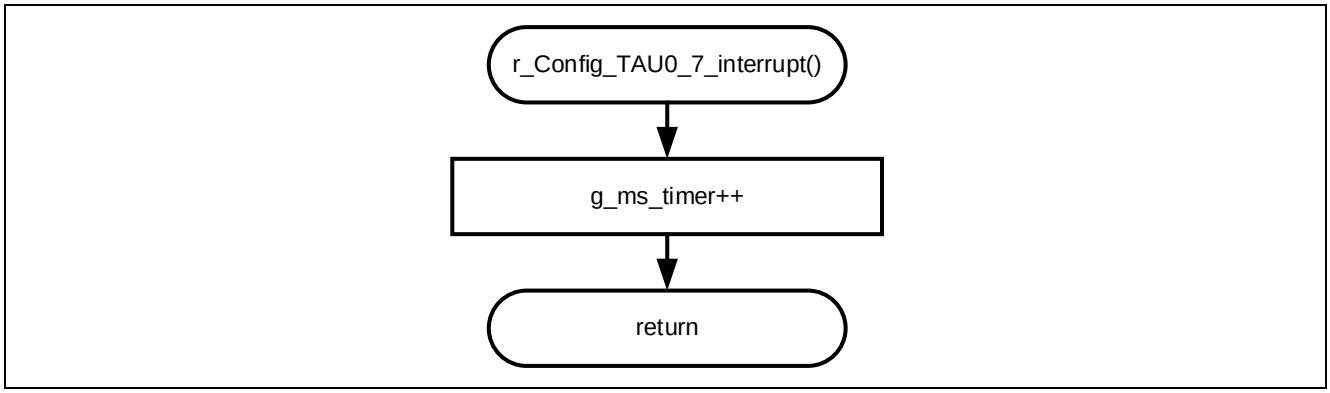
**Figure 5-25 Wait process**




### 5.1.8.23 TAU0 channel 7 interrupt process

Figure 5-26 shows flowchart of TAU0 channel 7 interrupt process

**Figure 5-26 TAU0 channel 7 interrupt process**



## 5.2 Application example

In addition to the sample code, this application note contains the following Smart Configurator configuration files.

r01an6420_elcl_i2s_slave.scfg

The following is a description of the file and examples of settings and notes for use.

### 5.2.1 r01an6420_elcl_i2s_slave.scfg

This is the Smart Configurator configuration file used in the sample code. It contains all the features configured in the Smart Configurator. The sample code settings are as follows.

**Table 5-6 Parameters of Smart Configurator (1/2)**

| Tag name | Component | Contents |
|---|---|---|
| Clocks | - | Operation mod: High-speed main mode 2.4 (V) ~ 5.5 (V)<br>$EV_{DD}$ setting: $1.8V \leqq EV_{DD0} < 5.5V$<br>High-speed on-chip oscillator: 32MHz<br>$f_{IHP}$: 32MHz<br>$f_{CLK}$: 32000kHz (High-speed on-chip oscillator)<br>$f_{SXP}$: 32.768kHz (Low-speed on-chip oscillator) |
| System | - | On-chip debug operation setting: COM port Note 1<br>Pseudo-RRM/DMM function setting: Used<br>Start/Stop function setting: Unused<br>Trace function setting: Used<br>Security ID setting: Use security ID<br>Security ID: 0x00000000000000000000<br>Security ID authentication failure setting: Do not erase flash memory data |
| Components | r_bsp | Start up select: Enable (use BSP startup)<br>Control of invalid memory access detection: Disable<br>RAM guard space (GRAM0-1): Disabled<br>Guard of control registers of port function (GPORT): Disabled<br>Guard of registers of interrupt function (GINT): Disabled<br>Guard of control registers of clock control function, voltage detector, and RAM parity error detection function (GCSC): Disabled<br>Data flash access control (DFLEN): Disables<br>Initialization of peripheral functions by Code Generator/Smart Configurator: Enable<br>API functions disable: Enable<br>Parameter check enable: Enable<br>Setting for starting the high-speed on-chip oscillator at the times of release from STOP mode and of transitions to SNOOZE mode: High-speed<br>Enable user warm start callback (PRE): Unused<br>Enable user warm start callback (POST): Unused<br>Watchdog Timer refresh enable: Unused |
| | Config_LVD0 | Operation mode setting: Reset mode<br>Voltage detection setting: Reset generation level ($V_{LVD0}$): 1.86 (V) |

**Table 5-7 Parameters of Smart Configurator (2/2)**

| Tag name | Component | Contents |
|---|---|---|
| Components | Config_TAU0_7 | Components: Interval timer<br>Operating mode: 16 bits count mode<br>Resource: TAU0_7<br>Operation clock: CK01<br>Clock source: $f_{CLK}/2^8$<br>Interval value: 1ms<br>Interrupt setting: use<br>Priority: Level 2 |
| | Config_PORT | Components: Port<br>Port selection: PORT5<br>P53: Out (Output 1) |

#### 5.2.1.1 Clocks

Set the clock used in the sample code.

#### 5.2.1.2 System

Set the on-chip debug of the sample code.

"Control of on-chip debug operation" and "Security ID authentication failure setting" affect "On-chip debugging is enabled" in "Table 5-2 Option Byte Settings". Note that changing the settings.

#### 5.2.1.3 r_bsp

Set the startup of the sample code.

#### 5.2.1.4 Config_LVD0

Set the power management of the sample code.

Affects "Setting of LVD0" in "Table 5-2 Option Byte Settings". Note that changing the settings.

#### 5.2.1.5 Config_TAU07

Set the TAU07 of the sample code.

In the sample code, it is used as a 1ms interval timer.

#### 5.2.1.6 Config_PORT

Set the port of the sample code.

In the sample code, P53 is used to control LED1.

### 5.2.2 How to change sound data

#### 5.2.2.1 To add new sound data to the sample code

In this sample code, new sound data can be added only to data with sampling frequency: 48 kHz and number of data: 32 bits.

(1) Please add new sound data to the following g_tx_buf[] in sound_data.c of the sample code.

```
#else /*  Not DATA_48K_32B */
/* For user setting */
const uint8_t g_tx_buf[] =
       {
              /* Add sound data here */
       };

#endif /* DATA_48K_32B */
```

(2) Set all sound data to be played in sound_data.h in the sample code to 0 as follows.

```
#define DATA_48K_32B (0)
```

(3) Set the constant "DECODE_PCM_SIZE" for the number of data to be sent in the same file.

```
  #else /* For user setting */
  #define DECODE_PCM_SIZE   (0)

  #endif
```

The above settings allow you to add new sound data to the sample code.

### 5.2.3 When using other CODEC modules

When using a CODEC module other than RS Audio CODEC 754-1974, delete the files necessary for setting up the RS Audio CODEC 754-1974 module in Table 5-1. Also, delete the CODEC module setup process and CODEC module start process in the main function and add various settings according to the CODEC module to be used.

## 6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

The following two types of project sample codes are available.

Sample code when RL78/G23 is master: Sample projects under the ¥Master folder

Sample code when RL78/G23 is slave: Sample project under the ¥Slave folder

## 7. Reference

RL78/G23 User's Manual: Hardware (R01UH0896E)

RL78 Family User's Manual: Software (R01US0015E)

RL78 Smart Configurator User's Guide: CS+ (R20AN0580E)

RL78 Smart Configurator User's Guide: e² studio (R20AN0579E)

RL78 Smart Configurator User's Guide: IAREW (R20AN0581E)

(The latest version can be downloaded from the Renesas Electronics website.)

Technical Update / Technical News

(The latest version can be downloaded from the Renesas Electronics website.

## Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Jun.24.22 | - | First edition |

# 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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. 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 shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended 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; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

## Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

## Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.