OKI SEMICONDUCTOR

SQ003116E101

ML610Q400 Series
Sample Program API Manual

3rd Edition
Issue Date: April 16, 2010

NOTICE

1. The information contained herein can change without notice owing to product and/or technical improvements. Before using the product, please make sure that the information being referred to is up-to-date.

2. The outline of action and examples for application circuits described herein have been chosen as an explanation for the standard action and performance of the product. When planning to use the product, please ensure that the external conditions are reflected in the actual circuit, assembly, and program designs.

3. When designing your product, please use our product below the specified maximum ratings and within the specified operating ranges including, but not limited to, operating voltage, power dissipation, and operating temperature.

4. **OKI SEMICONDUCTOR CO., LTD. assumes no responsibility or liability whatsoever for any failure or unusual or unexpected operation resulting from misuse, neglect, improper installation, repair, alteration or accident, improper handling, or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified operating range.**

5. Neither indemnity against nor license of a third party's industrial and intellectual property right, etc. is granted by us in connection with the use of the product and/or the information and drawings contained herein. No responsibility is assumed by us for any infringement of a third party's right which may result from the use thereof.

6. The products listed in this document are intended for use in general electronics equipment for commercial applications (e.g., office automation, communication equipment, measurement equipment, consumer electronics, etc.). These products are not, unless specifically authorized by OKI SEMICONDUCTOR CO., LTD., authorized for use in any system or application that requires special or enhanced quality and reliability characteristics nor in any system or application where the failure of such system or application may result in the loss or damage of property, or death or injury to humans.

   Such applications include, but are not limited to, traffic and automotive equipment, safety devices, aerospace equipment, nuclear power control, medical equipment, and life-support systems.

7. Certain products in this document may need government approval before they can be exported to particular countries. The purchaser assumes the responsibility of determining the legality of export of these products and will take appropriate and necessary steps at their own expense for these.

8. No part of the contents contained herein may be reprinted or reproduced without our prior permission.

Table of Contents

1. Overview

This document describes the functions of the standard APIs provided to control hardware devices that the ML610Q400 Series MCU (hereafter called the MCU) has.

Since a wide variety of API functions that can be of reference to software developers are provided, refer to them in designing a program. Refer also to the AP Notes provided separately.

2. Module APIs

Table 2-1 lists the module APIs.

Table 2-1 Module APIs

	Module name	Description
1	SA-ADC Module	successive-approximation type A/D converter
2	RC-ADC Module	RC oscillation-type A/D converter
3	Temperature Calculation Module	Temperature Calculation by RC-ADC
4	UART Module	UART
5	Baud rate adjustment Module	UART
6	I2C Module	I2C
7	EEPCOM Module	EEPROM control by I2C
8	LCD Module	LCD Driver
9	Key Read In Module	Port 0 Input port function
10	Melody Module	Melody Driver
11	Real time clock Module	RTC
12	Timer Module	Timer
13	Clock control Module	Clock generator
14	Time base counter Module	Time base counter
15	1kHz timer control Module	1kHz Timer
16	Stop Watch Module	Stop Watch by 1kHz timer
17	BLD Module	BLD circuit

3. Module Description

3.1. SA-ADC Module

3.1.1. Overview of Functions

The SA-ADC module controls the SA-ADC (successive-approximation type A/D converter) of the MCU.
The measurement function by use of SA-ADC is achieved by APIs that perform operations such as initialization (setting of conversion count, operating mode, the 2nd amp and so on), releasing the short-circuit between/short-circuiting the AIN0 and AIN1 pins, termination judgment, and conversion result acquisition.

3.1.2. List of APIs

Table 3-1 lists the SA-ADC module APIs.

Table 3-1 SA-ADC Module APIs

Function name	Description
saAdc_init function	Specifies the number of times conversion is performed by the SA-ADC (once/continuous), selects the clock frequency range of the HSCLK being used, specifies the operating mode, and sets the offset and gain of the 2nd amplifier.
saAdc_short_CH0CH1_on function	Short-circuits the input pins AIN0 and AIN1. *Only enabled at the time of differential amplification input
saAdc_short_CH0CH1_off function	Release the short-circuit between the input pins AIN0 and AIN1.
saAdc_execute function	Starts/stops SA-ADC conversion.
saAdc_checkFin function	Judges whether SA-ADC conversion has been terminated.
saAdc_getResult function	Acquires SA-ADC conversion results (12 bits).

3.1.3. List of Constants

The following tables list the constants used in the SA-ADC module.

Table 3-2 Constants for Arguments (1)

Constant name	Defined value	Description
SAADC_CH_CH0	1	Performs conversion. Specifies channel 0 only, as the channel of the SA-ADC.
SAADC_CH_CH1	2	Performs conversion. Specifies channel 1 only, as the channel of the SA-ADC.
SAADC_CH_CH0_CH1	3	Performs conversion. Specifies both channel 0 and channel 1, as the channels of the SA-ADC.
SAADC_LP_ONESHOT	0	Performs conversion only once, then stops.
SAADC_LP_CONTINUE	1	Performs conversion continuously.
SAADC_CK_LOW	0	Specify when HSCLK is set between 375 kHz and 1.1 MHz.
SAADC_CK_HIGH	1	Specify when HSCLK is set between 1.99 MHz and 4.2 MHz.
SAADC_EN_CH0NML_CH1NML	0	Channel 0: Direct input, Channel 1: Direct input
SAADC_EN_CH0NML_CH1AMP	1	Channel 0: Direct input, Channel 1: Amp input
SAADC_EN_CH0AMP_CH1AMP	2	Channel 0: Amplified input, Channel 1: Amp input
SAADC_EN_DIFF	3	Differential amp input
SAADC_OFFSET_M1_5	0	Offset adjustment: -1.5[%]
SAADC_OFFSET_M1_0	1	Offset adjustment: -1.0[%]
SAADC_OFFSET_M0_5	2	Offset adjustment: -0.5[%]
SAADC_OFFSET__0_0	3	Offset adjustment: 0[%]
SAADC_OFFSET_P0_5	4	Offset adjustment: 0.5[%]
SAADC_OFFSET_P1_0	5	Offset adjustment: 1.0[%]
SAADC_OFFSET_P1_5	6	Offset adjustment: 1.5[%]
SAADC_OFFSET_P2_0	7	Offset adjustment: 2.0[%]
SAADC_OFFSET_P2_5	8	Offset adjustment: 2.5[%]
SAADC_OFFSET_P3_0	9	Offset adjustment: 3.0[%]
SAADC_OFFSET_P3_5	10	Offset adjustment: 3.5[%]
SAADC_OFFSET_P4_0	11	Offset adjustment: 4.0[%]
SAADC_OFFSET_P4_5	12	Offset adjustment: 4.5[%]
SAADC_OFFSET_P5_0	13	Offset adjustment: 5.0[%]
SAADC_OFFSET_P5_5	14	Offset adjustment: 5.5[%]
SAADC_OFFSET_P6_0	15	Offset adjustment: 6.0[%]
SAADC_OFFSET_M9_5	16	Offset adjustment: -9.5[%]
SAADC_OFFSET_M9_0	17	Offset adjustment: -9.0[%]
SAADC_OFFSET_M8_5	18	Offset adjustment: -8.5[%]
SAADC_OFFSET_M8_0	19	Offset adjustment: -8.0[%]
SAADC_OFFSET_M7_5	20	Offset adjustment: -7.5[%]
SAADC_OFFSET_M7_0	21	Offset adjustment: -7.0[%]
SAADC_OFFSET_M6_5	22	Offset adjustment: -6.5[%]
SAADC_OFFSET_M6_0	23	Offset adjustment: -6.0[%]
SAADC_OFFSET_M5_5	24	Offset adjustment: -5.5[%]

Table 3-3 Constants for Arguments (2)

Constant name	Defined value	Description
SAADC_OFFSET_M5_0	25	Offset adjustment: -5.0[%]
SAADC_OFFSET_M4_5	26	Offset adjustment: -4.5[%]
SAADC_OFFSET_M4_0	27	Offset adjustment: -4.0[%]
SAADC_OFFSET_M3_5	28	Offset adjustment: -3.5[%]
SAADC_OFFSET_M3_0	29	Offset adjustment: -3.0[%]
SAADC_OFFSET_M2_5	30	Offset adjustment: -2.5[%]
SAADC_OFFSET_M2_0	31	Offset adjustment: -2.0[%]
SAADC_GAIN_1_0	0	Amp gain: 1x
SAADC_GAIN_1_5	1	Amp gain: 1.5x
SAADC_GAIN_2_0	2	Amp gain: 2x
SAADC_GAIN_2_5	3	Amp gain: 2.5x
SAADC_GAIN_3_0	4	Amp gain: 3x
SAADC_GAIN_3_5	5	Amp gain: 3.5x
SAADC_GAIN_4_0	6	Amp gain: 4x
SAADC_GAIN_4_5	7	Amp gain: 4.5x
SAADC_GAIN_5_0	8	Amp gain: 5x
SAADC_GAIN_5_5	9	Amp gain: 5.5x
SAADC_GAIN_6_0	10	Amp gain: 6x
SAADC_GAIN_6_5	11	Amp gain: 6.5x
SAADC_GAIN_7_0	12	Amp gain: 7x
SAADC_GAIN_7_5	13	Amp gain: 7.5x
SAADC_GAIN_8_0	14	Amp gain: 8x
SAADC_GAIN_8_5	15	Amp gain: 8.5x

Table 3-4 Constants for Return Values

Constant name	Defined value	Description
SAADC_R_OK	0	Processing succeeded.
SAADC_R_ERR_CH	-1	The channel No./channel bit setting is outside the range.
SAADC_R_ERR_LP	-2	The conversion count is outside the range.
SAADC_R_ERR_CK	-3	The HSCLK setting is outside the range.
SAADC_R_ERR_EN	-4	The operating mode is outside the range.
SAADC_R_ERR_OFFSET	-5	The amount of input offset is outside the range.
SAADC_R_ERR_GAIN	-6	The gain setting is outside the range.
SAADC_R_NOT_FIN	0	Conversion has never been terminated.
SAADC_R_FIN	1	Conversion has been terminated at least once.

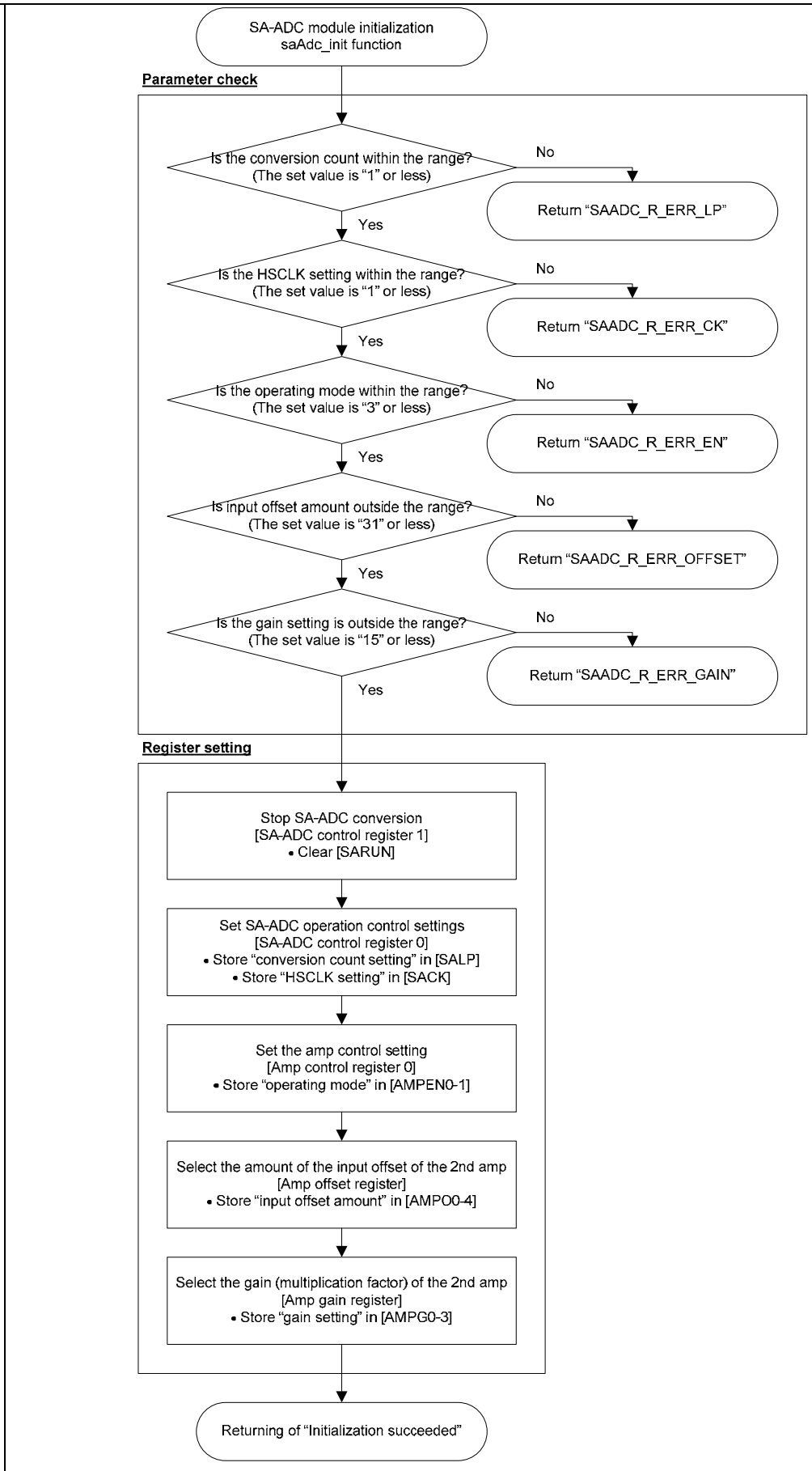
3.1.4. API Details

This section describes details of the SA-ADC module.

3.1.4.1. saAdc_init Function

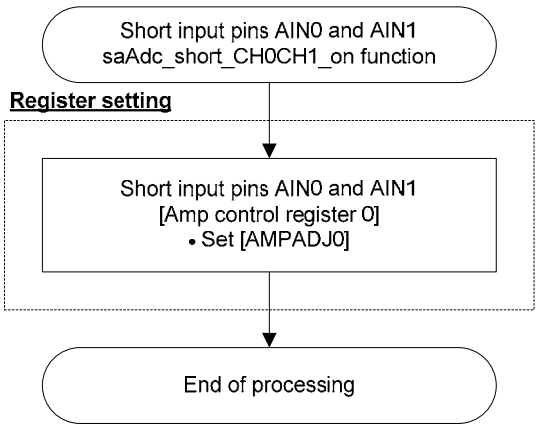
This function initializes the SA-ADC of the MCU. It initializes conversion count, HSCLK setting, operating mode, and the 2nd amp.

Function name:	int saAdc_init(unsigned char lp, unsigned char ck, unsigned char en, unsigned char offset, unsigned char gain)
Arguments:	unsigned char lp … Conversion count Convert only once: SAADC_LP_ONESHOT(=0) Convert continuously: SAADC_LP_CONTINUE(=1) unsigned char ck … HSCLK setting Sets HSCLK to 375 kHz to 1.1 MHz: SAADC_CK_LOW(=0) Sets HSCLK to 1.99 MHz to 4.2 MHz: SAADC_CK_HIGH(=1) unsigned char en … Operation mode Channel 0 = Direct input, Channel 1 = Direct input: SAADC_EN_CH0NML_CH1NML(=0) Channel 0 = Direct input, Channel 1 = Direct input: SAADC_EN_CH0NML_CH1AMP(=1) Channel 0 = Amp input, Channel 1 = Amp input: SAADC_EN_CH0AMP_CH1AMP(=2) Differential amp input: SAADC_EN_DIFF(=3) unsigned char offset … Input offset of the 2nd amplifier Offset adjustment: -1.5[%]: SAADC_OFFSET_M1_5(=0) … (* For details, see Table 3-2 and Table 3-3.) Offset adjustment: -2.0[%]: SAADC_OFFSET_M2_0(=31) unsigned char gain … Gain of the 2nd amplifier Amp gain 1x: SAADC_GAIN_1_0(=0) … (For details, see Table 3-3.) Amp gain 8.5x: SAADC_GAIN_8_5(=15)
Return values:	int Initialization succeeded: SAADC_R_OK(=0) The conversion count is outside the range: SAADC_R_ERR_LP(=-2) The HSCLK setting is outside the range: SAADC_R_ERR_CK(=-3) The operating mode is outside the range: SAADC_R_ERR_EN(=-4) The amount of input offset is outside the range: SAADC_R_ERR_OFFSET(=-5) The gain setting is outside the range: SAADC_R_ERR_GAIN(=-6)
Processing:	See next page.


SA-ADC module initialization
saAdc_init function
Parameter check
Is the conversion count within the range?
(The set value is "1" or less)
No
Return "SAADC_R_ERR_LP"
Yes
Is the HSCLK setting within the range?
(The set value is "1" or less)
No
Return "SAADC_R_ERR_CK"
Yes
Is the operating mode within the range?
(The set value is "3" or less)
No
Return "SAADC_R_ERR_EN"
Yes
Is input offset amount outside the range?
(The set value is "31" or less)
No
Return "SAADC_R_ERR_OFFSET"
Yes
Is the gain setting is outside the range?
(The set value is "15" or less)
No
Return "SAADC_R_ERR_GAIN"
Yes
Register setting
Stop SA-ADC conversion
[SA-ADC control register 1]
• Clear [SARUN]
Set SA-ADC operation control settings
[SA-ADC control register 0]
• Store "conversion count setting" in [SALP]
• Store "HSCLK setting" in [SACK]
Set the amp control setting
[Amp control register 0]
• Store "operating mode" in [AMPEN0-1]
Select the amount of the input offset of the 2nd amp
[Amp offset register]
• Store "input offset amount" in [AMPO0-4]
Select the gain (multiplication factor) of the 2nd amp
[Amp gain register]
• Store "gain setting" in [AMPG0-3]
Returning of "Initialization succeeded"

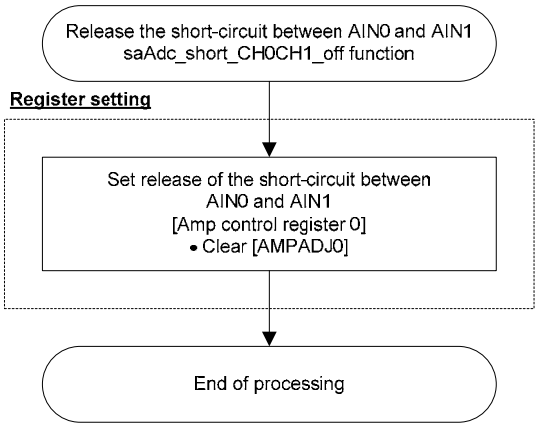
3.1.4.2. saAdc_short_CH0CH1_on Function

This function short-circuits the input pins AIN0 and AIN1. Only enabled at differential amplifier input (when SAADC_EN_DIFF is specified for the argument “en” of the saAdc_init function).

Function name:	void saAdc_short_CH0CH1_on(void)
Arguments:	None
Return values:	None
Processing:	

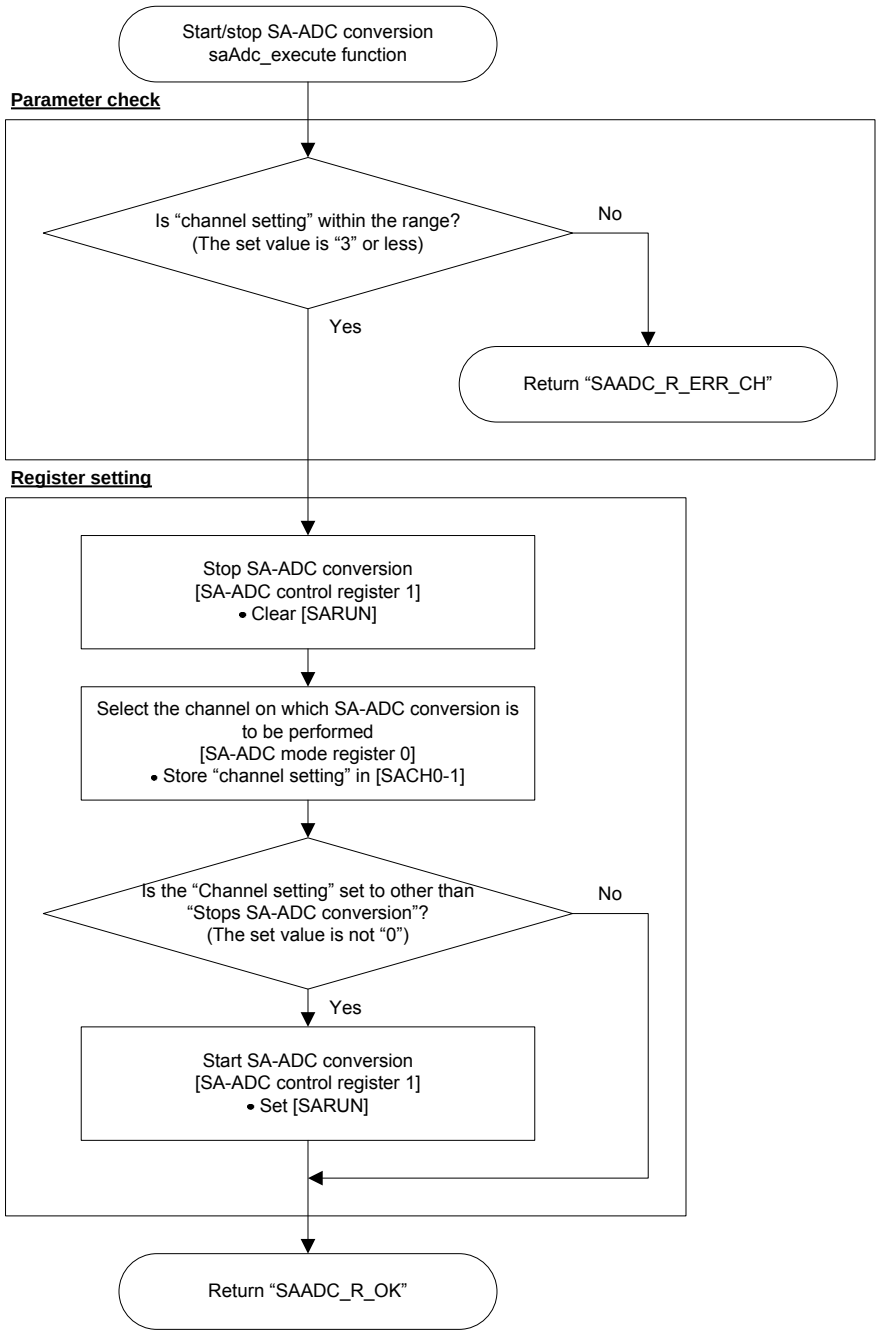
3.1.4.3. saAdc_short_CH0CH1_off Function

This function releases the short-circuit between the input pins AIN0 and AIN1.

Function name:	void saAdc_short_CH0CH1_off(void)
Arguments:	None
Return values:	None
Processing:	

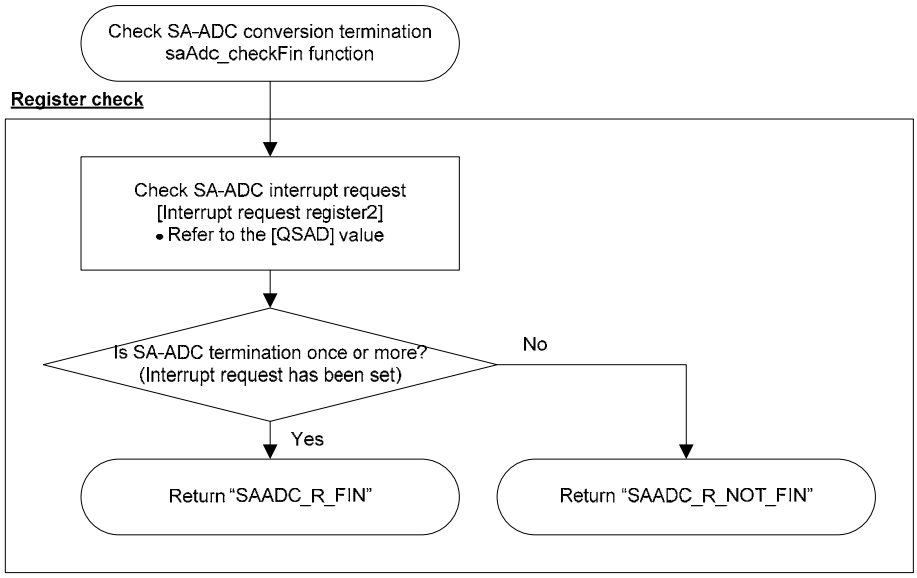
3.1.4.4. saAdc_execute Function

This function starts/stops SA-ADC conversion.

Function name:	int saAdc_execute(unsigned char chBit)
Arguments:	unsigned char chBit … Channel setting Stops SA-ADC conversion: SAADC_CHBIT_OFF(=0) Channel 0 only: SAADC_CHBIT_CH0(=1) Channel 1 only: SAADC_CHBIT_CH1(=2) Channels 0 and 1: SAADC_CHBIT_CH0_CH1(=3)
Return values:	int Start/stop processing succeeded: SAADC_R_OK(=0) The channel setting is outside the range: SAADC_R_ERR_CH (=-1)
Processing:	

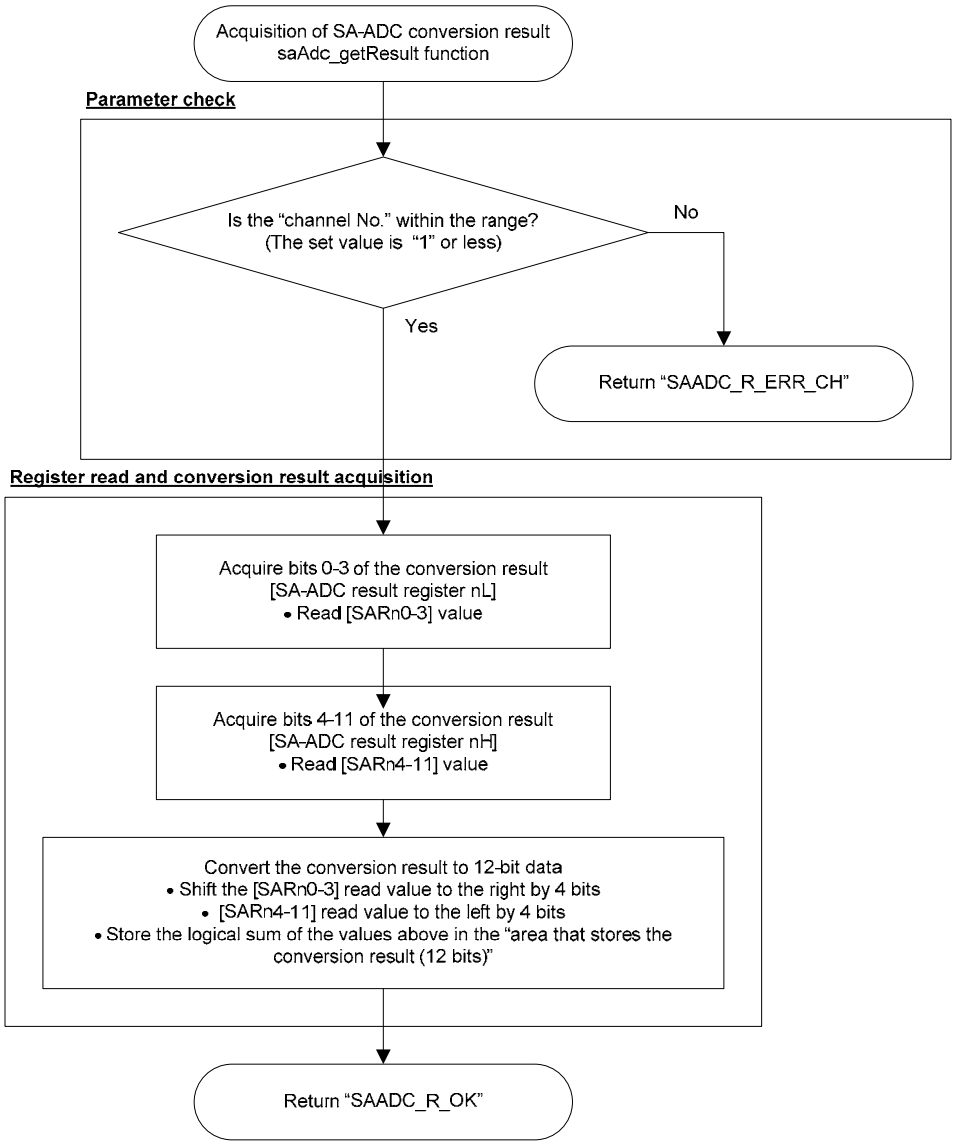
3.1.4.5. saAdc_checkFin Function

This function checks the QSAD bit of the interrupt request register of the MCU to judge whether SA-ADC conversion has been terminated or not.

Function name:	int saAdc_checkFin(void)
Arguments:	None
Return values:	Judgment result as to whether conversion has been terminated or not. Conversion has been terminated at least once: SAADC_R_FIN(=1) Conversion has never been terminated: SAADC_R_NOT_FIN(=0)
Processing:	

3.1.4.6. saAdc_getResult Function

This function reads the H/L value of the SA-ADC result register and acquires the SA-ADC conversion result (12-bit).

Function name:	int saAdc_getResult(unsigned char chNo, unsigned short *result)
Arguments:	unsigned char chNo … SA-ADC channel No. (0 to 1) unsigned char *result … Pointer to the area that stores the SA-ADC conversion result (12-bit)
Return values:	int Acquisition succeeded: SAADC_R_OK(=0) The channel No. is outside the range: SAADC_R_ERR_CH (=-1)
Processing:	

3.2. RC-ADC Module

3.2.1. Overview of Functions

The RC-ADC module controls the RC-ADC (RC oscillation-type A/D converter) of the MCU.
The RCAD measurement function is achieved by APIs that perform operations such as initialization (setting of oscillation mode and reference clock), RC-ADC counter setting, RC-ADC conversion start/stop, termination judgment, conversion result acquisition, and RC oscillation monitor output On/Off control.

3.2.2. List of APIs

The following table lists the RC-ADC module APIs.

Table 3-5 RC-ADC Module APIs

Function name	Description
rcAdc_init function	Specifies the RC-ADC oscillation mode and sets the reference clock.
rcAdc_setCounter function	Sets a value in the RC-ADC counter.
rcAdc_execute function	Executes RC-ADC conversion start/stop processing.
rcAdc_checkFin function	Checks whether RC-ADC conversion has terminated or not.
rcAdc_getResult function	Acquires the conversion result (24 bits) of the specified RC-ADC counter.
rcAdc_monitor_on function	Enables the RC oscillation monitor output.
rcAdc_monitor_off function	Disables the RC oscillation monitor output.

3.2.3. List of Constants

The following tables list the constants used in the RC-ADC module.

Table 3-6 Constants for Arguments

Constant name	Defined value	Description
RCADC_MODE0	0	Oscillation mode: IN0 pin external clock input mode
RCADC_MODE1	1	Oscillation mode: RS0-CS0 oscillation mode
RCADC_MODE2	2	Oscillation mode: RT0-CS0 oscillation mode
RCADC_MODE3	3	Oscillation mode: RT0-1-CS0 oscillation mode
RCADC_MODE4	4	Oscillation mode: RS0-CT0 oscillation mode
RCADC_MODE5	5	Oscillation mode: RS1-CS1 oscillation mode
RCADC_MODE6	6	Oscillation mode: RT1-CS1 oscillation mode
RCADC_MODE6	7	Oscillation mode: IN1 pin external clock input mode
RCADC_CK_LOW	0	Reference clock of counter A: LSCLK
RCADC_CK_LOW2	1	Reference clock of counter A: LSCLK×2
RCADC_CK_HIGH	2	Reference clock of counter A: HSCLK
RCADC_CK_HIGH_DIV2	3	Reference clock of counter A: 1/2 HSCLK
RCADC_CK_HIGH_DIV4	4	Reference clock of counter A: 1/4 HSCLK
RCADC_CK_HIGH_DIV8	5	Reference clock of counter A: 1/8 HSCLK
RCADC_A	0	Specifies counter A.
RCADC_B	1	Specifies counter B.
RCADC_OFF	0	Stops A/D conversion.
RCADC_RUN	1	Starts A/D conversion.

Table 3-7 Constants for Return Values

Constant name	Defined value	Description
RCADC_R_OK	0	Processing succeeded.
RCADC_R_ERR_MODE	-1	The oscillation mode setting is outside the range.
RCADC_R_ERR_CK	-2	The reference clock setting is outside the range.
RCADC_R_ERR_CNTNO	-3	The counter No. is outside the range.
RCADC_R_ERR_CNTVAL	-4	The count value is outside the range.
SAADC_R_ERR_RUN	-5	The A/D conversion stop/start specified is outside the range.

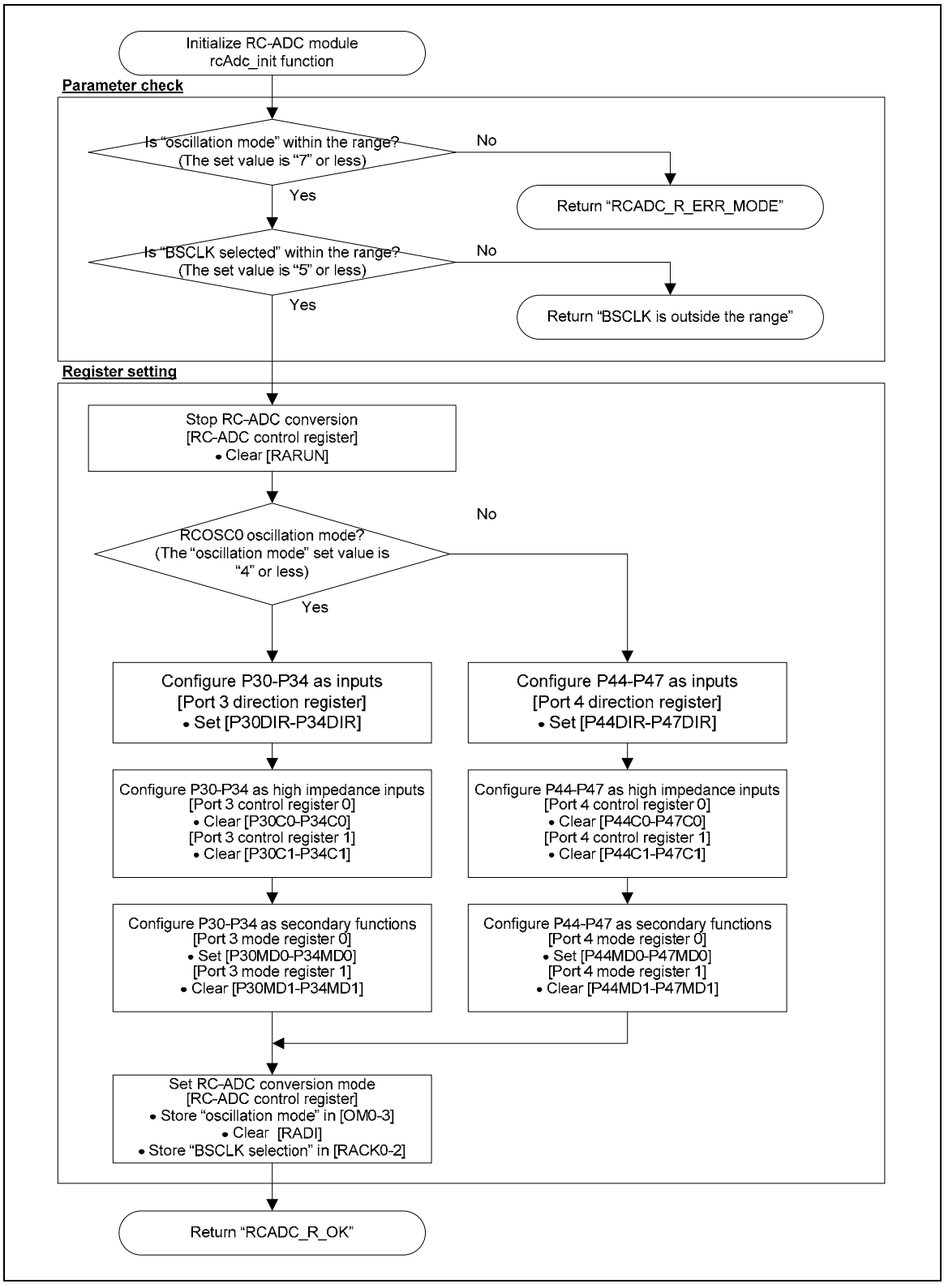
3.2.4. API Details

This section describes details of the RC-ADC module.

3.2.4.1. rcAdc_init Function

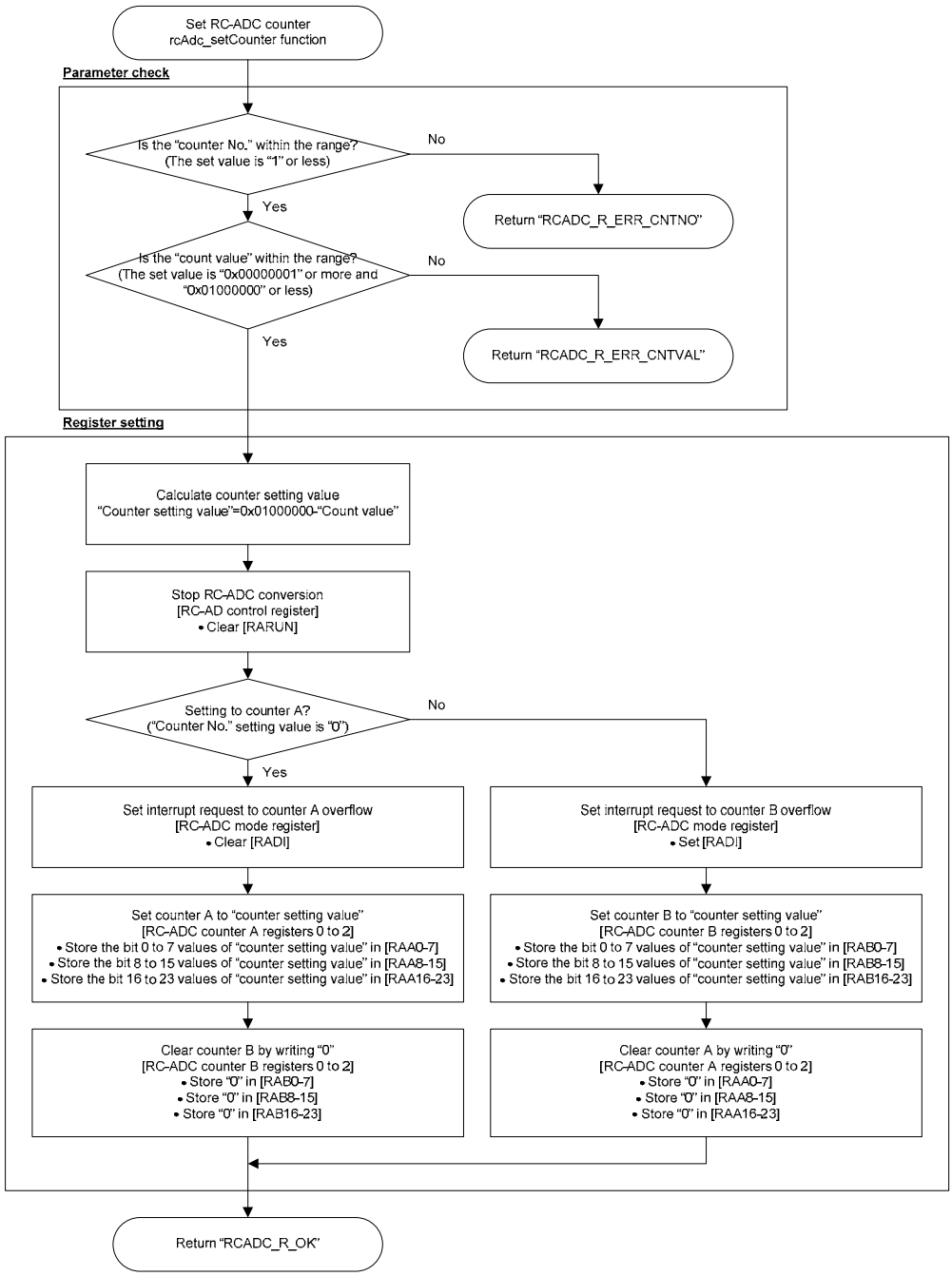
This function specifies the oscillation mode of the RC-ADC and sets the reference clock.

Function name:	int rcAdc_init(unsigned char mode, unsigned char ck)
Arguments:	unsigned char mode … Oscillation mode of the RC oscillation circuit IN0 pin external clock input mode: RCADC_MODE0(=0) RS0-CS0 oscillation mode: RCADC_MODE1(=1) RT0-CS0 oscillation mode: RCADC_MODE2(=2) RT0-1-CS0 oscillation mode: RCADC_MODE3(=3) RS0-CT0 oscillation mode: RCADC_MODE4(=4) RS1-CS1 oscillation mode: RCADC_MODE5(=5) RT1-CS1 oscillation mode: RCADC_MODE6(=6) IN1 pin external clock input mode: RCADC_MODE7(=7) unsigned char ck … Reference clock of counter A (BSCLK) LSCLK: RCADC_CK_LOW(=0) LSCLK×2: RCADC_CK_LOW2(=1) HSCLK: RCADC_CK_HIGH(=2) 1/2 HSCLK: RCADC_CK_HIGH_DIV2(=3) 1/4 HSCLK: RCADC_CK_HIGH_DIV4(=4) 1/8 HSCLK: RCADC_CK_HGIH_DIV8(=5)
Return values:	int Initialization succeeded: RCADC_R_OK(=0) The setting value of the oscillation mode is outside the range: RCADC_R_ERR_MODE(=-1) The setting value of the reference clock is outside the range: RCADC_R_ERR_CK(=-2)
Processing:	See next page.


Initialize RC-ADC module
rcAdc_init function
Parameter check
Is "oscillation mode" within the range?
(The set value is "7" or less)
No
Return "RCADC_R_ERR_MODE"
Yes
Is "BSCLK selected" within the range?
(The set value is "5" or less)
No
Return "BSCLK is outside the range"
Yes
Register setting
Stop RC-ADC conversion
[RC-ADC control register]
• Clear [RARUN]
RCOSC0 oscillation mode?
(The "oscillation mode" set value is
"4" or less)
No
Yes
Configure P30-P34 as inputs
[Port 3 direction register]
• Set [P30DIR-P34DIR]
Configure P44-P47 as inputs
[Port 4 direction register]
• Set [P44DIR-P47DIR]
Configure P30-P34 as high impedance inputs
[Port 3 control register 0]
• Clear [P30C0-P34C0]
[Port 3 control register 1]
• Clear [P30C1-P34C1]
Configure P44-P47 as high impedance inputs
[Port 4 control register 0]
• Clear [P44C0-P47C0]
[Port 4 control register 1]
• Clear [P44C1-P47C1]
Configure P30-P34 as secondary functions
[Port 3 mode register 0]
• Set [P30MD0-P34MD0]
[Port 3 mode register 1]
• Clear [P30MD1-P34MD1]
Configure P44-P47 as secondary functions
[Port 4 mode register 0]
• Set [P44MD0-P47MD0]
[Port 4 mode register 1]
• Clear [P44MD1-P47MD1]
Set RC-ADC conversion mode
[RC-ADC control register]
• Store "oscillation mode" in [OM0-3]
• Clear [RADI]
• Store "BSCLK selection" in [RACK0-2]
Return "RCADC_R_OK"

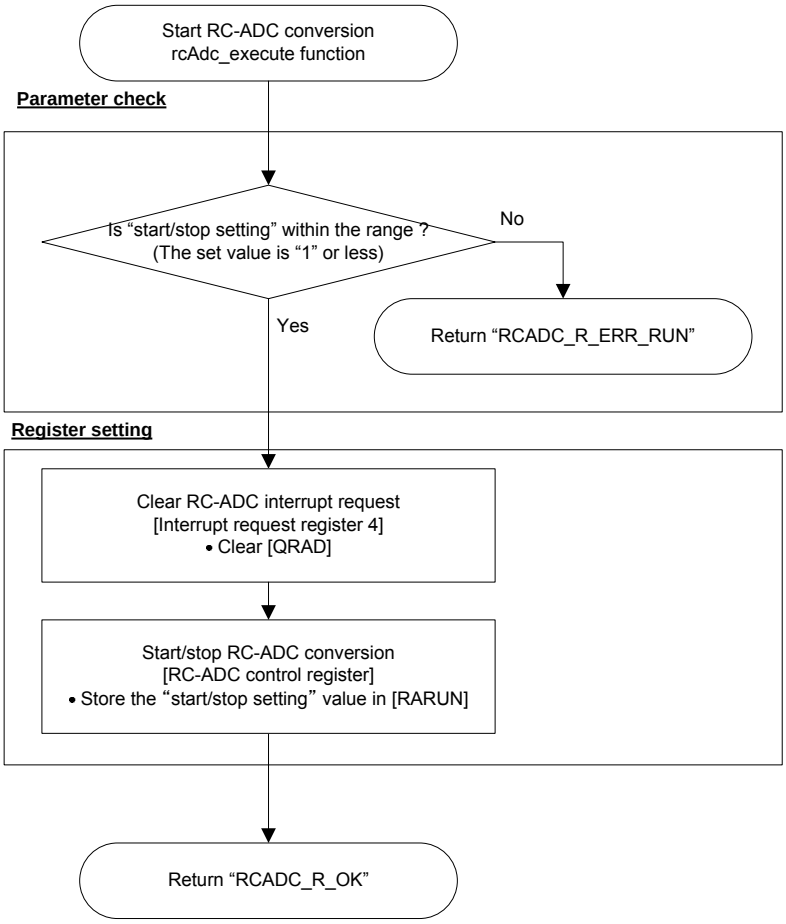
3.2.4.2. rcAdc_setCounter Function

This function sets a count value in the counters of the RC-ADC. If a value is set in counter A, counter B is cleared to 0, and vice versa.

Function name:	int rcAdc_setCounter(unsigned char cntNo, unsigned long value)
Arguments:	unsigned char cntNo … Counter No. Counter A: RCADC_A(=0) Counter B: RCADC_B(=1) unsigned long value … Count value (0x00000001 to 0x01000000)
Return values:	int Setting succeeded: RCADC_R_OK(=0) The counter No. is outside the range: RCADC_R_ERR_CNTNO(=-3) The count value is outside the range: RCADC_R_ERR_CNTVAL(=-4)
Processing:	

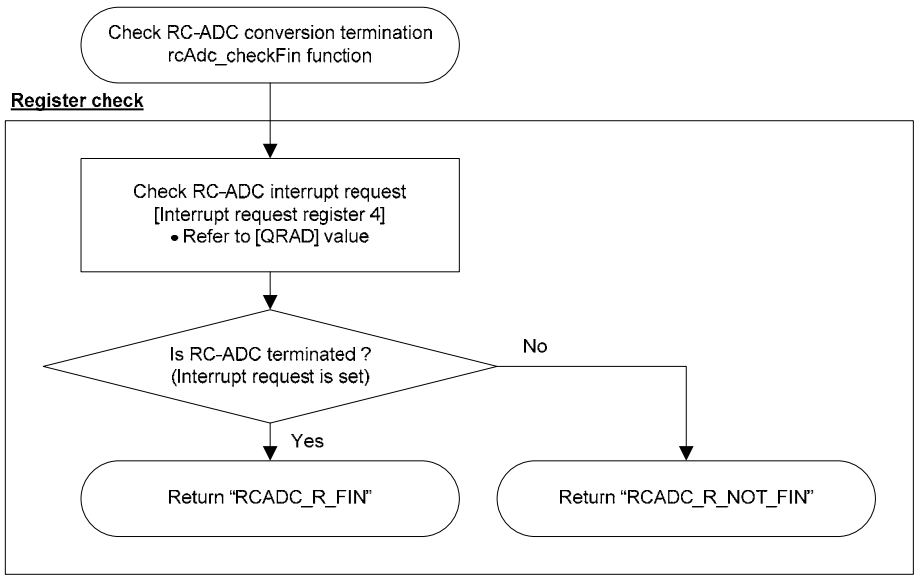
3.2.4.3. rcAdc_execute Function

This function starts/stops RC-ADC conversion.

Function name:	int rcAdc_execute(unsigned char run)
Arguments:	unsigned char run … Stop/start of A/D conversion Stop: RCADC_OFF(=0) Start: RCADC_RUN(=1) unsigned long value … Count value (0x00000001 to 0x01000000)
Return values:	int Setting succeeded: RCADC_R_OK(=0) Stop/start setting is outside the range: RCADC_R_ERR_RUN(=-5)
Processing:	

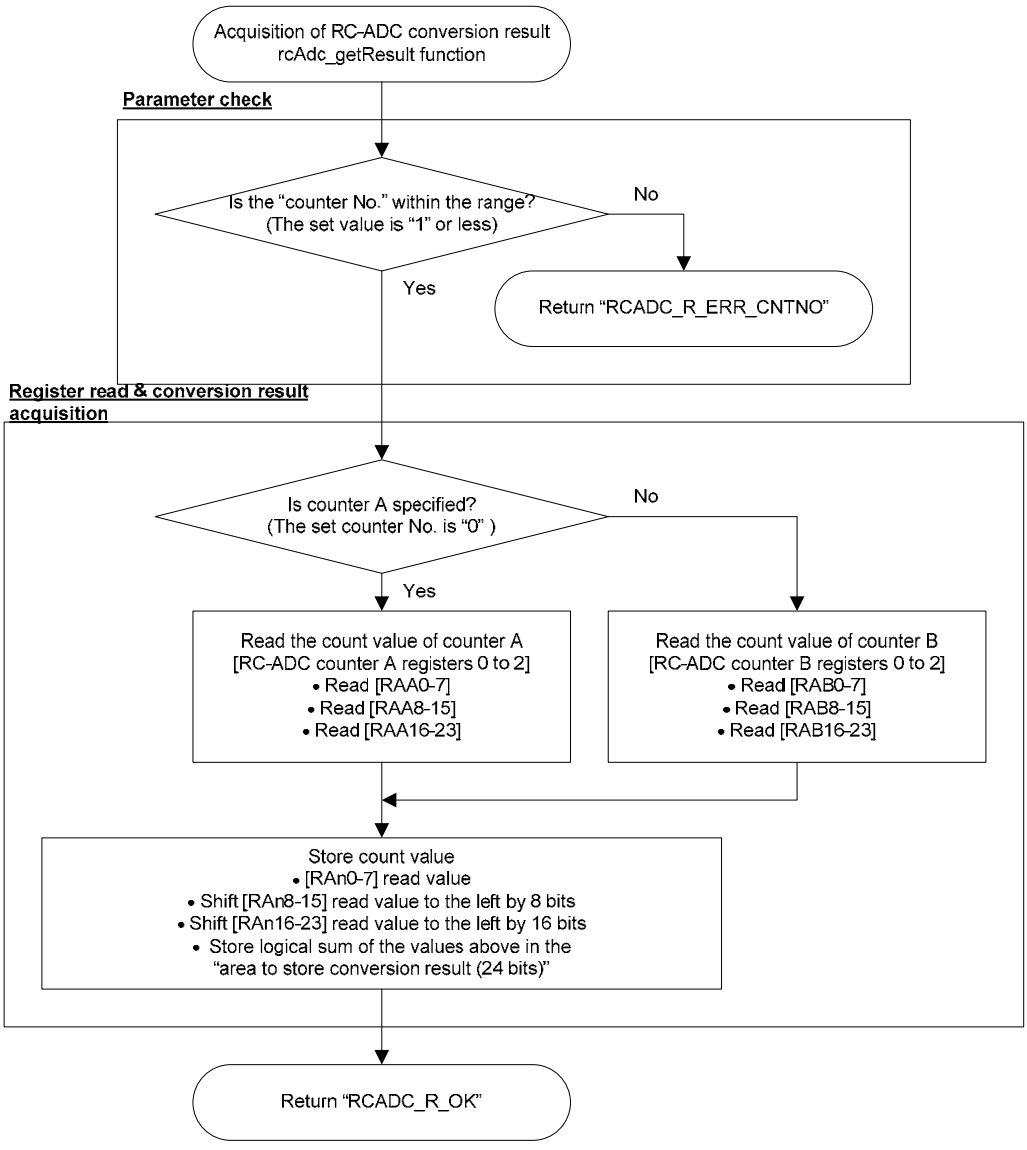
3.2.4.4. rcAdc_checkFin Function

This function judges whether RC-ADC conversion has been completed or not.

Function name:	int rcAdc_execute(void)
Arguments:	None
Return values:	int Conversion has not been completed yet: RCADC_R_NOT_FIN(=0) Conversion ha been completed: RCADC_R_FIN(=1)
Processing:	

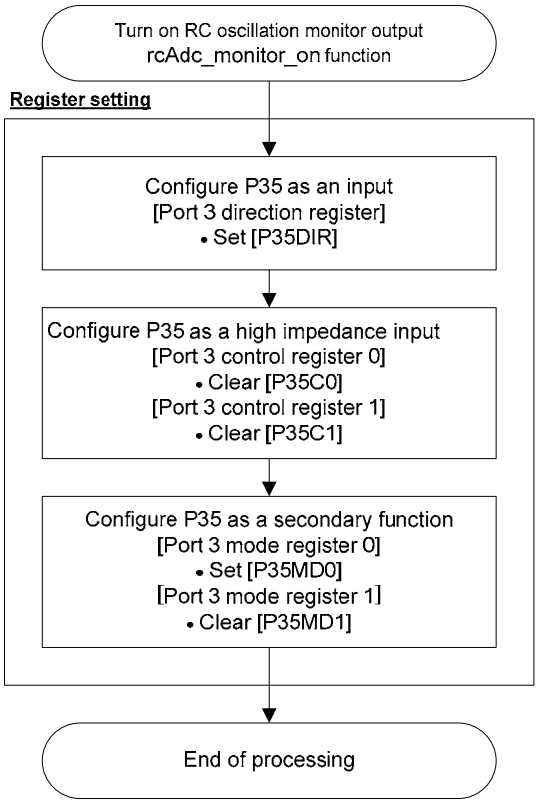
3.2.4.5. rcAdc_getResult Function

This function acquires conversion result (24 bits) from the specified counter of the RC-ADC.

Function name:	int rcAdc_getResult(unsigned char cntNo, unsigned long *result)
Arguments:	unsigned char cntNo … Counter No. Counter A: RCADC_A(=0) Counter B: RCADC_B(=1) unsigned long *result …Pointer to the area that stores the conversion result (24 bits)
Return values:	int Acquisition succeeded: RCADC_R_OK(=0) Counter No. is outside the range: RCADC_R_ERR_CNTNO(=-3)
Processing:	

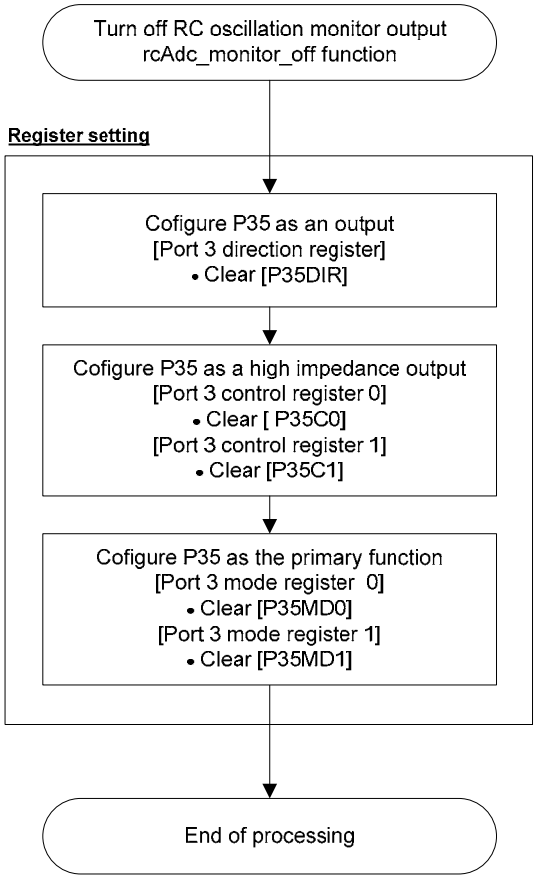
3.2.4.6. rcAdc_monitor_on Function

This function enables RC oscillation monitor output.

Function name:	void rcAdc_monitor_on(void)
Arguments:	None
Return values:	None
Processing:	

3.2.4.7. rcAdc_monitor_off Function

This function disables RC oscillation monitor output.

Function name:	void rcAdc_monitor_off(void)
Arguments:	None
Return values:	None
Processing:	

3.3. Temperature Calculation Module

3.3.1. Overview of Functions

The temperature calculation module calculates a temperature based on the RC-ADC (RC oscillation type A/D converter) conversion result.

3.3.2. List of APIs

The following table lists the temperature calculation module APIs.

Table 3-8 Temperature Calculation Module APIs

Function name	Description
temp_calc function	Calculates temperature from the RC-ADC count value.

3.3.3. List of Constants

The following table lists the constants used in the temperature calculation module.

Table 3-9 Constants for Return Values

Constant name	Defined value	Description
TEMP_R_OK	0	Processing succeeded.
TEMP_R_ERR_L	-1	The count value is smaller than the minimum value.
TEMP_R_ERR_H	-2	The count value is greater than the maximum value.

3.3.4. Structures

This section describes the structures referred in the temperature calculation module.

■ Temperature calculation table

```
typedef struct {
    unsigned long    baseCnt;    // Frequency count ratio
    unsigned long    divStep;    // Temperature slope
    signed long      offset;     // Temperature offset
} tTempTableList;
```

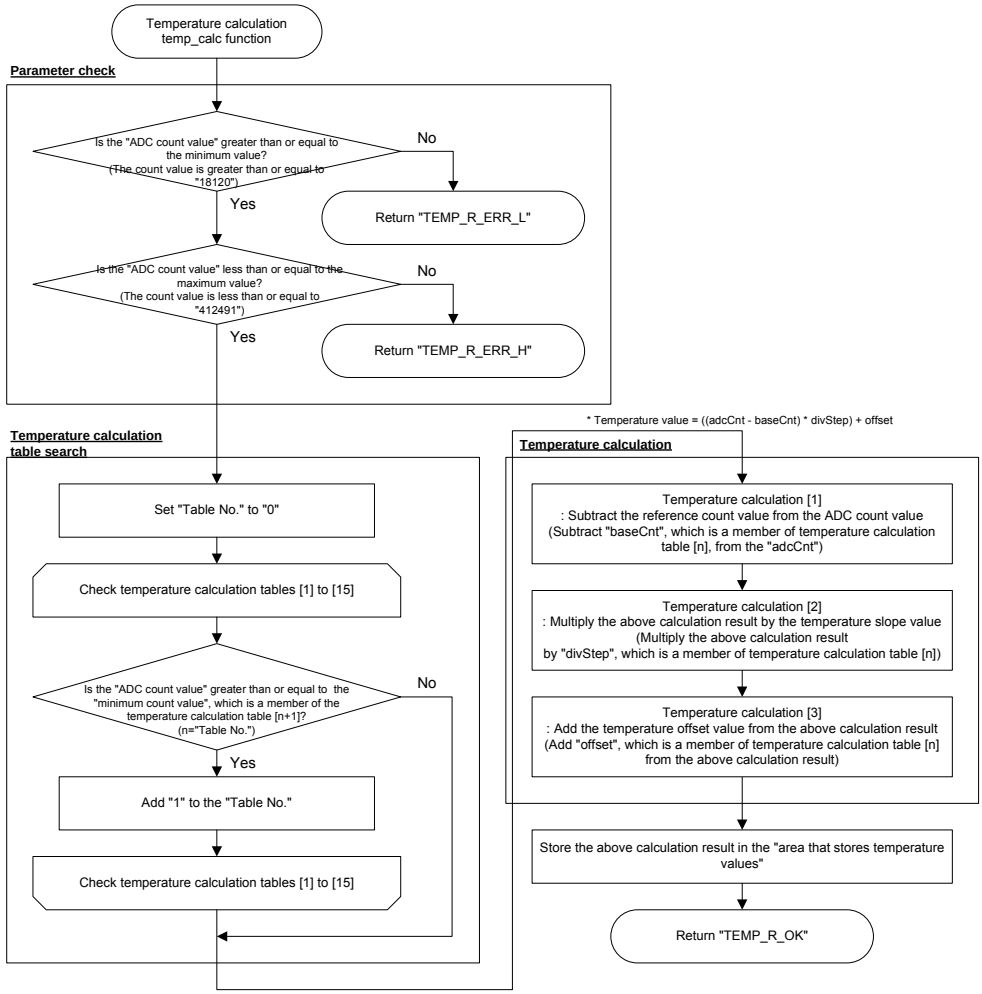
* About the example of the value setting of the temperature calculation table value, see the chapter "Temperature Calculation Module" in the "ML610Q400 Series Sample Program AP Notes For Sensor/Measurement Application".

3.3.5. API Details

This section describes details of the temperature calculation module APIs.

3.3.5.1. temp_calc Function

This function obtains temperature from the count value of the RC-ADC.

Function name:	int temp_calc(unsigned long adcCnt, signed short *temp tTempTableList *pTbl;)
Arguments:	unsigned long adcCnt … Count value of the RC-ADC signed short *temp … Pointer to the area that stores the temperature values (in 0.1°C steps) Example: For –10.5°C, "–105" will be stored. tTempTableList *pTbl … Pointer to the temperature calculation table
Return values:	int Acquisition succeeded: TEMP_R_OK(=0) The count value is less than the minimum value: TEMP_R_ERR_L(= –1) The count value is greater than the maximum value: TEMP_R_ERR_H(= –2)
Processing:	

3.4. UART Module

3.4.1. Overview of Functions

The UART module controls the UART (asynchronous serial interface) of the MCU.
The UART communication function is achieved by APIs that perform operations such as initialization (sets settings for the clock to be input to the baud rate generator, communication mode, and baud rate), data transmission/reception start/stop, continuation of data transmission/reception, transmit/receive interrupt request acquisition and clear, and transmit/receive size acquisition.

3.4.2. List of APIs

The following table lists the UART module APIs.

Table 3-10 UART Module APIs

Function name	Description
uart_init function	Selects the clock to be input to the baud rate generator and sets the communication mode (data length = 8 bits, no parity bit, 2 stop bits, positive logic, LSB first) and baud rate.
uart_startSend function	Executes data transmission start processing.
uart_startReceive function	Executes data reception start processing.
uart_continue function	Executes data transmission/reception continuation processing.
uart_stop function	Executes data transmission/reception stop processing.
uart_checkIRQ function	Checks if a transmit/receive interrupt request is present or absent.
uart_clearIRQ function	Clears the transmission/reception interrupt request.
uart_getTransSize function	Checks the size of the data transmitted/received
uart_PortClear function	Initializes the transmission/reception port setting.
uart_PortSet function	Set the transmission/reception port for UART communication.

3.4.3. Callback Function

This section describes the callback function specified by the uart_startSend function or uart_startReceive function.
The callback function is called upon completion of transmission/reception from the UART module. Describe processing required upon completion of transmission/reception within the callback function.

Definition of the callback function

```
typedef void (*cbfUart)( unsigned int size, unsigned char errStat )
```

<Example of Definition>

Function name:	void uart_callback(unsigned int size, unsigned char errStat)
Arguments:	unsigned int size … Size of transmitted/received data unsigned char errStat … Transmission/reception result bit0 … 0 = No framing error /1 = Framing error bit1 … 0 = No overrun error /1 = Overrun error bit2 … 0 = No parity error /1 = Prity error
Return values:	None

3.4.4. List of Constants

The following tables list the constants used in the UART module.

Table 3-11 Constants for Arguments

Constant name	Defined value	Description
UART_CK_LSCLK	0	Specifies LSCLK as the clock to be input to the baud rate generator.
UART_CK_LSCLK2	1	Specifies LSCLK×2 as the clock to be input to the baud rate generator.
UART_CK_HSCLK	2	Specifies HSCLK as the clock to be input to the baud rate generator.
UART_BR_2400BPS	2400	Specifies 2400 bps as the baud rate.
UART_BR_4800BPS	4800	Specifies 4800 bps as the baud rate.
UART_BR_9600BPS	9600	Specifies 9600 bps as the baud rate.
UART_BR_19200BPS	19200	Specifies 19200 bps as the baud rate.
UART_BR_38400BPS	38400	Specifies 38400 bps as the baud rate.
UART_BR_57600BPS	57600	Specifies 57600 bps as the baud rate.
UART_BR_115200BPS	115200	Specifies 115200 bps as the baud rate.
UART_LG_8BIT	0	Specifies 8-bit length as the communication date length.
UART_LG_7BIT	1	Specifies 7-bit length as the communication date length.
UART_LG_6BIT	2	Specifies 6-bit length as the communication date length.
UART_LG_5BIT	3	Specifies 5-bit length as the communication date length.
UART_PT_EVEN	0	Specifies an even parity bit.
UART_PT_ODD	1	Specifies an odd parity bit.
UART_PT_NON	2	Specifies no parity.
UART_STP_1BIT	0	Specifies 1 bit as the stop bit length.
UART_STP_2BIT	1	Specifies 2 bits as the stop bit length.
UART_NEG_POS	0	Specifies positive logic.
UART_NEG_NEG	1	Specifies negative logic.
UART_DIR_LSB	0	Specifies LSB first.
UART_DIR_MSB	1	Specifies MSB first.

Table 3-12 Constants for Return Values

Constant name	Defined value	Description
UART_R_OK	0	Processing succeeded.
UART_R_ERR_CS	-1	The input clock setting is outside the range.
UART_R_ERR_BR	-2	The specified baud rate is unsettable.
UART_R_ERR_LG	-3	The communication data length is outside the range.
UART_R_ERR_PT	-4	The parity setting is outside the range.
UART_R_ERR_STP	-5	The stop bit length is outside the range.
UART_R_ERR_NEG	-6	The positive logic/negative logic specification is outside the range.
UART_R_ERR_DIR	-7	The LSB/MSB specification is outside the range.
UART_R_TRANS_FIN	1	Transmit/receive processing terminated.
UART_R_TRANS_CONT_OK	0	The transmit/receive processing is going on (transmission succeeded)
UART_R_TRANS_CONT_NG	-1	The transmit/receive processing is going on (transmission failed)
UART_R_TRANS_FIN_NG	-2	Transmit/receive processing terminated (termination failed).
UART_R_IRQ	1	Transmit/receive complete interrupt request is present.
UART_R_NON_IRQ	0	Transmit/receive complete interrupt request is absent.

3.4.5. Structures

This section describes the structures used in the UART module.

■ UART setting parameters

```
typedef struct {
    unsigned long    br;     // Specify the baud rate.
    unsigned char    lg;     // Specify the communication data length.
    unsigned char    pt;     // Select even parity/odd parity/no parity.
    unsigned char    stp;    // Select the stop bit length.
    unsigned char    neg;    // Select positive logic/negative logic.
    unsigned char    dir;    // Select LSB first/MSB first.
} tUartSetParam;
```

For the setting values, see Table 3-11

■ Callback functions

typedef void (*tCallBack)(unsigned int size, unsigned char errStat)

■ UART transmit/receive control parameters

```
typedef struct {
    unsigned char *  data;     // Pointer to the area that contains transmit/receive data
    unsigned int     size;     // Transmit/receive data size
    unsigned int     cnt;      // Size of transmitted/received data
    tCallBack        callBack  // Callback function
    unsigned char    errStat   // Error status
} tUartCtrlParam;
```

3.4.6. List of Variables

The following table lists the variables used in the UART module.

Variable name	Initial value	Description
static tUartCtrlParam _gsCtrlParam	data: NULL size: 0 cnt: 0 callBack: NULL errStat: 0	Variable to manage the data for UART transmission/reception

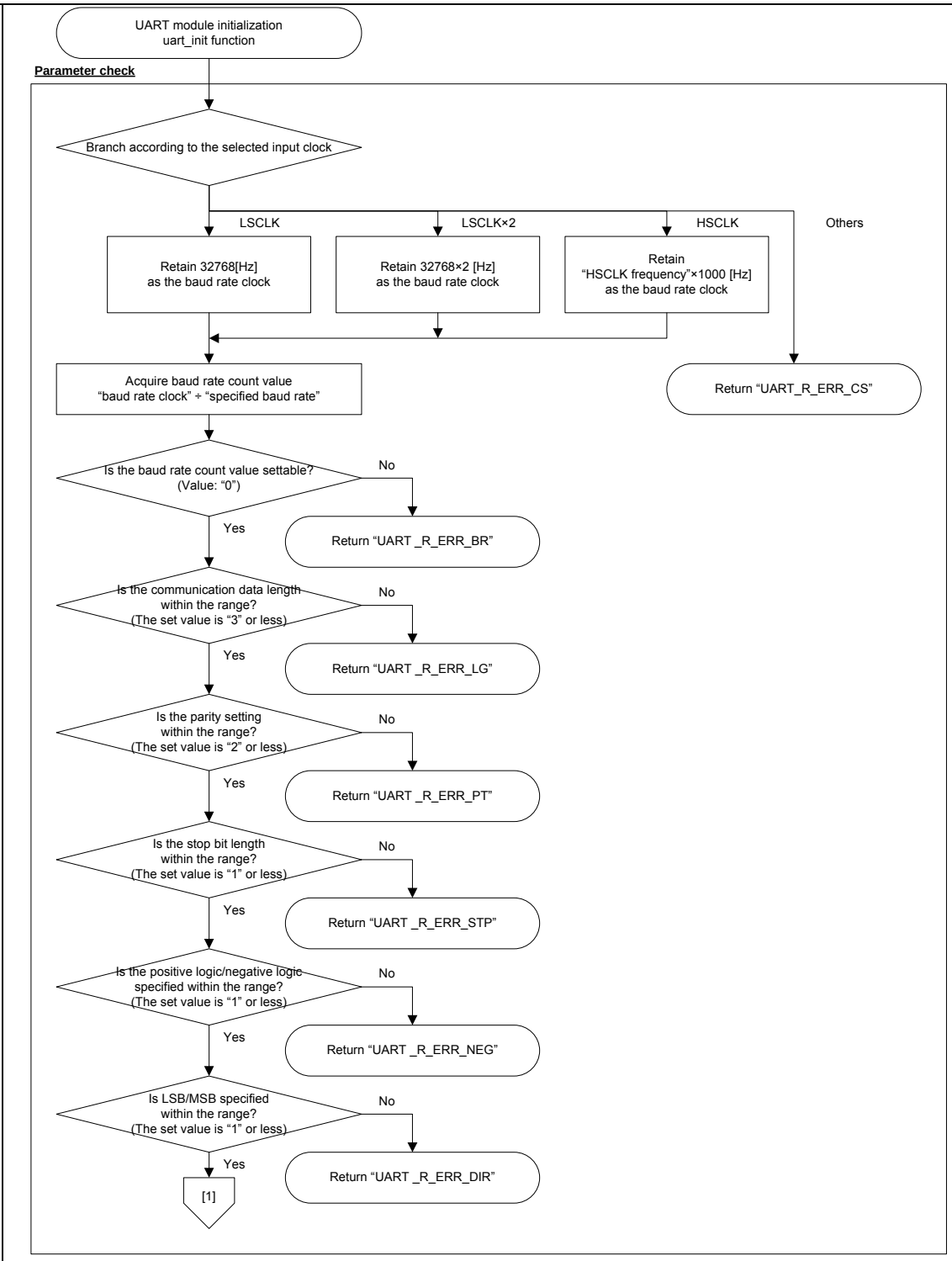
3.4.7. API Details

This section describes details of the UART module APIs.

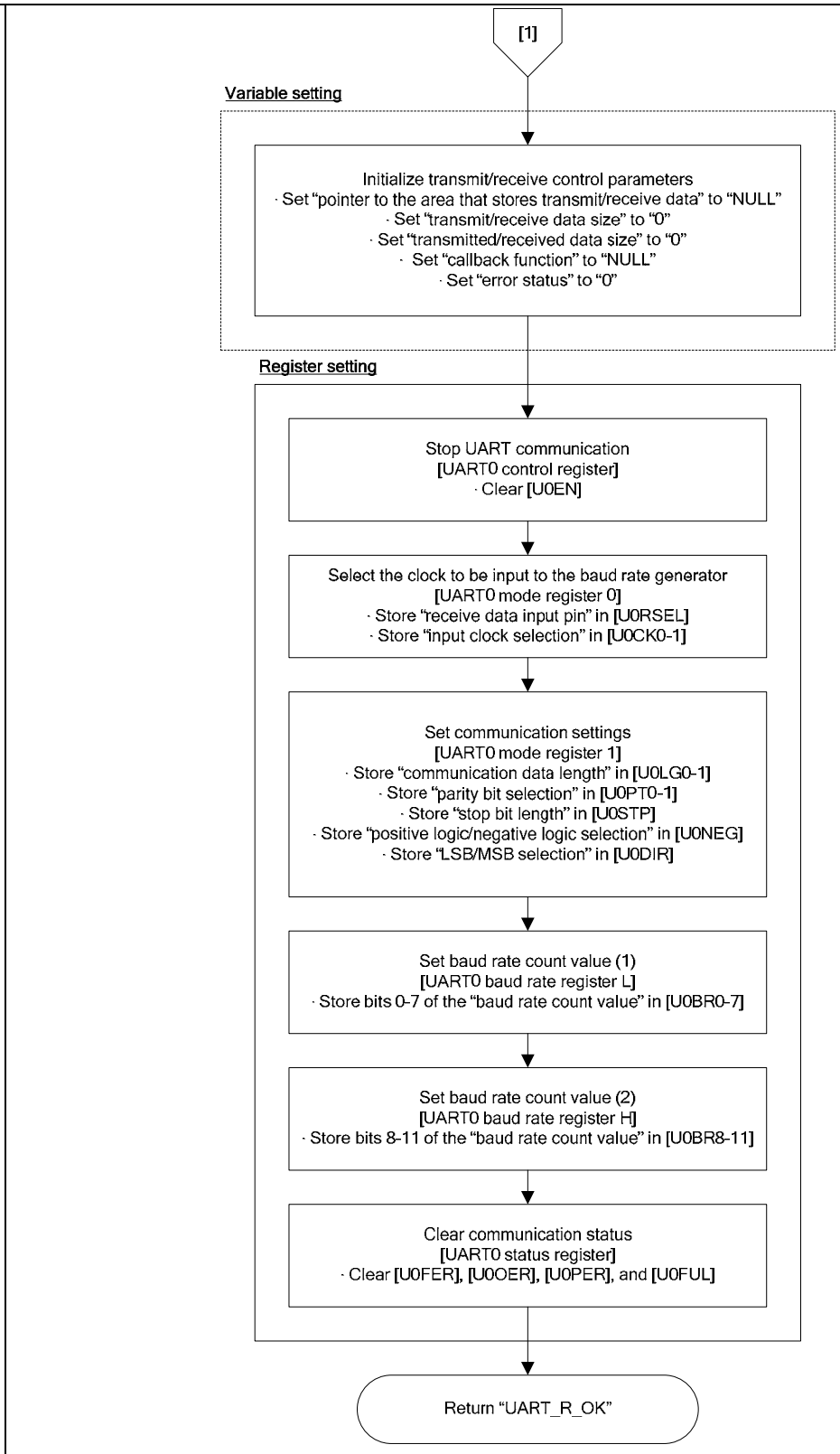
3.4.7.1. uart_init Function

This function initializes the UART of the MCU. It selects the clock to be input to the baud rate generator and sets the HSCLK frequency and communication settings (such as character length, parity, and baud rate).

Function name:	int uart_init(unsigned char cs, unsigned short kHz, tUartSetParam *prm)
Arguments:	unsigned char cs … Selection of the clock to be input to the baud rate generator LSCLK: UART_CS_LSCLK(=0) LSCLK×2: UART_CS_LSCLK2(=1) HSCLK: UART_CS_HSCLK(=2) unsigned short kHz … Frequency of HSCLK (* Referenced only when HSCLK is selected) tUartSetParam *prm … Setting parameter
Return values:	int Initializing succeeded: UART_R_OK(=0) The selected input clock is outside the range: UART_R_ERR_CS(=-1) The baud rate setting is outside the range: UART _R_ERR_BR(=-2) The communication data length is outside the range: UART _R_ERR_LG(=-3) The parity setting is outside the range: UART _R_ERR_PT(=-4) The stop bit length is outside the range: UART _R_ERR_STP(=-5) The positive logic/negative logic specified is outside the range: UART _R_ERR_NEG(=-6) LSB/MSB specified is outside the range: UART _R_ERR_DIR(=-7)
Processing:	See next page.



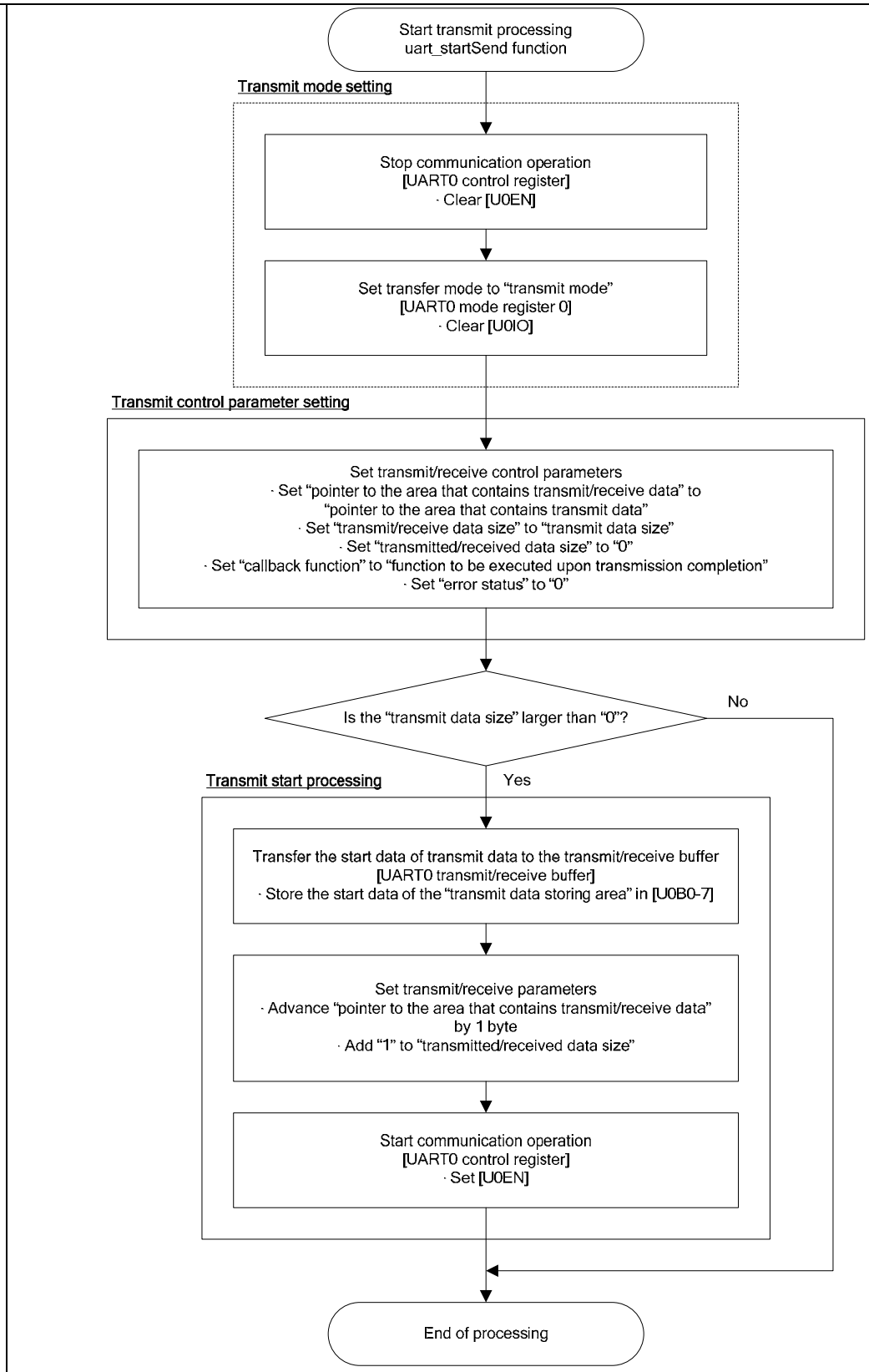
Continued on next page


[1]
Variable setting
Initialize transmit/receive control parameters
· Set "pointer to the area that stores transmit/receive data" to "NULL"
· Set "transmit/receive data size" to "0"
· Set "transmitted/received data size" to "0"
· Set "callback function" to "NULL"
· Set "error status" to "0"
Register setting
Stop UART communication
[UART0 control register]
· Clear [U0EN]
Select the clock to be input to the baud rate generator
[UART0 mode register 0]
· Store "receive data input pin" in [U0RSEL]
· Store "input clock selection" in [U0CK0-1]
Set communication settings
[UART0 mode register 1]
· Store "communication data length" in [U0LG0-1]
· Store "parity bit selection" in [U0PT0-1]
· Store "stop bit length" in [U0STP]
· Store "positive logic/negative logic selection" in [U0NEG]
· Store "LSB/MSB selection" in [U0DIR]
Set baud rate count value (1)
[UART0 baud rate register L]
· Store bits 0-7 of the "baud rate count value" in [U0BR0-7]
Set baud rate count value (2)
[UART0 baud rate register H]
· Store bits 8-11 of the "baud rate count value" in [U0BR8-11]
Clear communication status
[UART0 status register]
· Clear [U0FER], [U0OER], [U0PER], and [U0FUL]
Return "UART_R_OK"

3.4.7.2. uart_startSend Function

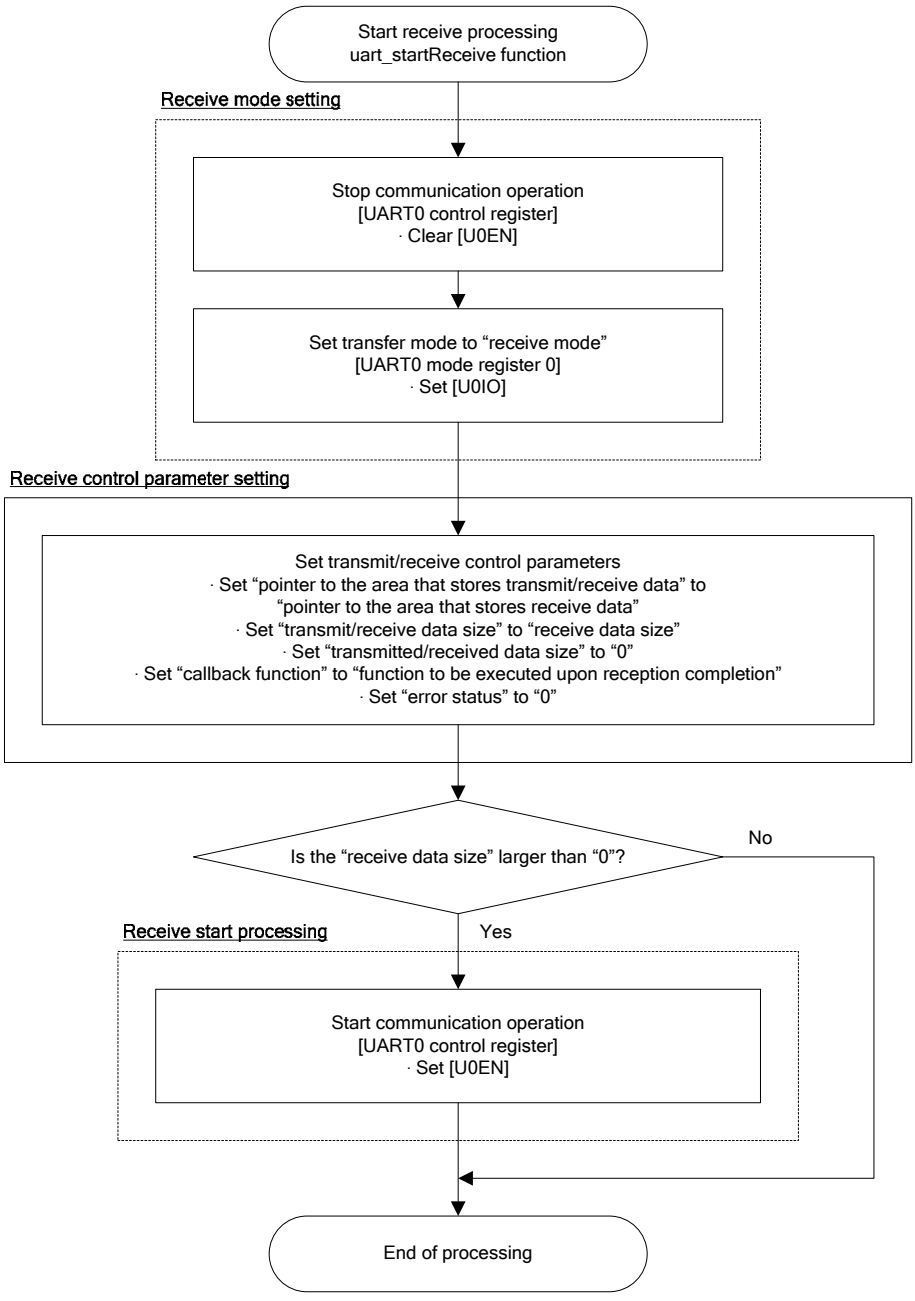
This function performs data transmission start processing. If this function is called during data transmission/reception, the transmission/reception being performed at the time stops and transmission processing newly starts.

Function name:	void uart_startSend(unsigned char *data, unsigned int size, tCallBack func)
Arguments:	unsigned char *data … Pointer to the area that contains transmit data unsigned int size …. Transmit data size (bytes) tCallBack func … Function to be executed upon transmission completion
Return values:	None
Processing:	See next page.


Start transmit processing
uart_startSend function
Transmit mode setting
Stop communication operation
[UART0 control register]
· Clear [U0EN]
Set transfer mode to "transmit mode"
[UART0 mode register 0]
· Clear [U0IO]
Transmit control parameter setting
Set transmit/receive control parameters
· Set "pointer to the area that contains transmit/receive data" to
"pointer to the area that contains transmit data"
· Set "transmit/receive data size" to "transmit data size"
· Set "transmitted/received data size" to "0"
· Set "callback function" to "function to be executed upon transmission completion"
· Set "error status" to "0"
Is the "transmit data size" larger than "0"?
No
Yes
Transmit start processing
Transfer the start data of transmit data to the transmit/receive buffer
[UART0 transmit/receive buffer]
· Store the start data of the "transmit data storing area" in [U0B0-7]
Set transmit/receive parameters
· Advance "pointer to the area that contains transmit/receive data"
by 1 byte
· Add "1" to "transmitted/received data size"
Start communication operation
[UART0 control register]
· Set [U0EN]
End of processing

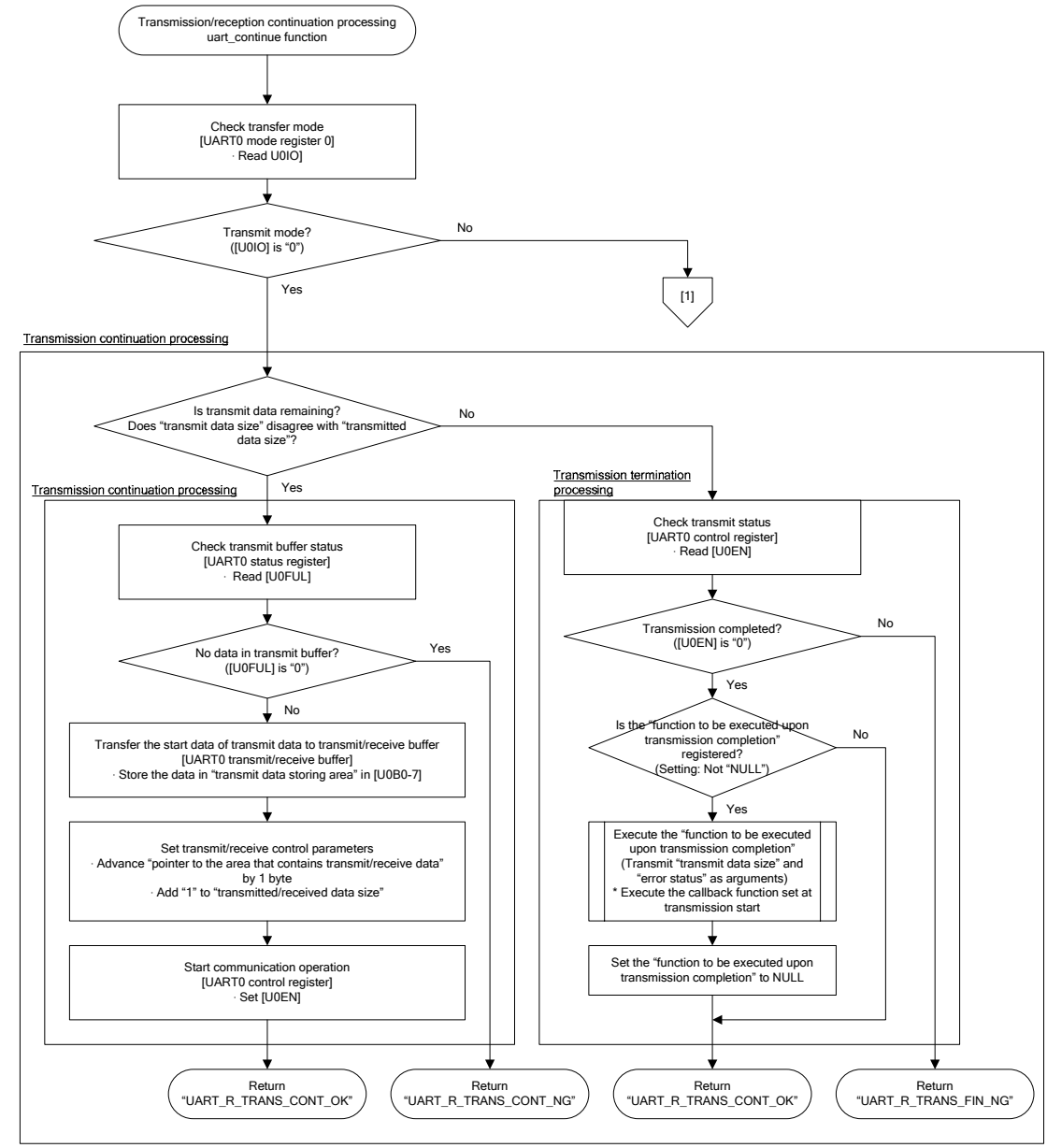
3.4.7.3. uart_startReceive Function

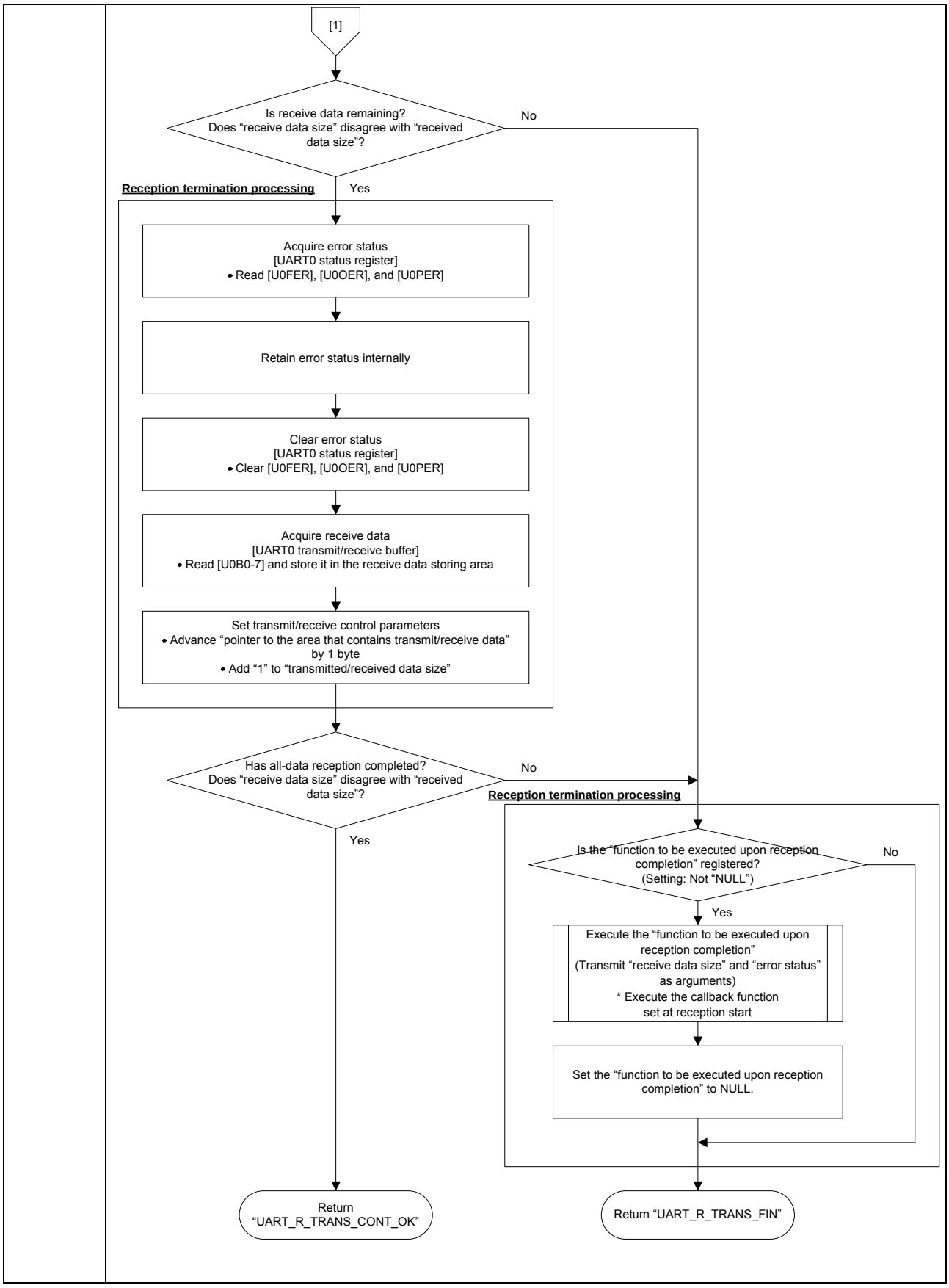
This function performs data reception start processing. If this function is called during data transmission/reception, the transmission/reception being performed at the time stops and reception processing newly starts.

Function name:	void uart_startReceive(unsigned char *data, unsigned int size, tCallBack func)
Arguments:	unsigned char *data … Pointer to the area that stores receive data unsigned int size … Receive data size (bytes) tCallBack func … Function to be executed upon reception completion
Return values:	None
Processing:	

3.4.7.4. uart_continue Function

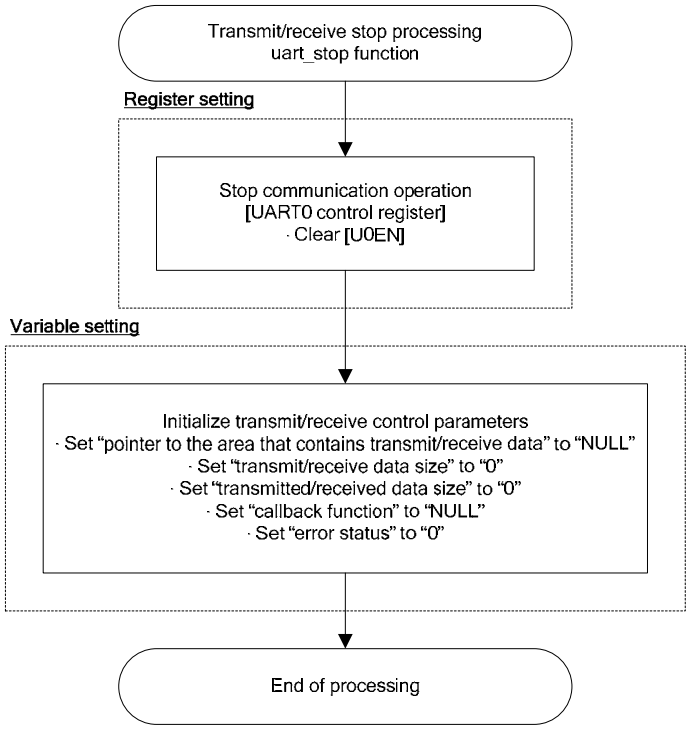
This function performs data transmission/reception continuation processing.

Function name:	int uart_continue(void)
Arguments:	None
Return values:	Int Transmit/receive processing terminated: UART_R_TRANS_FIN(=1) Transmit/receive processing continuing (transmission/reception succeeded): UART_R_TRANS_CONT_OK(=0) Transmit/receive processing continuing (transmission/reception failed): UART_R_TRANS_CONT_NG(=-1) Transmit/receive processing terminated (termination failed): UART_R_TRANS_FIN_NG(=-2)
Processing:	 Continued on next page


[1]
Is receive data remaining?
Does “receive data size” disagree with “received data size”?
No
Reception termination processing
Yes
Acquire error status
[UART0 status register]
• Read [U0FER], [U0OER], and [U0PER]
Retain error status internally
Clear error status
[UART0 status register]
• Clear [U0FER], [U0OER], and [U0PER]
Acquire receive data
[UART0 transmit/receive buffer]
• Read [U0B0-7] and store it in the receive data storing area
Set transmit/receive control parameters
• Advance “pointer to the area that contains transmit/receive data” by 1 byte
• Add “1” to “transmitted/received data size”
Has all-data reception completed?
Does “receive data size” disagree with “received data size”?
No
Reception termination processing
Yes
Is the “function to be executed upon reception completion” registered?
(Setting: Not “NULL”)
No
Yes
Execute the “function to be executed upon reception completion”
(Transmit “receive data size” and “error status” as arguments)
* Execute the callback function set at reception start
Set the “function to be executed upon reception completion” to NULL.
Return “UART_R_TRANS_CONT_OK”
Return “UART_R_TRANS_FIN”

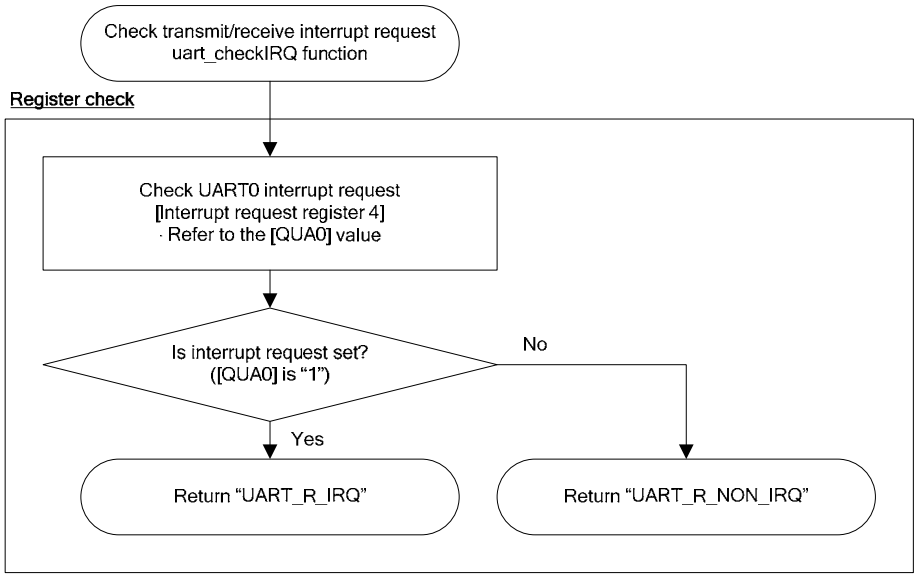
3.4.7.5. uart_stop Function

This function performs data transmission/reception stop processing.

Function name:	void uart_stop(void)
Arguments:	None
Return values:	None
Processing:	

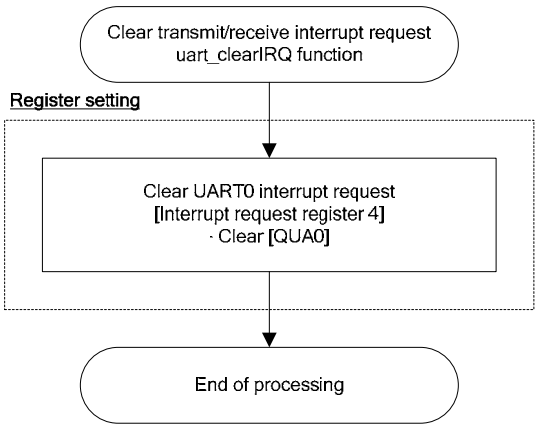
3.4.7.6. uart_checkIRQ Function

This function checks the QUA0 bit of the interrupt request register 4 to check whether a transmit/receive interrupt request is present.

Function name:	int uart_checkIRQ(void)
Arguments:	None
Return values:	int Transmit/receive interrupt requested: UART_R_IRQ No transmit/receive interrupt: UART_R_NON_IRQ
Processing:	

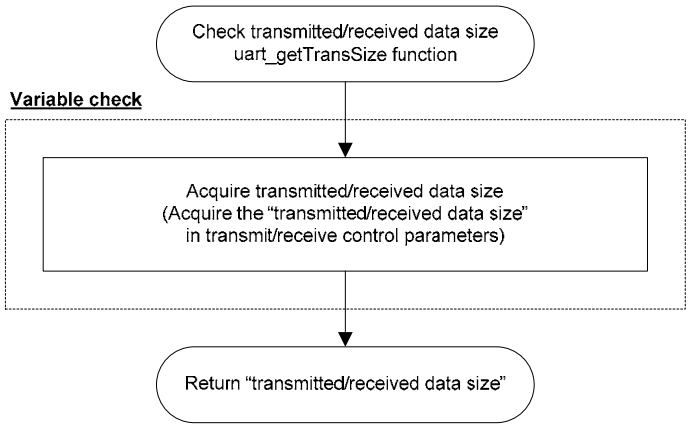
3.4.7.7. uart_clearIRQ Function

This function clears the transmit/receive interrupt request (QUA0 bit of the interrupt request register 4).

Function name:	void uart_clearIRQ(void)
Arguments:	None
Return values:	None
Processing:	

3.4.7.8. uart_getTransSize Function

This function acquires the transmitted/received data size. If the uart_stop function is called, the transmitted/received data size is cleared.

Function name:	unsigned int uart_getTransSize(void)
Arguments:	None
Return values:	unsigned int Transmitted/received data size (* Indicates the data size that was transferred from the execution of the uart_startSend function or the uart_startReceive function to the call made to this function)
Processing:	

3.4.7.9. uart_PortClear Function

This function initializes the port P43 (TXD0 : output) and the port P42 (RXD0 : input).

Function name:	void uart_PortClear(void)
Arguments:	None
Return values:	None
Processing:	This function initializes P43 and P42 to the following setting.

Port	Input/output mode	Input/output state	Function
P43	output	High-impedance output	Primary function : General-purpose input/output
P42	output	High-impedance output	Primary function : General-purpose input/output

3.4.7.10. uart_PortSet Function

This function sets the port P43 (TXD0 : output) and the port P42 (RXD0 : input) for UART data output and input port respectively. Please be sure to call this function after you call uart_init function and at least before you start UART transmit or receive operation by uart_startSend or uart_startReceive function.

Function name:	void uart_PortSet(void)
Arguments:	None
Return values:	None
Processing:	This function set P43 and P42 to the following setting.

Port	Input/output mode	Input/output state	Function
P43	output	CMOS output	Secondary function : UART0 output
P42	input	High-impedance input	Secondary function : UART0 input

3.5. UART Baud Rate Correction Module

A high-speed clock frequency is known to fluctuate at about ±15% depending on the temperature change in cases where the high-speed clock of the ML610Q431 is operating in RC oscillation mode. Also note that for the baud rate of UART, a communication error occurs if the baud rate clock frequency fluctuates exceeding ±2%. In other words, UART communication is expected to stop if there is a change in the ambient temperature.
The aim of the UART baud rate correction module is to make UART communication continue by measuring a high-speed clock frequency by use of the low-speed 32 kHz crystal oscillation clock and then correcting the UART baud rate timer value from the clock frequency obtained.

3.5.1. Overview of Functions

This module measures a high-speed clock frequency using Timers 2 and 3 and the 128Hz interrupt, calculates the setting value of the UART baud rate timer register and then corrects the UART baud rate timer value.

3.5.2. List of APIs

The following table lists the baud rate correction module APIs.

Table 3-13 Baud Rate Correction Module APIs

Function name	Description
adjustBaudrate_startCount function	Starts baud rate correction counter operation.
adjustBaudrate_checkFin function	Checks whether the baud rate correction counter operation is complete.
adjustBaudrate_getCount function	Obtains the count value of the baud rate correction counter.
adjustBaudrate_setBRT function	Sets the baud rate register.
adjustBaudrate_intCount function	Performs 128Hz interrupt processing for baud rate correction. Used in cases where the frequency measurement function is not provided.
adjustBaudrate_intCountTM3 function	Performs timer 3 interrupt processing for baud rate correction. Used in cases where the frequency measurement function is provided.

3.5.3. List of Constants

The following table lists the constants used in the baud rate correction module.

Table 3-14 List of Constants for Arguments

Constant name	Defined value	Description
ADJUSTBAUDRATE_NOT_USE_FM	0	The 16-bit frequency measurement function is not used.
ADJUSTBAUDRATE_USE_FM	1	The 16-bit frequency measurement function is used.
ADJUSTBAUDRATE_2400BPS	0	Specifies 2400 bps as the baud rate.
ADJUSTBAUDRATE_4800BPS	1	Specifies 4800 bps as the baud rate.
ADJUSTBAUDRATE_9600BPS	2	Specifies 9600 bps as the baud rate.
ADJUSTBAUDRATE_19200BPS	3	Specifies 19200 bps as the baud rate.
ADJUSTBAUDRATE_38400BPS	4	Specifies 38400 bps as the baud rate.

Table 3-15 List of Constants for Return Values

Constant name	Defined value	Description
ADJUSTBAUDRATE_R_OK	0	Processing succeeded.
ADJUSTBAUDRATE_R_NG	-1	Processing failed.
ADJUSTBAUDRATE_R_FIN	1	Baud rate correction terminated.
ADJUSTBAUDRATE_R_NOT_FIN	0	Baud rate correction is being done.

3.5.4. List of Variables

The following tables list the variables used in the baud rate correction module.

Table 3-16 List of Variables

Variable name	Initial value	Description
static unsigned short _cntTimer	0	This is a variable used to retain the difference in the 16-bit timer count value for each 128Hz interrupt. * Used only when the frequency measurement function is not provided.
static int _cnt128Hz	0	If the frequency measurement function is not provided: This is a variable (0 to 2) used to retain the remaining number of 128Hz interrupts required for baud rate correction. If the frequency measurement function is provided: Variable used to indicate the frequency measurement status (1: Measurement is being performed / 0: Measurement is stopped).
static unsigned long _cnt437c	0	This is a variable used to retain the 437C timer count value.

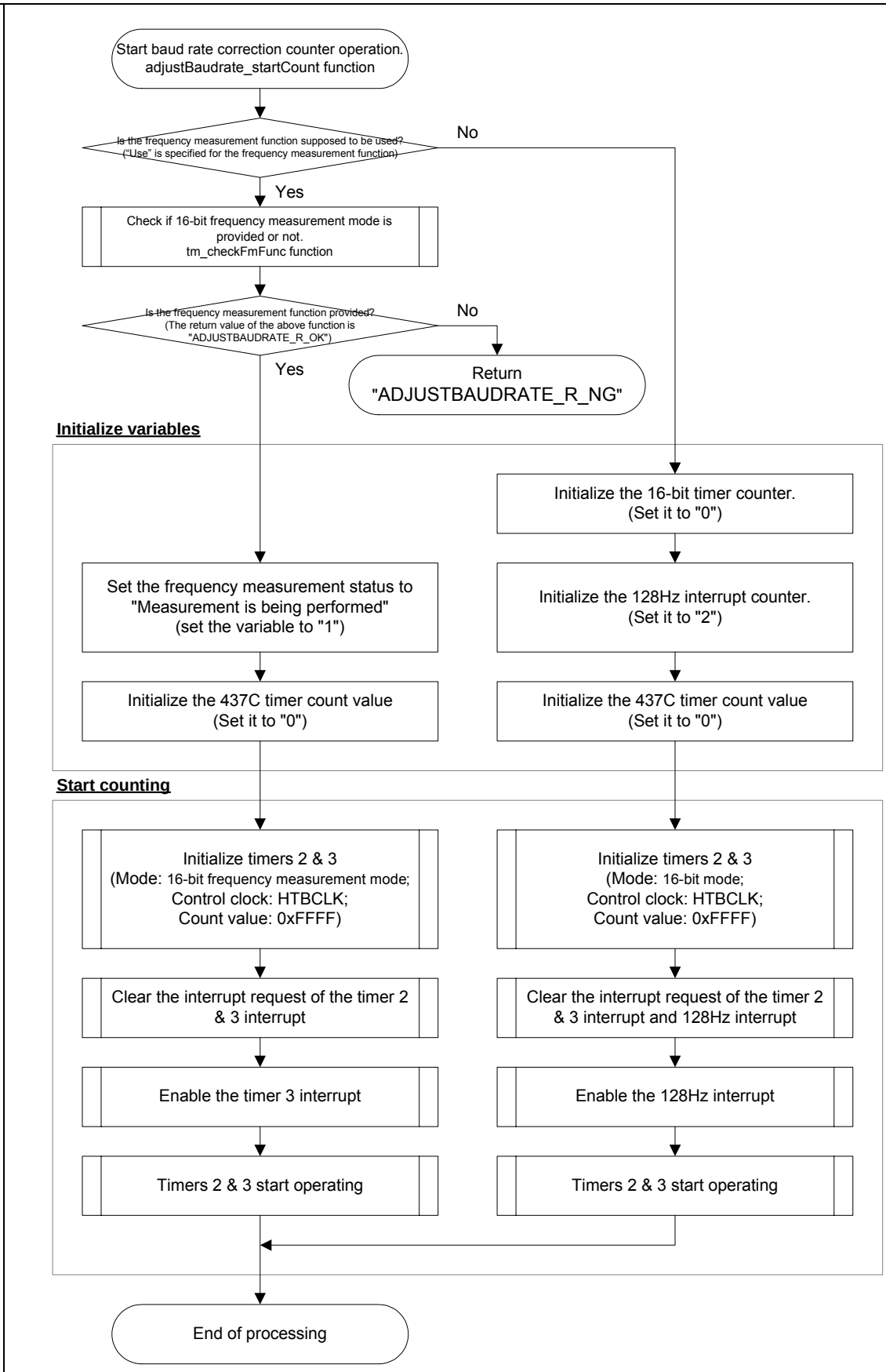
3.5.5. Details of APIs

This section describes details of the baud rate correction module APIs.

3.5.5.1. adjustBaudrate_startCount Function

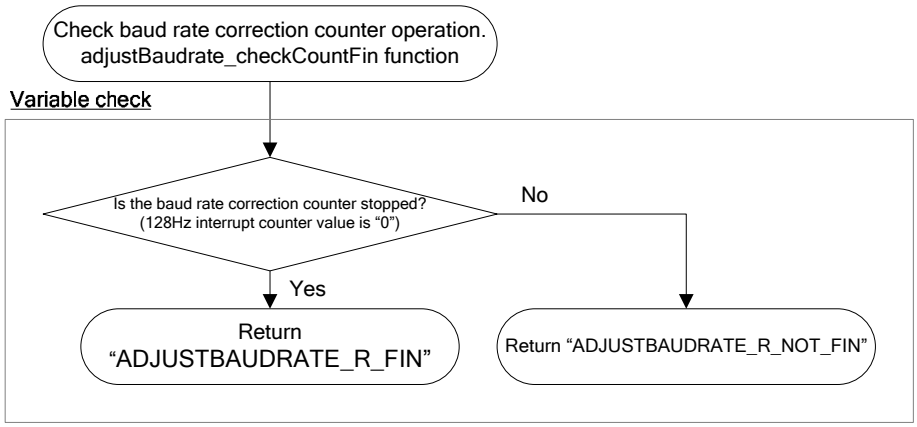
This function enables the 128Hz interrupt, activates the 16-bit timer, and starts baud rate counter operation.

Function name:	int adjustBaudrate_startCount(unsigned char useFm)
Arguments:	unsigned char useFm … Specify whether or not to use the 16-bit frequency measurement function. Do not use the 16-bit frequency measurement function: ADJUSTBAUDRATE_NOT_USE_FM(=0) Use the 16-bit frequency measurement function: ADJUSTBAUDRATE_USE_FM(=1)
Return values:	Int The start processing was successful: ADJUSTBAUDRATE_R_OK(=0) The 16-bit frequency measurement function is not provided: ADJUSTBAUDRATE_R_NG(=-1)
Processing:	See next page.


Start baud rate correction counter operation.
adjustBaudrate_startCount function
Is the frequency measurement function supposed to be used?
("Use" is specified for the frequency measurement function)
No
Yes
Check if 16-bit frequency measurement mode is provided or not.
tm_checkFmFunc function
Is the frequency measurement function provided?
(The return value of the above function is "ADJUSTBAUDRATE_R_OK")
No
Yes
Return "ADJUSTBAUDRATE_R_NG"
Initialize variables
Initialize the 16-bit timer counter.
(Set it to "0")
Set the frequency measurement status to "Measurement is being performed"
(set the variable to "1")
Initialize the 128Hz interrupt counter.
(Set it to "2")
Initialize the 437C timer count value
(Set it to "0")
Initialize the 437C timer count value
(Set it to "0")
Start counting
Initialize timers 2 & 3
(Mode: 16-bit frequency measurement mode;
Control clock: HTBCLK;
Count value: 0xFFFF)
Initialize timers 2 & 3
(Mode: 16-bit mode;
Control clock: HTBCLK;
Count value: 0xFFFF)
Clear the interrupt request of the timer 2 & 3 interrupt
Clear the interrupt request of the timer 2 & 3 interrupt and 128Hz interrupt
Enable the timer 3 interrupt
Enable the 128Hz interrupt
Timers 2 & 3 start operating
Timers 2 & 3 start operating
End of processing

3.5.5.2. adjustBaudrate_checkCountFin Function

This function checks whether the operation of the baud rate correction counter has been completed.

Function name:	int adjustBaudrate_checkCountFin(void)
Arguments:	None
Return values:	int Baud rate correction completed: ADJUSTBAUDRATE_R_FIN (=1) Baud rate correction in progress: ADJUSTBAUDRATE_R_NOT_FIN (=0)
Processing:	

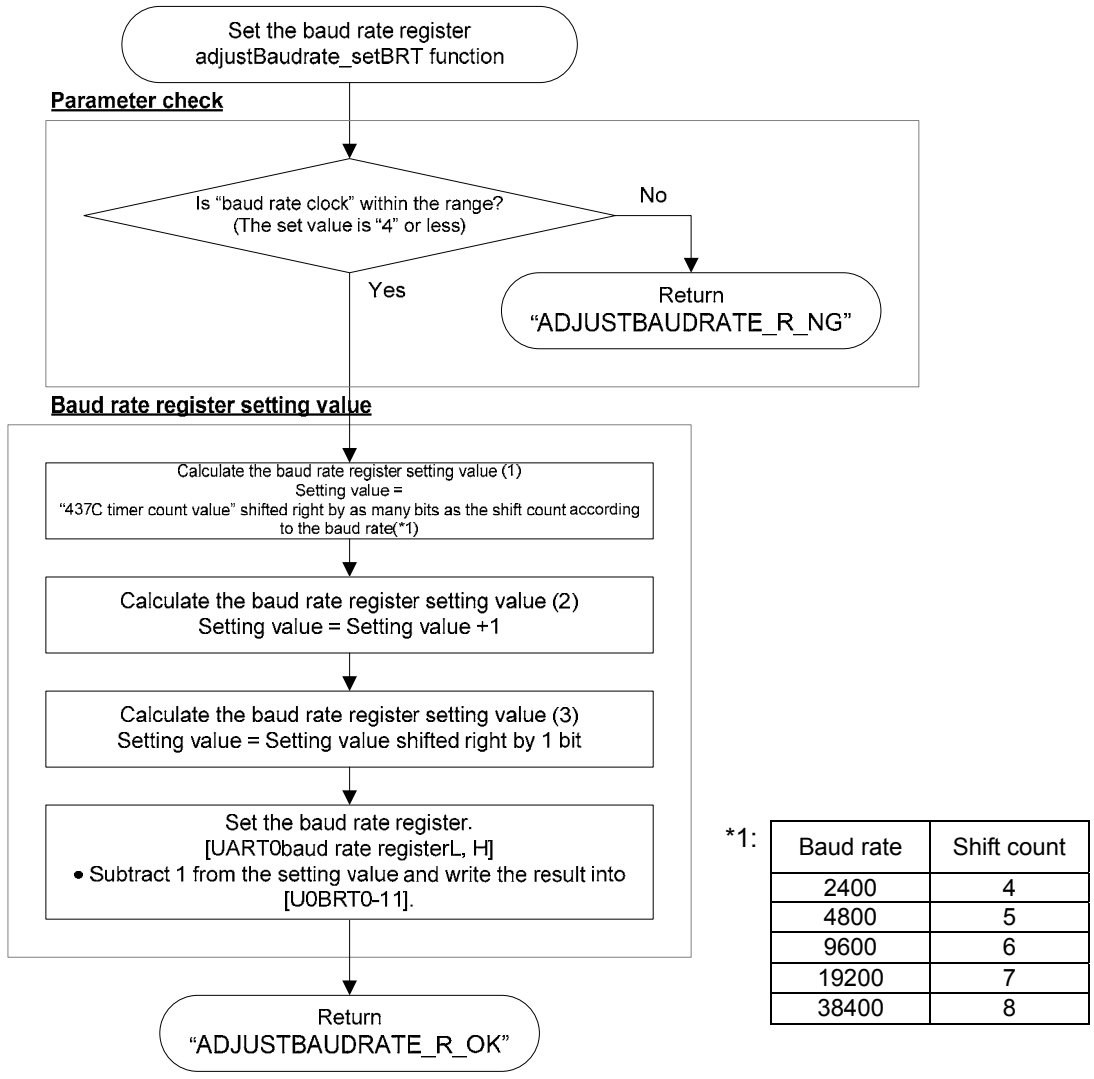
3.5.5.3. adjustBaudrate_getCount Function

This function acquires the count value of the baud rate correction counter.

Function name:	unsigned long adjustBaudrate_getCount(void)
Arguments:	void
Return values:	unsigned long Count value of the baud rate correction counter
Processing:	Returns the value of the variable "_cnt437c". ⇒ The value that matches the count value when the 437C timer is used is retained in this variable.

3.5.5.4. adjustBaudrate_setBRT Function

This function calculates the baud rate count value from the baud rate value and 437C timer count value and sets it in the baud rate register. The calculation result will be rounded.

Function name:	int adjustBaudrate_setBRT (unsigned char baudrateType, unsigned long cnt437c)
Arguments:	unsigned char baudrateType … Baud rate clock 2400bps: ADJUSTBAUDRATE_2400BPS(=0) 4800bps: ADJUSTBAUDRATE_4800BPS(=1) 9600bps: ADJUSTBAUDRATE_9600BPS(=2) 19200bps: ADJUSTBAUDRATE_19200BPS(=3) 38400bps: ADJUSTBAUDRATE_38400BPS(=4) unsigned long cnt437c … 437C timer count value
Return values:	int Setting succeeded: ADJUSTBAUDRATE_R_OK(=0) The parameter is outside the range: ADJUSTBAUDRATE_R_NG(=-1)
Processing:	

*1:

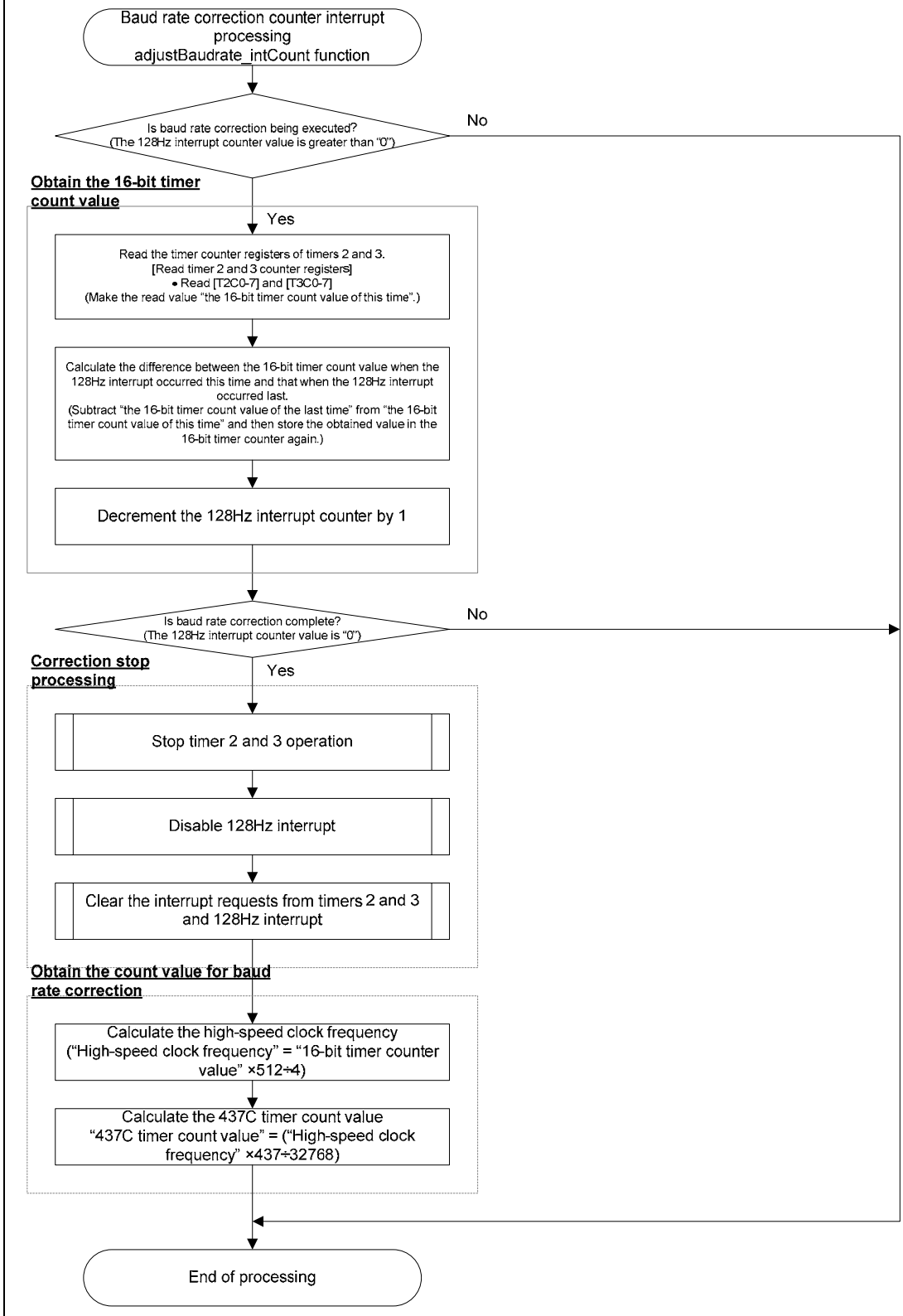
Baud rate	Shift count
2400	4
4800	5
9600	6
19200	7
38400	8

3.5.5.5. adjustBaudrate_intCount Function

This function reads the count value of the 16-bit timer at 128-Hz intervals and calculates the count value equivalent to the 437C count value from the difference between the 1st and 2nd reads. The function must be set in the 128Hz interrupt vector in order to execute it at the timing of the 128Hz interrupt.

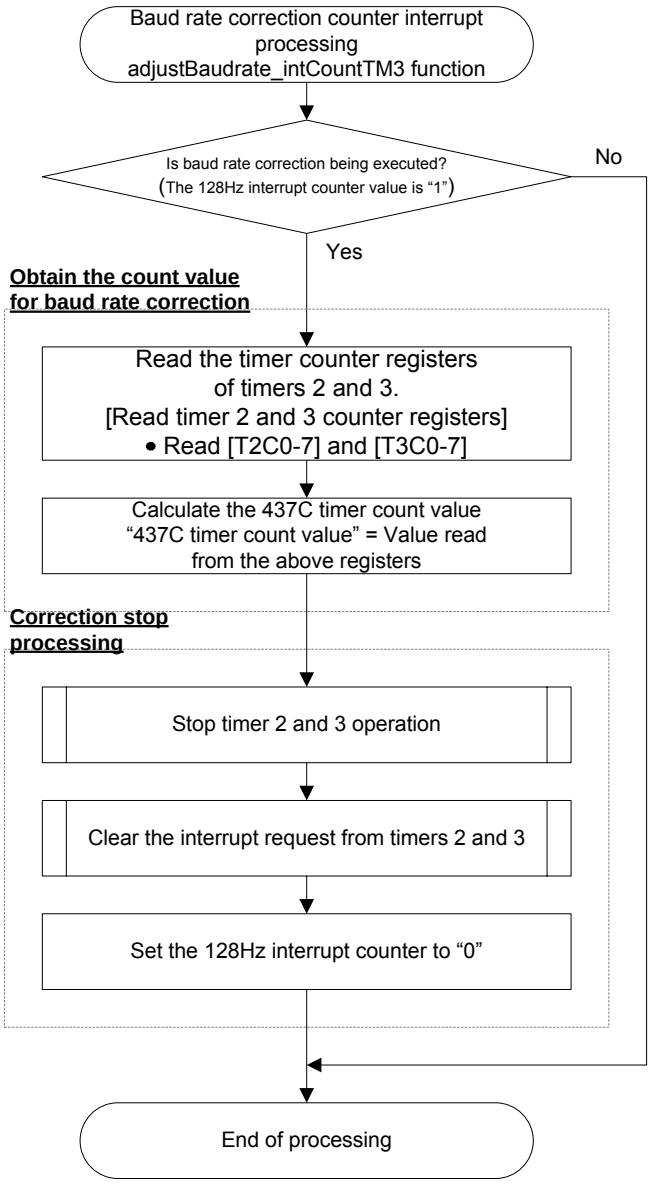
Function name:	void adjustBaudrate_intCount(void)
Arguments:	None
Return values:	None

Processing:


Baud rate correction counter interrupt processing
adjustBaudrate_intCount function
Is baud rate correction being executed?
(The 128Hz interrupt counter value is greater than "0")
No
Yes
Obtain the 16-bit timer count value
Read the timer counter registers of timers 2 and 3.
[Read timer 2 and 3 counter registers]
• Read [T2C0-7] and [T3C0-7]
(Make the read value "the 16-bit timer count value of this time".)
Calculate the difference between the 16-bit timer count value when the 128Hz interrupt occurred this time and that when the 128Hz interrupt occurred last.
(Subtract "the 16-bit timer count value of the last time" from "the 16-bit timer count value of this time" and then store the obtained value in the 16-bit timer counter again.)
Decrement the 128Hz interrupt counter by 1
Is baud rate correction complete?
(The 128Hz interrupt counter value is "0")
No
Yes
Correction stop processing
Stop timer 2 and 3 operation
Disable 128Hz interrupt
Clear the interrupt requests from timers 2 and 3 and 128Hz interrupt
Obtain the count value for baud rate correction
Calculate the high-speed clock frequency
("High-speed clock frequency" = "16-bit timer counter value" ×512÷4)
Calculate the 437C timer count value
"437C timer count value" = ("High-speed clock frequency" ×437÷32768)
End of processing

3.5.5.6. adjustBaudrate_intCountTM3 Function

This function reads the count value of the 16-bit timer at the timer 3 interrupt generation timing and calculates the 437C count value. The function must be set in the timer 3 interrupt vector in order to execute it at the timing of the timer 3 interrupt.

Function name:	void adjustBaudrate_intCountTM3(void)
Arguments:	None
Return values:	None
Processing:	

3.6. I2C Module

3.6.1. Overview of Functions

The I2C module controls the I2C bus interface (master) of the MCU. To be specific, it can control initialization (communication speed setting and use/nonuse setting of clock synchronization), data transmission/reception start/stop, continuation of data transmission/reception, obtaining and clearing of I2C interrupt requests, and acquisition of transmit/receive size.
When specified, callback functions can be executed upon completion of transmission/reception. Use the i2c_startSend function or i2c_startReceive function to specify callback functions.

3.6.2. List of APIs

The following table lists the I2C module APIs.

Table 3-17 I2C Module APIs

Function name	Description
i2c_init function	Selects the communication speed and the use/nonuse of clock synchronization.
i2c_startSend function	Executes address and data transmission start processing.
i2c_startReceive function	Executes address and data reception start processing.
i2c_continue function	Executes address and data transmission/reception continuation processing.
i2c_stop function	Executes transmission/reception stop processing.
i2c_checkIRQ function	Checks the presence or absence of a transmit/receive interrupt request.
i2c_clearIRQ function	Clears transmit/receive interrupt requests.
i2c_getTransSize function	Checks the size of the data transmitted/received.

3.6.3. Callback Function

This section describes the callback function specified by the i2c_startSend function or i2c_startReceive function.
The callback function is called upon completion of transmission/reception or upon occurrence of an error from the I2C module. Describe required processing within the callback function.

Definition of the callback function

```
typedef void (*cbfI2c)( unsigned int size, unsigned char errStat )
```

<Example of definition>

Function name:	void i2c_callback(unsigned int size, unsigned char errStat)
Arguments:	unsigned int size … Size of transmitted/received data unsigned char errStat … Transmission/reception result I2C_R_OK(=0): Transmission/reception ended normally I2C_ERR_ACR(=1): Received acknowledgment data “1” I2C_ERR_SEND_ERR(=2): Transmission error
Return values:	None

3.6.4. List of Constants

The following tables list the constants used in the I2C module.

Table 3-18 List of Constants for Arguments

Constant name	Defined value	Description
I2C_MOD_STD	0	Specifies standard mode (100 kHz) for communication speed.
I2C_MOD_FST	1	Specifies fast mode (400 kHz) for communication speed.
I2C_SYN_OFF	0	Specifies the nonuse of the clock synchronization function.
I2C_SYN_ON	1	Specifies the use of the clock synchronization function.

Table 3-19 List of Constants for Return Values

Constant name	Defined value	Description
I2C_R_OK	0	Processing succeeded.
I2C_R_ERR_SYN	-1	The specified value for the clock synchronization function is outside the range.
I2C_R_ERR_MODE	-2	The specified communication speed was unsettable.
I2C_R_ERR_FREQ	-3	The high-speed clock frequency is outside the range.
I2C_R_TRANS_START_OK	0	Transmit/receive start processing terminated.
I2C_R_BUS_BUSY	-1	I2C bus busy state
I2C_R_TRANS_FIN	1	Transmit/receive processing terminated.
I2C_R_TRANS_CONT_OK	0	The transmit/receive processing is going on (transmission succeeded)
I2C_R_IRQ	1	Transmit/receive complete interrupt request is present.
I2C_R_NON_IRQ	0	Transmit/receive complete interrupt request is absent.

3.6.5. Structure

This section describes the structures used in the I2C module.

■ I2C transmit/receive control parameters

```
typedef struct {
    unsigned char       mode        // Transmit/receive mode    (0: Transmission, 1: Reception)
    unsigned char *     addr;       // Pointer to the area that contains address data
    unsigned int        addr_size;  // Address size
    unsigned char *     data;       // Pointer to the area that contains transmit/receive data
    unsigned int        data_size;  // Transmit/receive data size
    unsigned int        cnt;        // Size of transmitted/received data
    tCallBack           callBack    // Callback function
    unsigned char       errStat     // Error status
    unsigned char       status      // Transmit/receive status
 tI2cCtrlParam;
```

3.6.6. List of Variables

The following table lists the variables used in the I2C module.

Variable name	Initial value	Description
static tI2cCtrlParam_gsCtrlParam	mode: 0 addr: NULL addr_size: 0 data: NULL data_size: 0 cnt: 0 callBack: NULL errStat: 0 status: 0	Variable to manage the data for I2C transmission/reception

3.6.7. Details of APIs

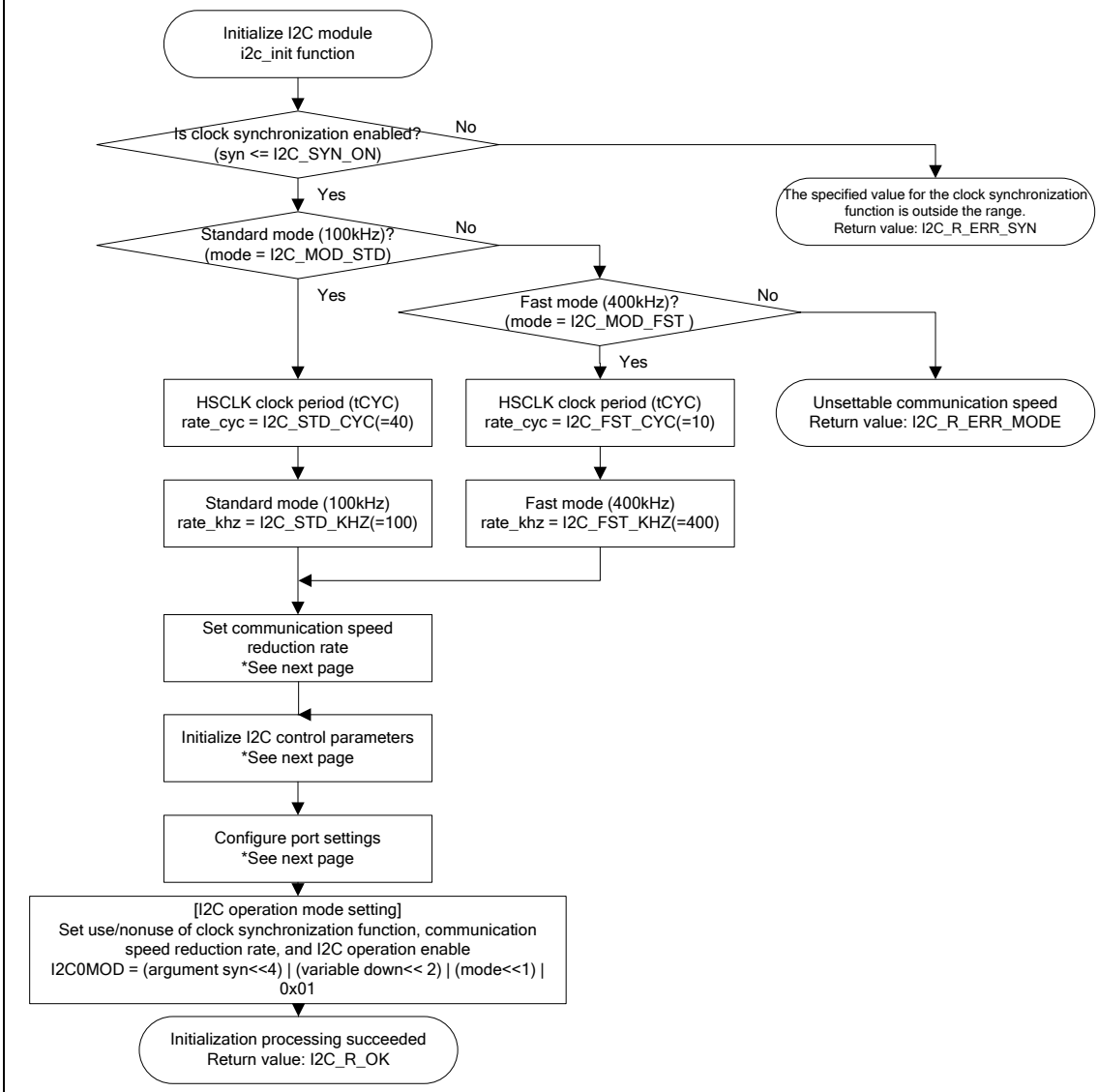
This section describes details of the I2C module APIs.

3.6.7.1. i2c_init Function

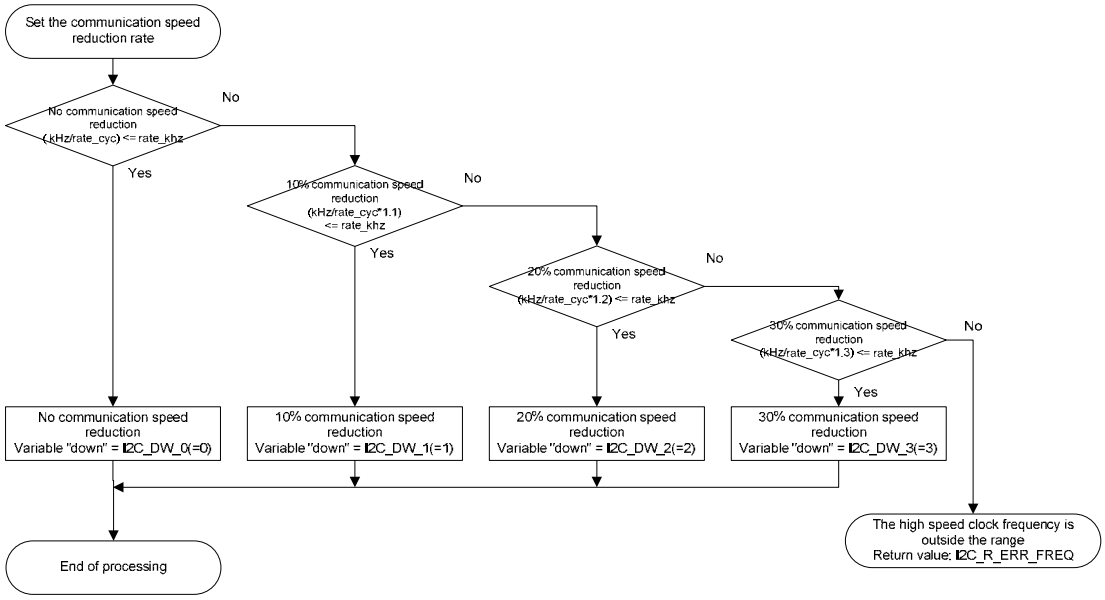
This function initializes the I2C bus interface of the MCU. In the initialization, the function selects communication speed and the use/nonuse of the clock synchronization function.

Function name:	int i2c_init(unsigned char mode, unsigned short kHz, unsigned char syn)
Arguments:	unsigned char mode … Selection of communication speed Standard mode (100kHz): I2C_MOD_STD (=0) Fast mode (400kHz): I2C_MOD_FST (=1) unsigned short kHz … Frequency of HSCLK [kHz] unsigned char syn …Use or nonuse of the clock synchronization function Do not use the clock synchronization: I2C_SYN_OFF (=0) Use the clock synchronization: I2C_SYN_ON (=1)
Return values:	int Initialization succeeded: I2C_R_OK(=0) The specified value for the clock synchronization function is outside the range: I2C_R_ERR_SYN (=-1) The specified communication speed was unsettable: I2C_R_ERR_MODE (=-2) The selected high-speed clock is outside the range: I2C_R_ERR_FREQ (=-3)

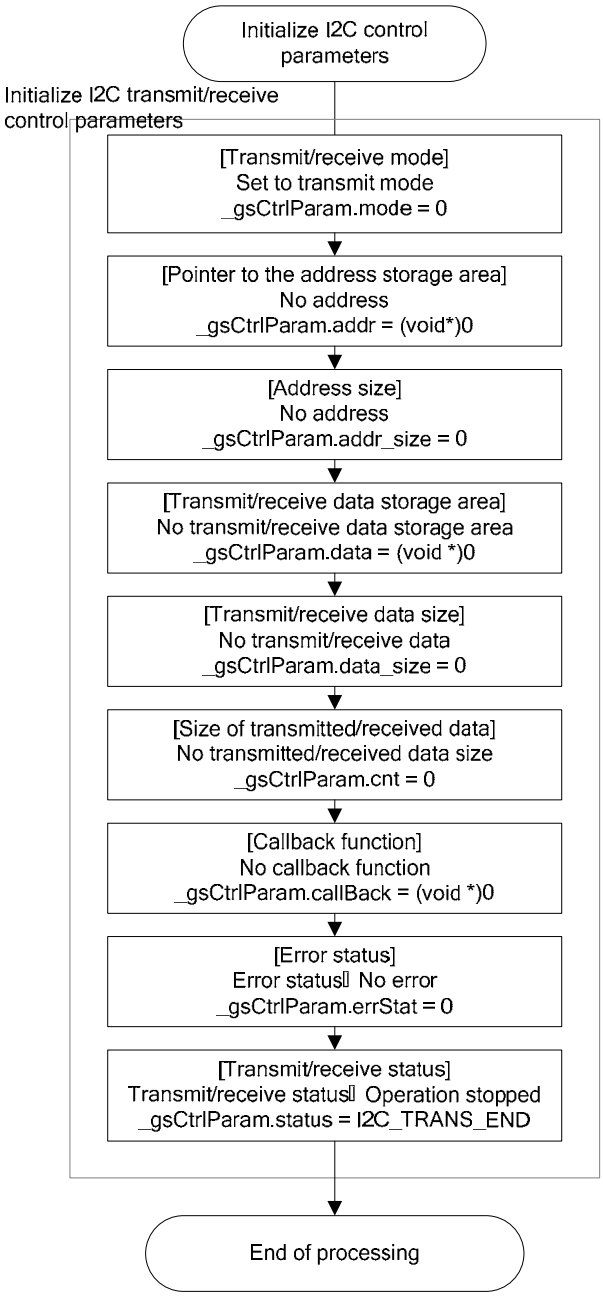
Processing: (1/4)



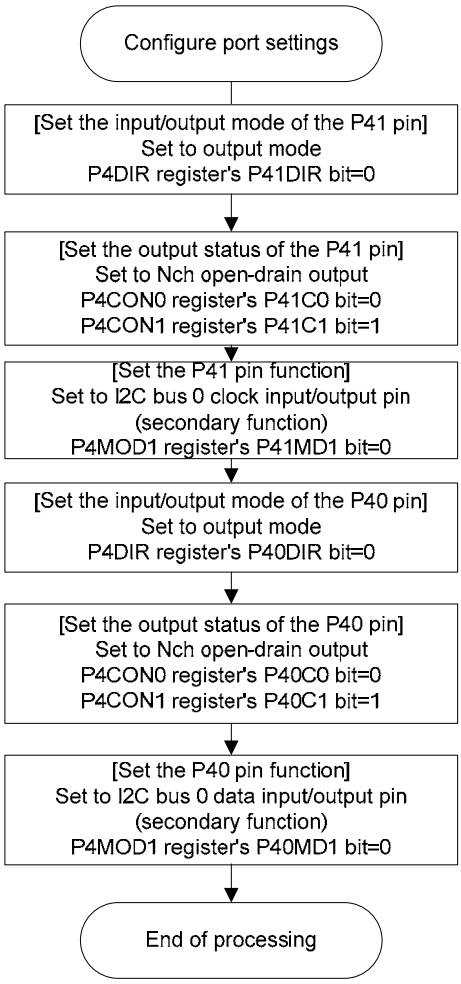
Processing: (2/4)



Processing: (3/4)



Processing: (4/4)

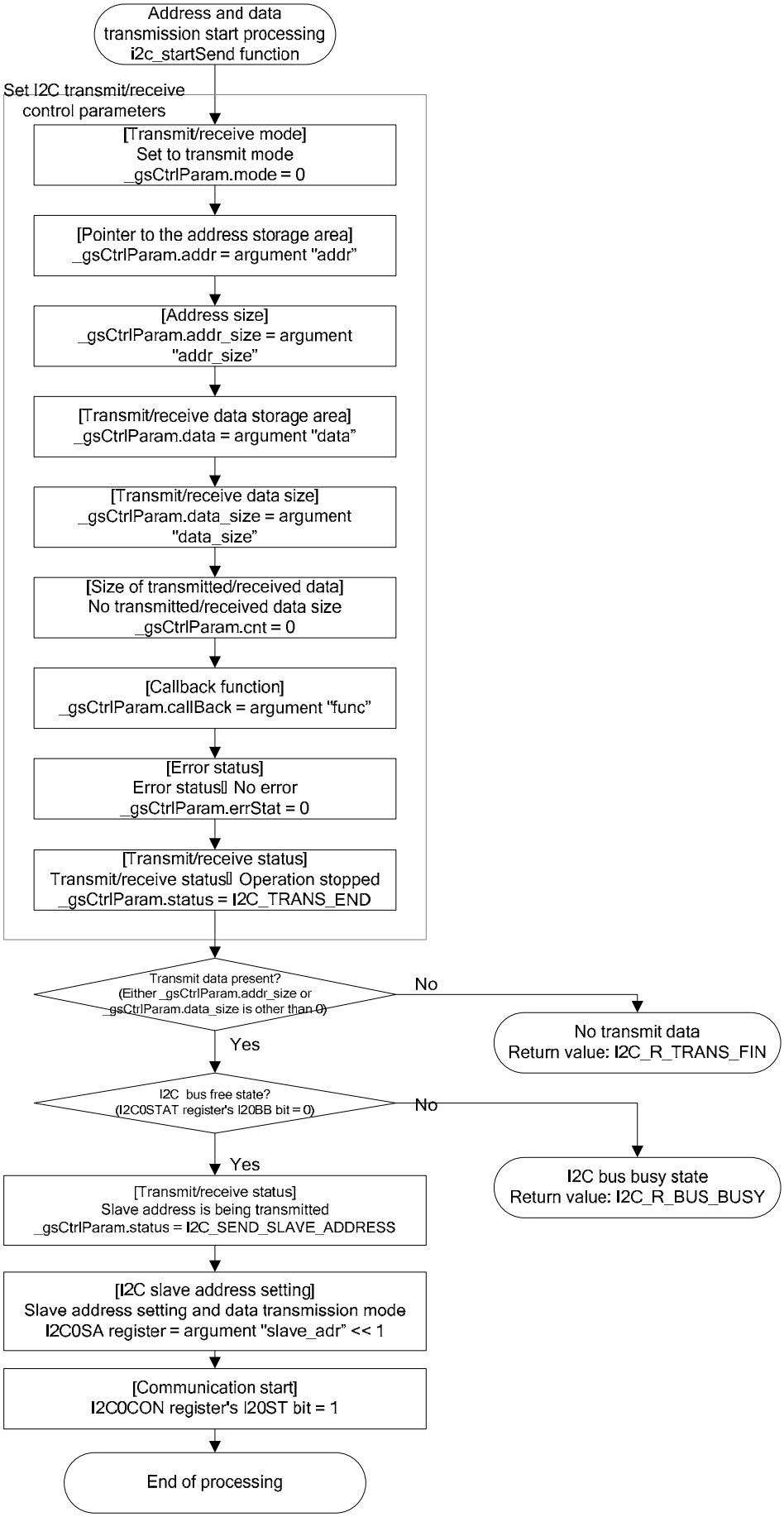


3.6.7.2. i2c_startSend Function

This function starts and stops transmission of address and data. If this function is called during data transmission/reception, I2C_R_BUS_BUSY (=-1) will be returned, terminating the transmission/reception.

Function name:	int i2c_startSend(unsigned char slave_adr, unsigned char *addr, unsigned int addr_size, unsigned char *data, unsigned int data_size, cbfI2c func)
Arguments:	unsigned char slave_adr …. Address of the transmission destination slave device unsigned char *addr … Pointer to the area that contains address and data unsigned int addr_size …. Size of address and data to be transmitted (byte units) unsigned char *data … Pointer to the area that contains transmit data unsigned int data_size …. Transmit data sizes (byte units) tCallBack func … Function to be executed upon completion of transmission
Return values:	int Transmit/receive start processing ended: I2C_R_TRANS_START_OK (=0) I2C bus busy state: I2C_R_BUS_BUSY (=-1) No transmit data: I2C_R_TRANS_FIN(=-2)

Processing:

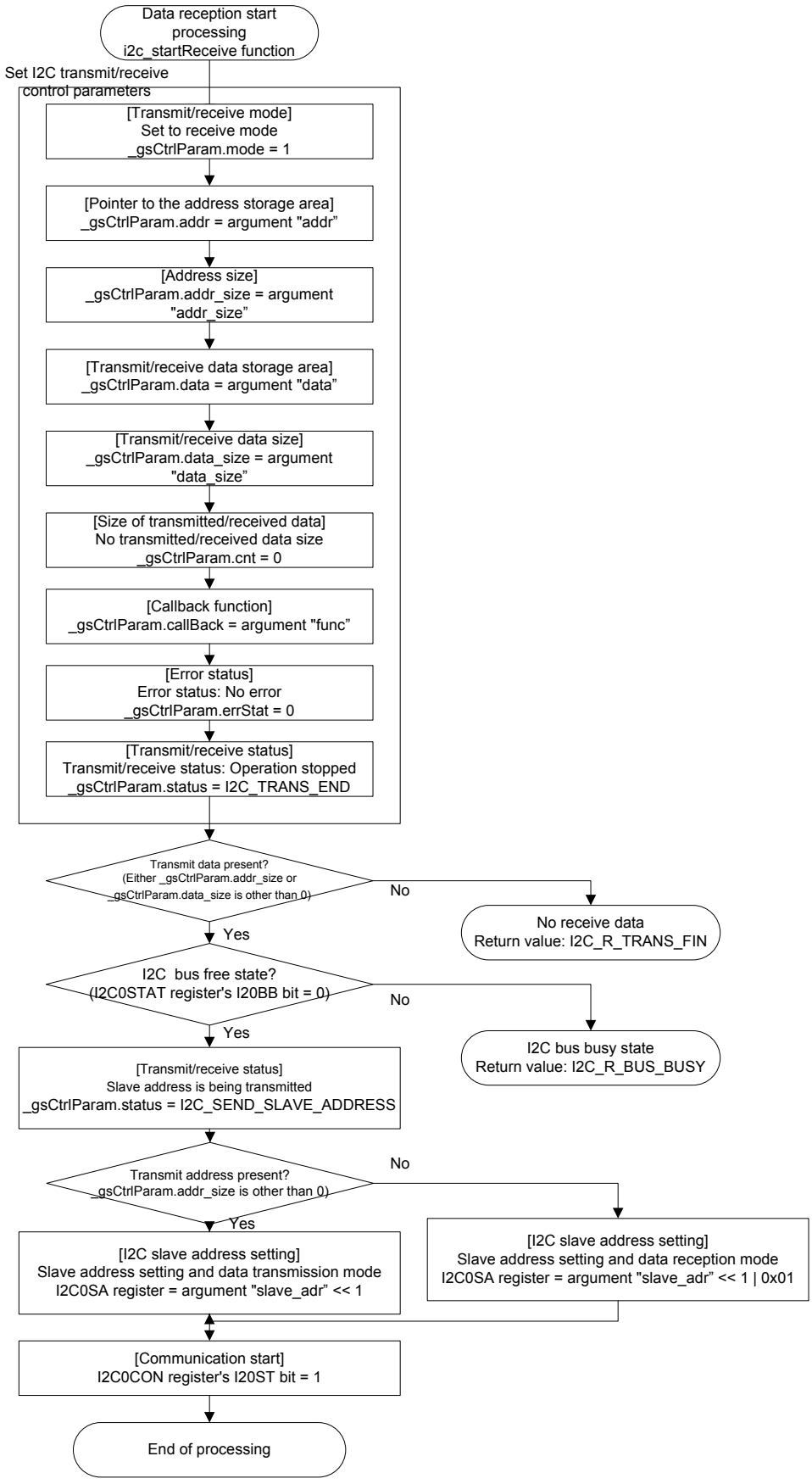

Address and data transmission start processing i2c_startSend function
Set I2C transmit/receive control parameters
[Transmit/receive mode] Set to transmit mode _gsCtrlParam.mode = 0
[Pointer to the address storage area] _gsCtrlParam.addr = argument "addr"
[Address size] _gsCtrlParam.addr_size = argument "addr_size"
[Transmit/receive data storage area] _gsCtrlParam.data = argument "data"
[Transmit/receive data size] _gsCtrlParam.data_size = argument "data_size"
[Size of transmitted/received data] No transmitted/received data size _gsCtrlParam.cnt = 0
[Callback function] _gsCtrlParam.callBack = argument "func"
[Error status] Error status No error _gsCtrlParam.errStat = 0
[Transmit/receive status] Transmit/receive status Operation stopped _gsCtrlParam.status = I2C_TRANS_END
Transmit data present? (Either _gsCtrlParam.addr_size or _gsCtrlParam.data_size is other than 0)
No
No transmit data Return value: I2C_R_TRANS_FIN
Yes
I2C bus free state? (I2C0STAT register's I20BB bit = 0)
No
I2C bus busy state Return value: I2C_R_BUS_BUSY
Yes
[Transmit/receive status] Slave address is being transmitted _gsCtrlParam.status = I2C_SEND_SLAVE_ADDRESS
[I2C slave address setting] Slave address setting and data transmission mode I2C0SA register = argument "slave_adr" << 1
[Communication start] I2C0CON register's I20ST bit = 1
End of processing

3.6.7.3. i2c_startReceive Function

This function starts data reception. If this function is called during data transmission/reception, I2C_R_BUS_BUSY (=-1) will be returned, terminating the transmission/reception.

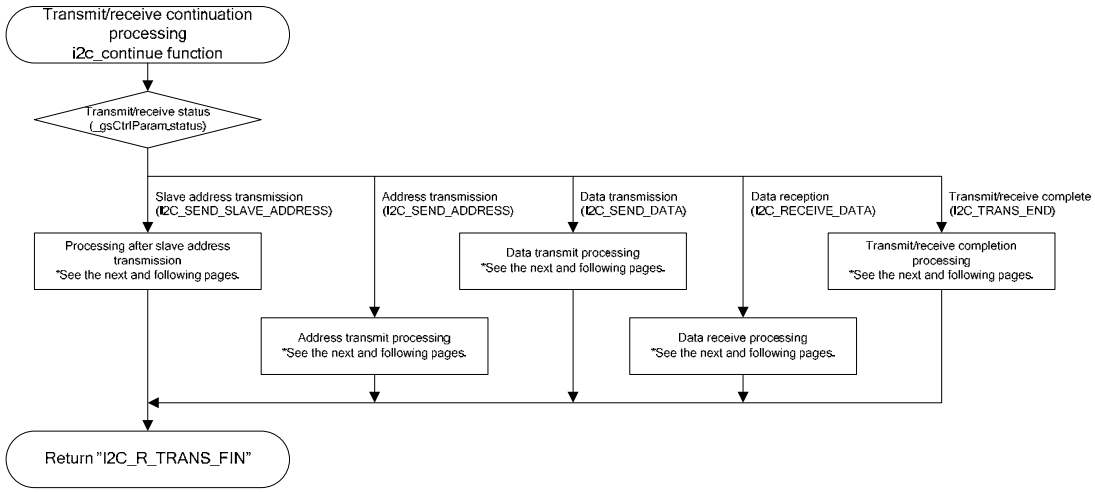
Function name:	int i2c_startReceive(unsigned char slave_adr, unsigned char *addr, unsigned int addr_size, unsigned char *data, unsigned int data_size, cbfI2c func)
Arguments:	unsigned char slave_adr …. Address of the receiving slave device unsigned char *addr … Pointer to the area that contains the receive start address unsigned int addr_size …. Size of address and data (byte units) unsigned char *data … Pointer to the area that stores receive data unsigned int data_size …. Receive data sizes (byte units) tCallBack func … Function to be executed upon completion of reception
Return values:	int Transmit/receive start processing ended: I2C_R_TRANS_START_OK (=0) I2C bus busy state: I2C_R_BUS_BUSY (=-1) No transmit data: I2C_R_TRANS_FIN(=-2)

Processing:

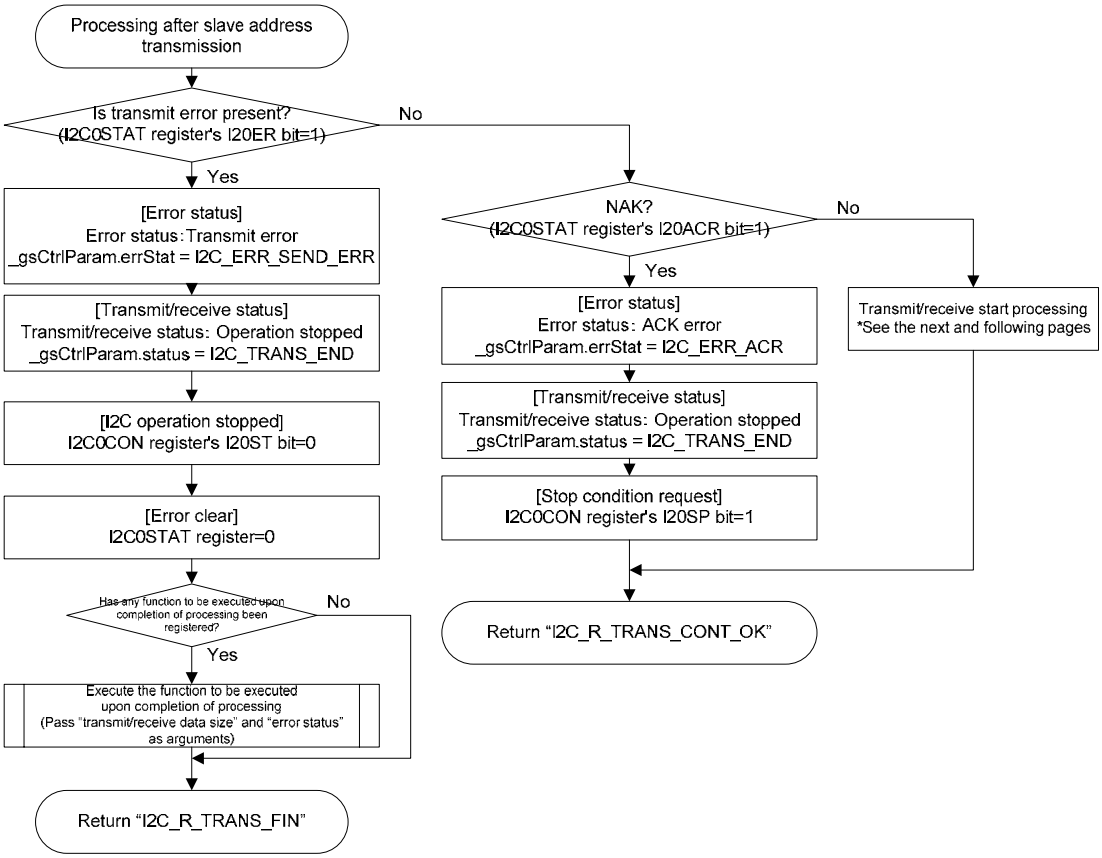

Data reception start processing
i2c_startReceive function
Set I2C transmit/receive control parameters
[Transmit/receive mode]
Set to receive mode
_gsCtrlParam.mode = 1
[Pointer to the address storage area]
_gsCtrlParam.addr = argument "addr"
[Address size]
_gsCtrlParam.addr_size = argument "addr_size"
[Transmit/receive data storage area]
_gsCtrlParam.data = argument "data"
[Transmit/receive data size]
_gsCtrlParam.data_size = argument "data_size"
[Size of transmitted/received data]
No transmitted/received data size
_gsCtrlParam.cnt = 0
[Callback function]
_gsCtrlParam.callBack = argument "func"
[Error status]
Error status: No error
_gsCtrlParam.errStat = 0
[Transmit/receive status]
Transmit/receive status: Operation stopped
_gsCtrlParam.status = I2C_TRANS_END
Transmit data present?
(Either _gsCtrlParam.addr_size or _gsCtrlParam.data_size is other than 0)
No
No receive data
Return value: I2C_R_TRANS_FIN
Yes
I2C bus free state?
(I2C0STAT register's I20BB bit = 0)
No
I2C bus busy state
Return value: I2C_R_BUS_BUSY
Yes
[Transmit/receive status]
Slave address is being transmitted
_gsCtrlParam.status = I2C_SEND_SLAVE_ADDRESS
Transmit address present?
_gsCtrlParam.addr_size is other than 0)
No
Yes
[I2C slave address setting]
Slave address setting and data transmission mode
I2C0SA register = argument "slave_adr" << 1
[I2C slave address setting]
Slave address setting and data reception mode
I2C0SA register = argument "slave_adr" << 1 | 0x01
[Communication start]
I2C0CON register's I20ST bit = 1
End of processing

3.6.7.4. i2c_continue Function

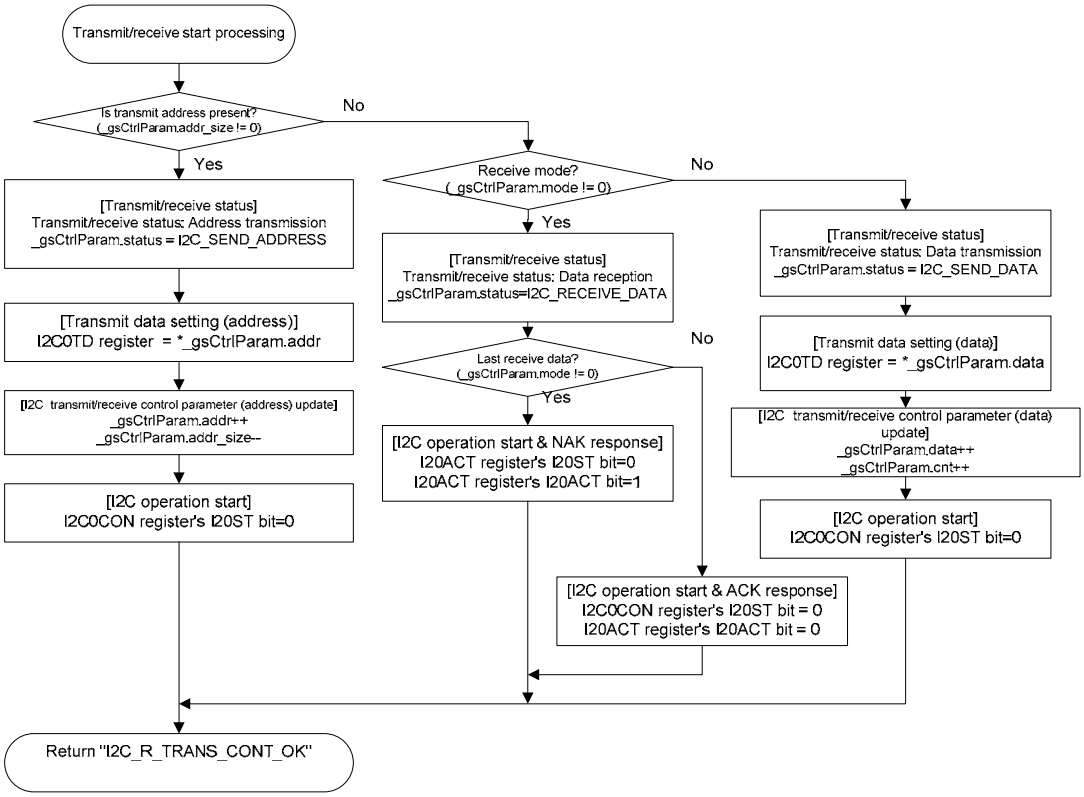
This function continues data transmission/reception. If there is any registered callback function upon completion of transmission/reception or upon occurrence of an error, that callback function will be executed.

Function name:	int i2c_continue(void)
Arguments:	None
Return values:	int Transmit/receive processing ended: I2C_R_TRANS_FIN (=1) Transmit/receive processing is going on (transmission/reception succeeded): I2C_R_TRANS_CONT_OK (=0) Transmit/receive processing ended (transmission/reception failed): I2C_R_TRANS_FIN_NG (=-2)
Processing: (1/8)	

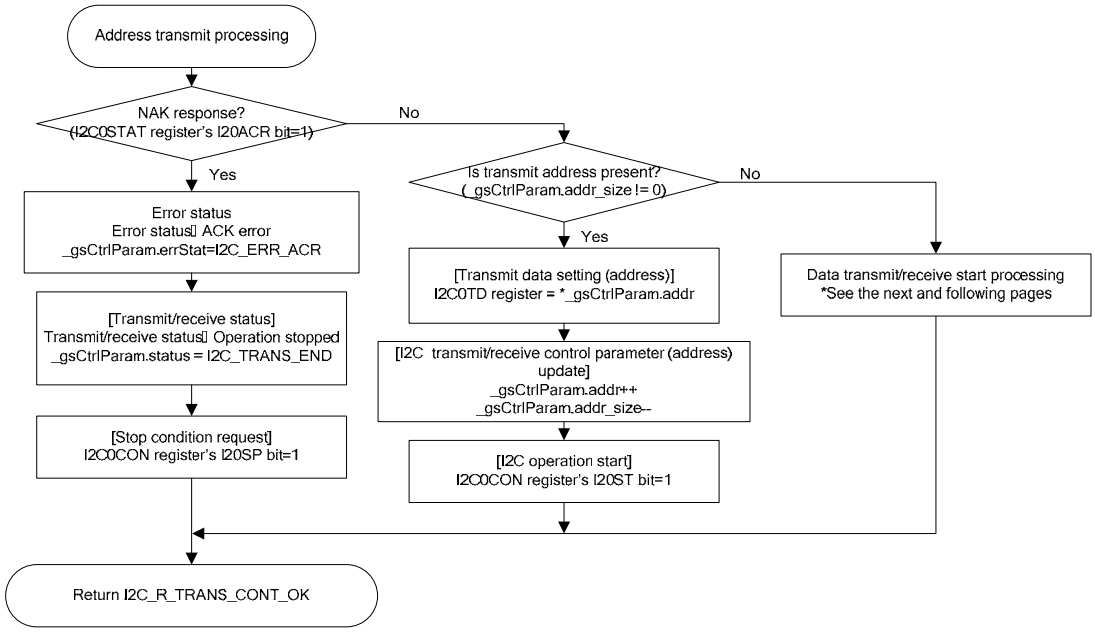
Processing: (2/8)



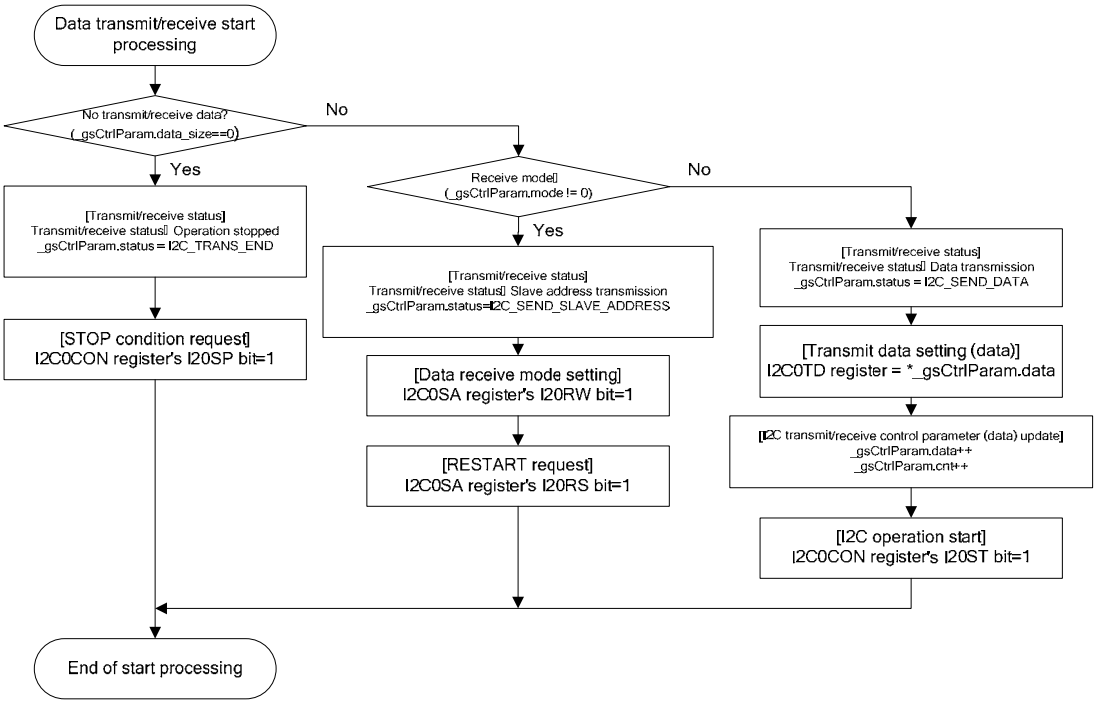
Processing: (3/8)



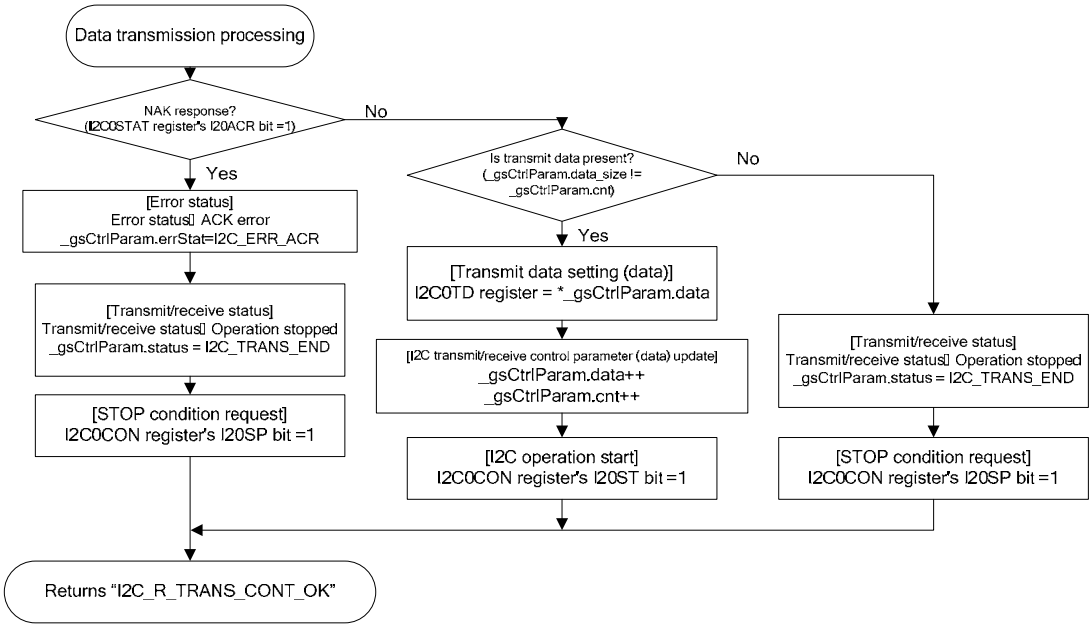
Processing: (4/8)



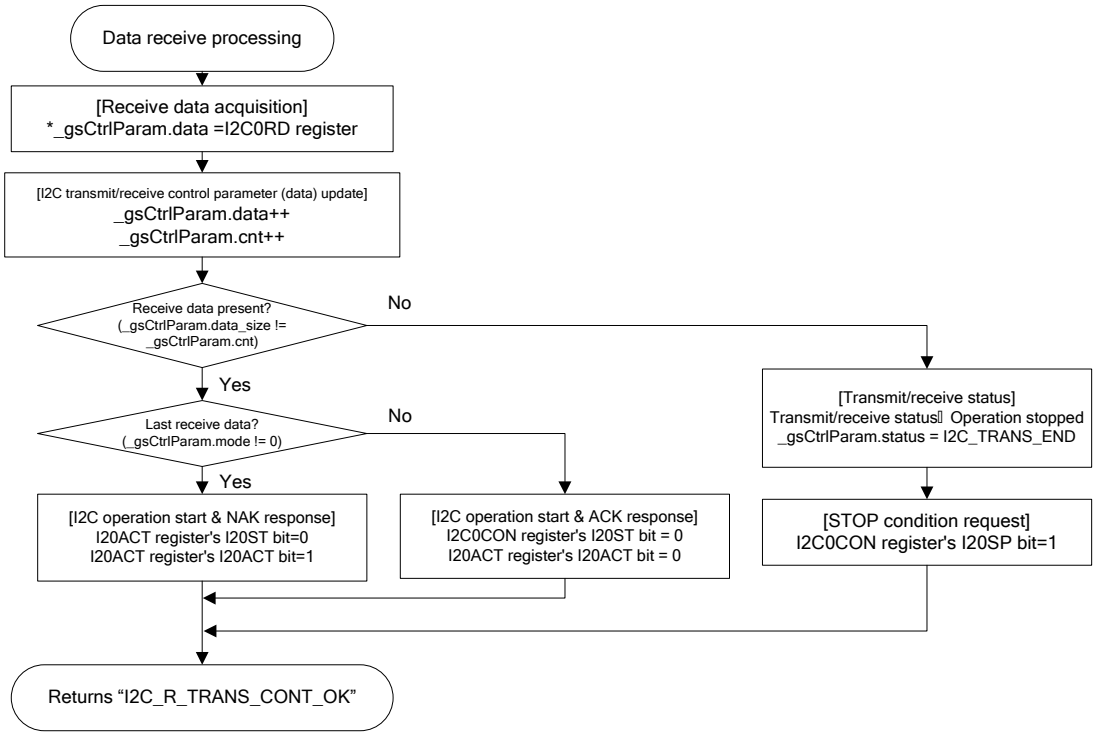
Processing:
(5/8)

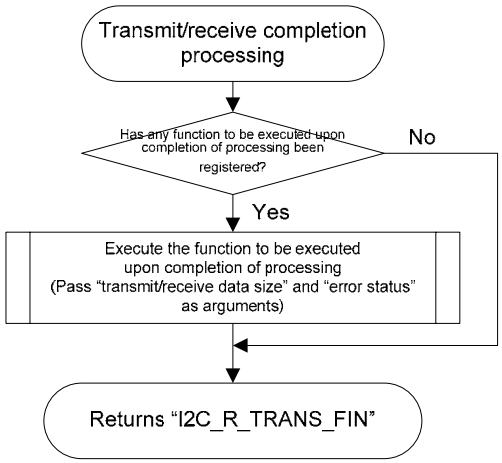


Processing: (6/8)



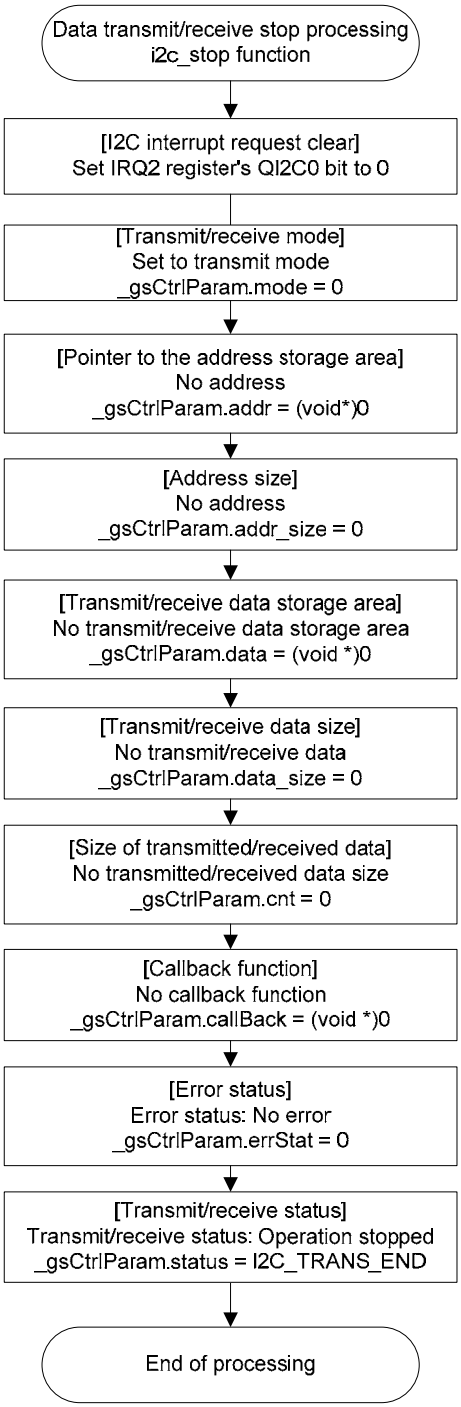
Processing: (7/8)



Processing: (8/8)	

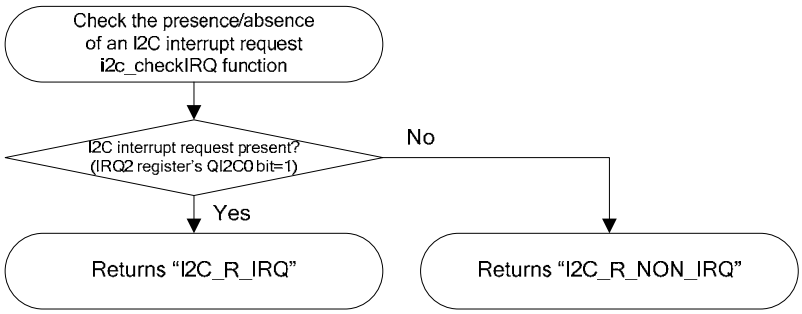
3.6.7.5. i2c_stop Function

This function stops data transmission/reception.

Function name:	void i2c_stop(void)
Arguments:	None
Return values:	None
Processing:	

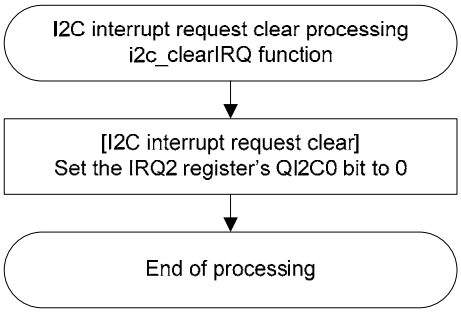
3.6.7.6. i2c_checkIRQ Function

This function checks the QI2C0 bit of the interrupt request register 2 to indicate the presence or absence of an I2C interrupt request.

Function name:	int i2c_checkIRQ(void)
Arguments:	None
Return values:	int I2C interrupt request is present: I2C_R_IRQ I2C interrupt request is absent: I2C_R_NON_IRQ
Processing:	

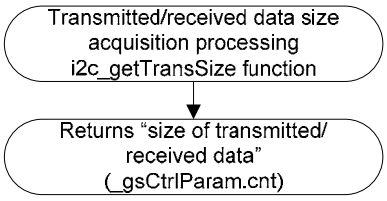
3.6.7.7. i2c_clearIRQ Function

This function clears the I2C interrupt request (interrupt request register 2's QI2C0 bit).

Function name:	void i2c_clearIRQ(void)
Arguments:	None
Return values:	None
Processing:	

3.6.7.8. i2c_getTransSize Function

This function acquires the size of transmitted/received data.
If the i2c_stop function is called, the size of transmitted/received data is cleared.

Function name:	unsigned int i2c_getTransSize(void)
Arguments:	None
Return values:	unsigned int Size of transmitted/received data (*Size of the data that has been transferred after the i2c_startSend or i2c_startReceive function is executed and before the i2c_getTransSize function is called.)
Processing:	

3.7. EEPROM Module

3.7.1. Overview of Functions

The EEPROM module writes/reads data to/from EEPROM using the I2C of the MCU. Controllable items are N-byte write, N-byte read, and write protection.

3.7.2. List of APIs

The following table lists the EEPROM module APIs.

Table 3-20 EEPROM APIs

Function name	Function
eeprom_init function	Enables EEPROM write-protect.
eeprom_read function	Starts reading data from EEPROM.
eeprom_write function	Starts writing data to EEPROM.
eeprom_continue function	Continues data write or read processing.
eeprom_stop function	Stops data write or read processing.
eeprom_getStatus function	Acquires the EEPROM control module status.
eeprom_writeProtect function	Enables or disables EEPROM write-protect.

Table 3-21 EEPROM Subroutines

Function name	Description
_i2cFin function	Callback function to be called upon completion of I2C transmission/reception

3.7.3. List of Constants

The following table lists the constants used in the EEPROM module.

Table 3-22 List of Constants for Return Values

Constant name	Defined value	Description
EEPROM_R_OK	0	Data write/read start processing succeeded
EEPROM_R_NG	-1	Data write/read start processing failed
EEPROM_R_PROCESS	1	Data write/read processing is in progress
EEPROM_R_SUCCESS	0	Data write/read processing ended normally
EEPROM_R_ERROR	-1	Data write/read processing ended abnormally

Table 3-23 List of Constants for EEPROM Customization

Constant name	Defined value	Description
EEPROM_SLAVE_ADDRESS	0x50	Slave address
EEPROM_PAGE_SIZE	64	Page size

Table 3-24 List of Constants for Internal Statuses

Constant name	Defined value	Description
ST_STOP	0	Processing stopped
ST_I2C_SEND_START	1	Started data transmission
ST_I2C_SEND_EXEC	2	Data is being transmitted
ST_I2C_RECEIVE_START	3	Started data reception
ST_I2C_RECEIVE_EXEC	4	Data is being received
ST_EEP_WRITE_START	5	Start writing to EEPROM
ST_EEP_WRITE_EXEC	6	Data is being written to EEPROM

3.7.4. Structure

This section describes the structures used in the EEPROM module.

■ Control parameters

```
typedef struct {
    unsigned char       address[2];     // Write/read start address
    unsigned char *     data;           // Pointer to the transmit/receive data storage area
    unsigned int        remain_size;    // Remaining data size
    unsigned int        total_size;     // Size of the data transmitted/received actually
    unsigned int        proc_size;      // Size for one data-write/read
    int                 result;         // Execution result
    int                 internal_status; // Internal status
} tEepromCtrlParam;
```

3.7.5. List of Variables

The following table lists the variables used in the EEPROM module.

Table 3-25 List of Variables

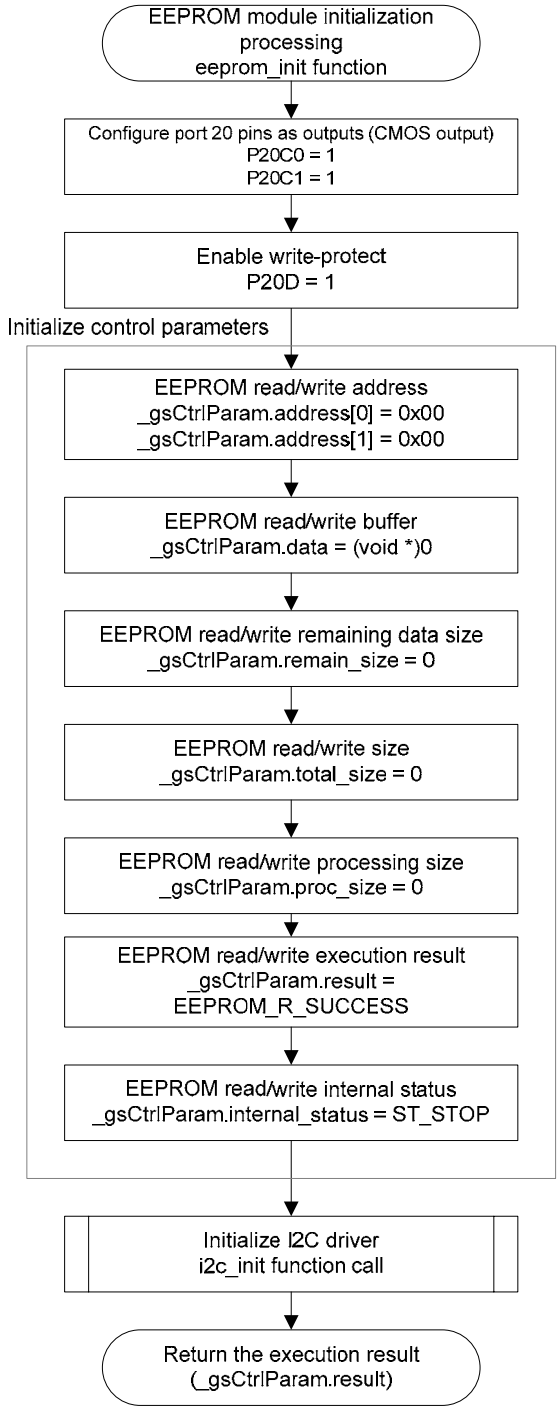
Variable name	Initial value	Description
static tEepromCtrlParam_gsCtrlParam	address: 0x00,0x00 data: NULL remain_size: 0 total_size: 0 proc_size: 0 result: 0 internal_status: 0	Information to be used at read/write

3.7.6. Details of APIs

This section describes details of the EEPROM module APIs.

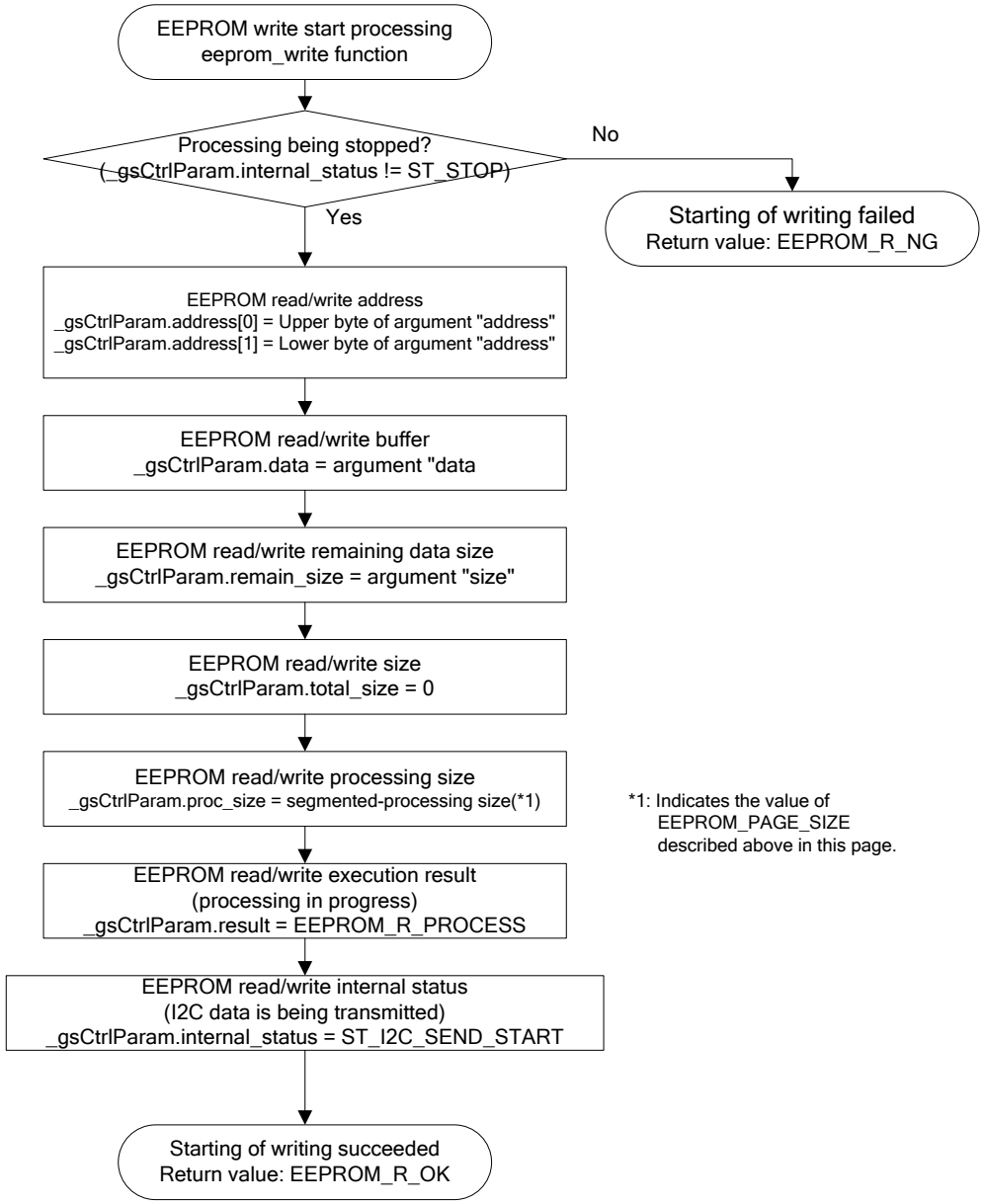
3.7.6.1. eeprom_init Function

This function initializes the EEPROM module. In initialization, the function enables EEPROM write-protect, sets the initial settings for internal variables, and initializes the I2C module.

Function name:	void eeprom_init(void)
Arguments:	None
Return values:	None
Processing:	

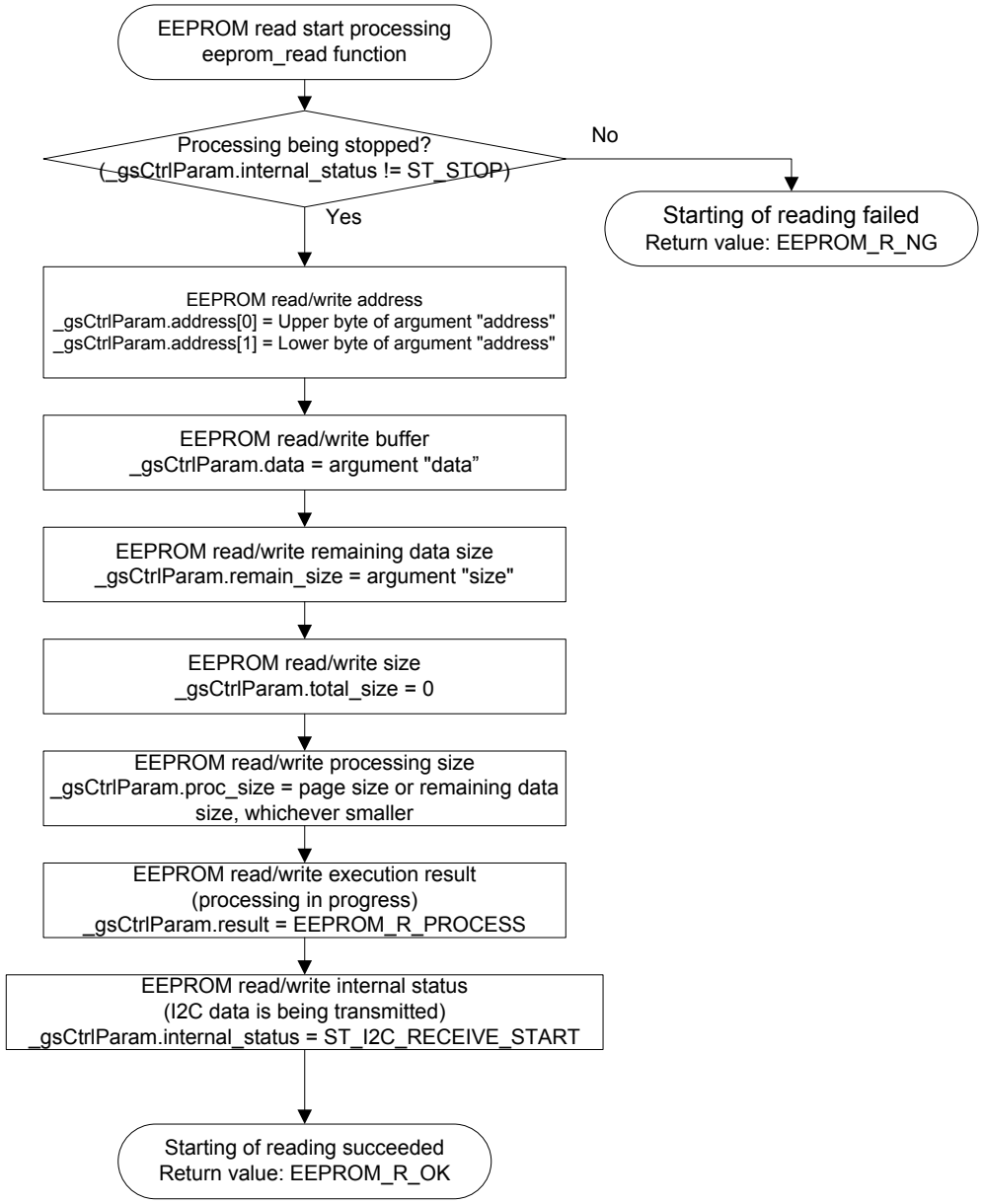
3.7.6.2. eeprom_write Function

This function starts writing to EEPROM. If the transmit data size is larger than EEPROM_PAGE_SIZE, the data is segmented into EEPROM_PAGE_SIZE size to write to EEPROM.

Function name:	int eeprom_write(void) unsigned int address, unsigned char* data, unsigned int size)
Arguments:	unsigned int address …. Device's address at which to start writing unsigned char* data … Pointer to the area that contains transmit data unsigned int size …. Transmit data size (byte units)
Return values:	Int EEPROM_R_OK(=0): The starting of writing succeeded. EEPROM_R_NG(=-1): The starting of writing failed.
Processing:	

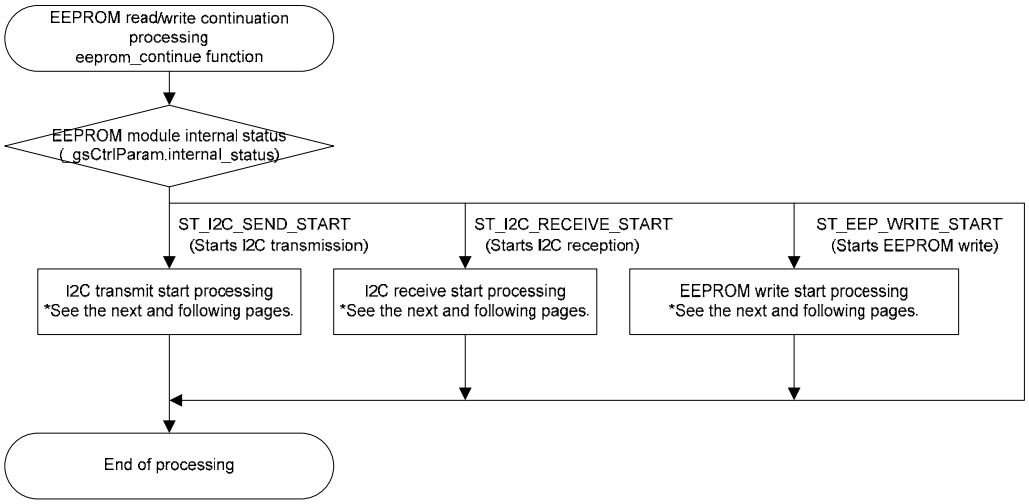
3.7.6.3. eeprom_read Function

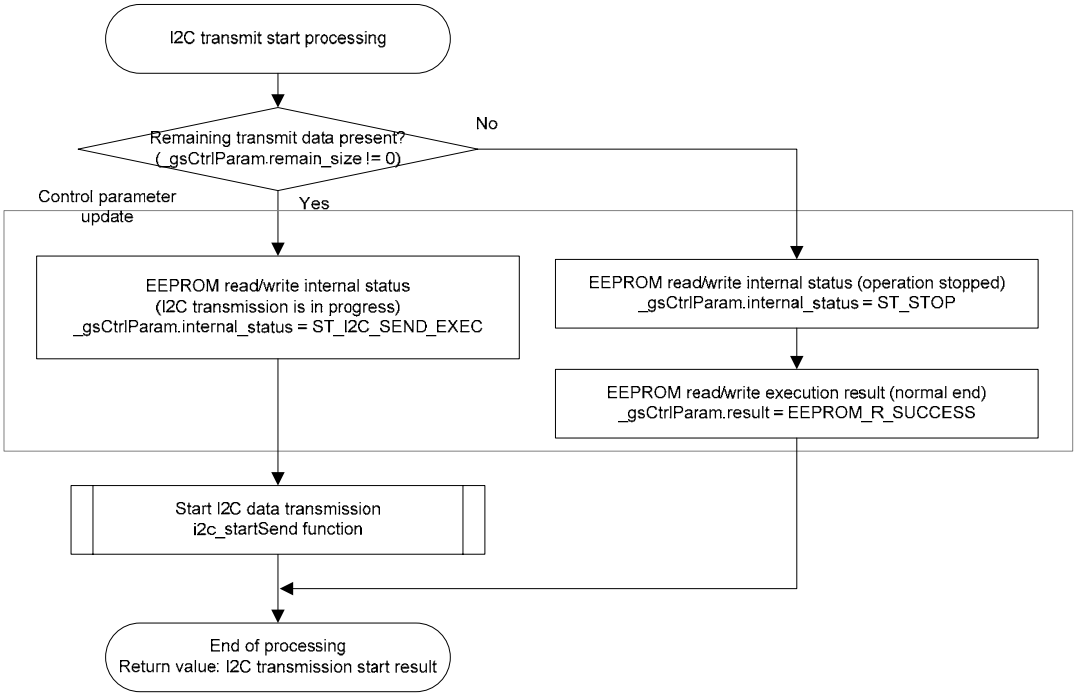
This function starts reading data from EEPROM.

Function name:	int eeprom_read(unsigned int address, unsigned char* data, unsigned int size)
Arguments:	unsigned int address …. Device's address at which to start reading unsigned char* data … Pointer to the area that contains receive data unsigned int size …. Receive data size (byte units)
Return values:	Int EEPROM_R_OK(=0): The starting of reading succeeded. EEPROM_R_NG(=-1): The starting of reading failed.
Processing:	

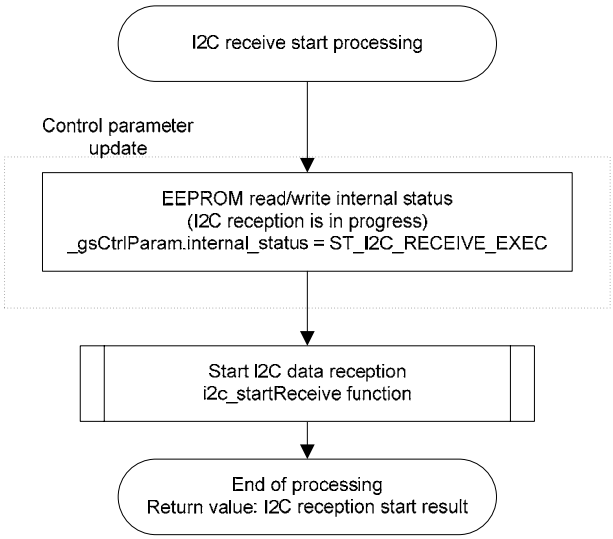
3.7.6.4. eeprom_continue Function

This function continues EEPROM read/write processing.

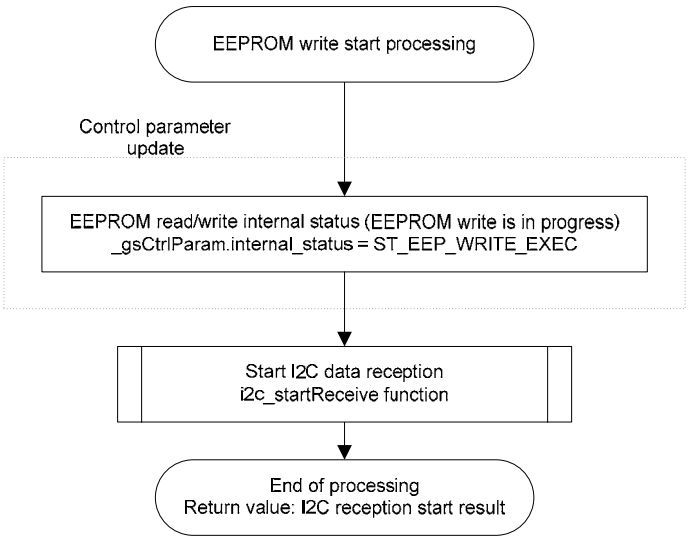
Function name:	int eeprom_continue(void)
Arguments:	None
Return values:	int … EEPROM module processing status EEPROM_R_PROCESS (= 1) … Read/write processing is being executed EEPROM_R_ERROR (= 0) … Abnormal end EEPROM_R_SUCCESS (= -1) … Normal end
Processing: (1/4)	

Processing: (2/4)	

Processing: (3/4)

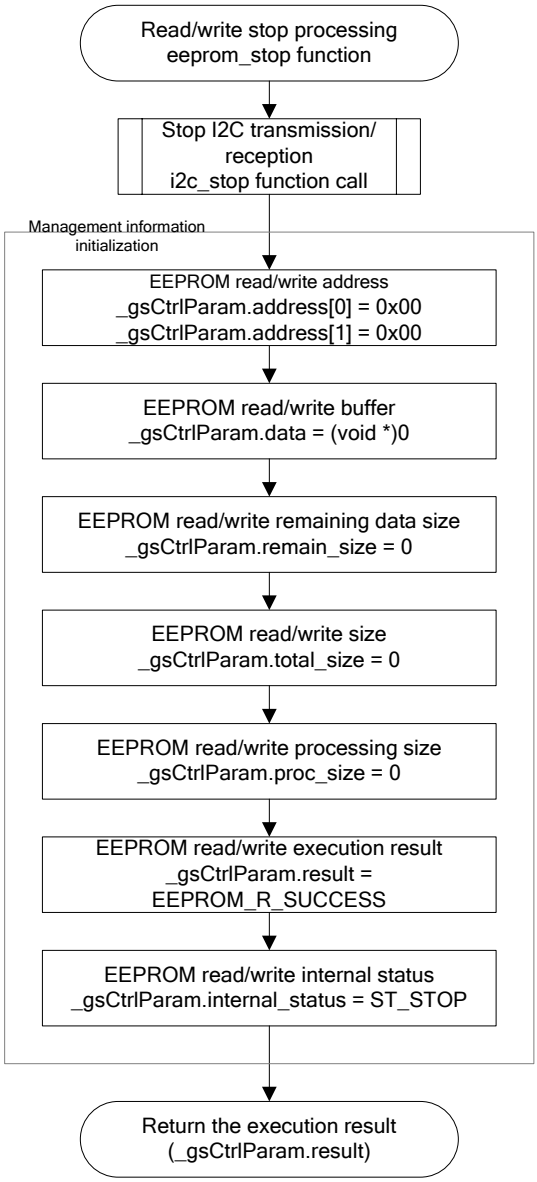


Processing: (4/4)



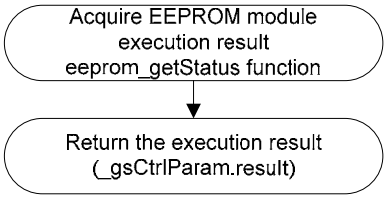
3.7.6.5. eeprom_stop Function

This function stops EEPROM read/write processing.

Function name:	void eeprom_stop(void)
Arguments:	None
Return values:	None
Processing:	

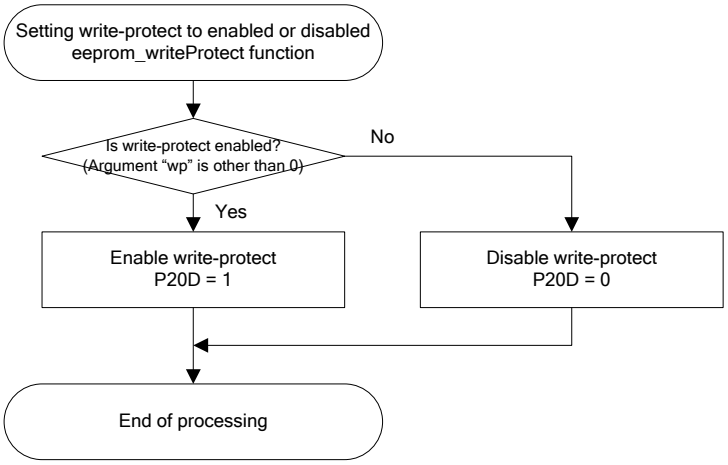
3.7.6.6. eeprom_getStatus Function

This function acquires the EEPROM module processing status.

Function name:	int eeprom_getStatus(void)
Arguments:	None
Return values:	int … EEPROM module processing status EEPROM_R_PROCESS (= 1) … Processing is being executed EEPROM_R_ERROR (= 0) … Abnormal end EEPROM_R_SUCCESS (= -1) … Normal end
Processing:	

3.7.6.7. eeprom_writeProtect Function

This function sets EEPROM write-protect to enabled or disabled.

Function name:	void eeprom_writeProtect(unsigned char wp)
Arguments:	unsigned char wp 0: Write-protect is disabled. Other than 0: Write-protect is enabled.
Return values:	None
Processing:	

.

3.8. LCD Module

3.8.1. Overview of Functions

The LCD module controls the LCD driver of the MCU.
Display on the LCD panel is achieved by APIs that perform initialization (setting of bias, duty, frame frequency, etc), contrast setting, display mode setting, 7-seg or 16-seg display, and mark display.

3.8.2. List of APIs

The following table lists the LCD module APIs.

Table 3-26 LCD Module APIs

Function name	Description
lcd_init function	Initialization
lcd_setContrast function	Contrast value setting
lcd_setLCDMode function	LCD display mode setting
lcd_dispHour function	Displays hours.
lcd_dispMin function	Displays minutes.
lcd_dispSec function	Displays seconds.
lcd_dispMain function	Displays the 12 digits on the lower part of the panel
lcd_disp1Digit function	Displays a 1-digit numeral.
lcd_disp2Digit function	Displays a 2-digit numeral.
lcd_disp4Digit function	Displays a 4-digit numeral.
lcd_dipMark function	Displays various marks.

3.8.3. List of Subroutines

The following table lists the LCD module subroutines.

Table 3-27 LCD Module Subroutines

Function name	Description
_getDSPR function	Acquisition of a DSPR address
_getMarkDSPR function	Acquisition of a mark display DSPR address
_get7SegPtn function	Acquisition of a 7-seg pattern
_dsp7SEG function	7-seg display (1 digit)
_dsp7SEG_Ndigit function	7-seg display (multiple digits)
_get16SegPtn function	Acquisition of a 16-seg pattern
_dsp16SEG function	16-seg display (1 digit)
_dsp16SEG_Ndigit function	16-seg display (multiple digits)
_TableCopyFunc function	Transfers assignment table data for the display assignment registers A and B.

3.8.4. List of Constants

The following tables list the constants used in the LCD module.

Table 3-28 Constants for Arguments (1)

Constant name	Defined value	Description
LCD_BSN_32KHZ	0	Specifies 32 kHz as the voltage multiplication clock of the bias generation circuit.
LCD_BSN_16KHZ	1	Specifies 16 kHz as the voltage multiplication clock of the bias generation circuit.
LCD_BSN_8KHZ	2	Specifies 8 kHz as the voltage multiplication clock of the bias generation circuit.
LCD_BSN_4KHZ	3	Specifies 4 kHz as the voltage multiplication clock of the bias generation circuit.
LCD_BSN_2KHZ	4	Specifies 2 kHz as the voltage multiplication clock of the bias generation circuit.
LCD_BSN_1KHZ	5	Specifies 1 kHz as the voltage multiplication clock of the bias generation circuit.
LCD_BSN_512HZ	6	Specifies 512 Hz as the voltage multiplication clock of the bias generation circuit.
LCD_BSN_256HZ	7	Specifies 256 Hz as the voltage multiplication clock of the bias generation circuit.
LCD_BSEL_DIV3	0	Specifies 1/3 bias for the bias generation circuit.
LCD_BSEL_DIV4	1	Specifies 1/4 bias for the bias generation circuit.
LCD_DUTY_DIV1	0	Specifies 1/1 duty.
LCD_DUTY_DIV2	1	Specifies 1/2 duty.
LCD_DUTY_DIV3	2	Specifies 1/3 duty.
LCD_DUTY_DIV4	3	Specifies 1/4 duty.
LCD_DUTY_DIV5	4	Specifies 1/5 duty.
LCD_DUTY_DIV6	5	Specifies 1/6 duty.
LCD_DUTY_DIV7	6	Specifies 1/7 duty.
LCD_DUTY_DIV8	7	Specifies 1/8 duty.
LCD_DUTY_DIV9	8	Specifies 1/9 duty.
LCD_DUTY_DIV10	9	Specifies 1/10 duty.
LCD_DUTY_DIV11	10	Specifies 1/11 duty.
LCD_DUTY_DIV12	11	Specifies 1/12 duty.
LCD_DUTY_DIV13	12	Specifies 1/13 duty.
LCD_DUTY_DIV14	13	Specifies 1/14 duty.
LCD_DUTY_DIV15	14	Specifies 1/15 duty.
LCD_DUTY_DIV16	15	Specifies 1/16 duty.
LCD_DUTY_DIV17	16	Specifies 1/17 duty.
LCD_DUTY_DIV18	17	Specifies 1/18 duty.
LCD_DUTY_DIV19	18	Specifies 1/19 duty.
LCD_DUTY_DIV20	19	Specifies 1/20 duty.
LCD_DUTY_DIV21	20	Specifies 1/21 duty.
LCD_DUTY_DIV22	21	Specifies 1/22 duty.
LCD_DUTY_DIV23	22	Specifies 1/23 duty.
LCD_DUTY_DIV24	23	Specifies 1/24 duty.

Table 3-29 Constants for Arguments (2)

Constant name	Defined value	Description
LCD_FRM_64HZ	0	Specifies 64 Hz as the reference frequency of the frame frequency.
LCD_FRM_73HZ	1	Specifies 73 Hz as the reference frequency of the frame frequency.
LCD_FRM_85HZ	2	Specifies 85 Hz as the reference frequency of the frame frequency.
LCD_FRM_102HZ	3	Specifies 102 Hz as the reference frequency of the frame frequency.
LCD_LMD_STOP	0	Specifies LCD stop mode as the mode of display.
LCD_LMD_ALLOFF	1	Specifies LCD all-off mode as the mode of display.
LCD_LMD_NORMAL	2	Specifies LCD display mode (normal mode) as the mode of display.
LCD_LMD_ALLON	3	Specifies LCD all-on mode as the mode of display.
LCD_POS_MAIN_01	0x00	Specifies the display of the 1st digit of the 12 digits on the lower section.
LCD_POS_MAIN_02	0x01	Specifies the display of the 2nd digit of the 12 digits on the lower section.
LCD_POS_MAIN_03	0x02	Specifies the display of the 3rd digit of the 12 digits on the lower section.
LCD_POS_MAIN_04	0x03	Specifies the display of the 4th digit of the 12 digits on the lower section.
LCD_POS_MAIN_05	0x04	Specifies the display of the 5th digit of the 12 digits on the lower section.
LCD_POS_MAIN_06	0x05	Specifies the display of the 6th digit of the 12 digits on the lower section.
LCD_POS_MAIN_07	0x06	Specifies the display of the 7th digit of the 12 digits on the lower section.
LCD_POS_MAIN_08	0x07	Specifies the display of the 8th digit of the 12 digits on the lower section.
LCD_POS_MAIN_09	0x08	Specifies the display of the 9th digit of the 12 digits on the lower section.
LCD_POS_MAIN_10	0x09	Specifies the display of the 10th digit of the 12 digits on the lower section.
LCD_POS_MAIN_11	0x0A	Specifies the display of the 11th digit of the 12 digits on the lower section.
LCD_POS_MAIN_12	0x0B	Specifies the display of the 12th digit of the 12 digits on the lower section.
LCD_POS_HOUR_01	0x10	Specifies the display of the ones digit of hours.
LCD_POS_HOUR_10	0x11	Specifies the display of the tens digit of hours.
LCD_POS_HOUR	0x10	Specifies hour display.
LCD_POS_MIN_01	0x20	Specifies the display of the ones digit of minutes.
LCD_POS_MIN_10	0x21	Specifies the display of the tens digit of minutes.
LCD_POS_MIN	0x20	Specifies minute display.
LCD_POS_SEC_01	0x30	Specifies the display of the ones digit of seconds.
LCD_POS_SEC_10	0x31	Specifies the display of the tens digit of seconds.
LCD_POS_SEC	0x30	Specifies second display.
LCD_POS_MODE_01	0x40	Specifies the display of the 1st digit of Mode (the upper-left four digits on the upper section).
LCD_POS_MODE_02	0x41	Specifies the display of the 2nd digit of Mode (the upper-left four digits on the upper section).
LCD_POS_MODE_03	0x42	Specifies the display of the 3rd digit of Mode (the upper-left four digits on the upper section).

LCD_POS_MODE_04	0x43	Specifies the display of the 4th digit of Mode (the upper-left four digits on the upper section).
LCD_POS_MODE_LOW	0x40	Specifies the display of the 1st and 2nd digits of Mode (the upper-left four digits on the upper section).
LCD_POS_MODE_HIGH	0x42	Specifies the display of the 3rd and 4th digits of Mode (the upper-left four digits on the upper section).
LCD_POS_MODE	0x40	Specifies the display of Mode (the upper-left four digits on the upper section).
LCD_TURNOFF	0	Specifies turning off the LCD.
LCD_TURNON	1	Specifies turning on the LCD.
LCD_TYPE_0_DISP	0	Displays “0” as “0”.
LCD_TYPE_0_BLANK	1	Displays “0” as “blank”.

Table 3-30 Constants for Return Values

Constant name	Defined value	Description
LCD_R_OK	0	Initialization succeeded
LCD_R_ERR_BSN	-1	Selection of the voltage multiplication clock for the bias generation circuit is outside the range.
LCD_R_ERR_BSEL	-2	The bias selection for the bias generation circuit is outside the range.
LCD_R_ERR_DUTY	-3	The duty selection is outside the range.
LCD_R_ERR_FRM	-4	The frame frequency selection is outside the range.
LCD_R_ERR_LMD	-5	The display mode selection is outside the range.
LCD_R_ERR_CNT	-6	The contrast value is outside the range.
LCD_R_ERR_POS	-7	The display position setting is outside the range.
LCD_R_ERR_EN	-8	The on/off setting is outside the range.
LCD_R_ERR_TYPE	-9	The display type is outside the range.
LCD_R_ERR_DIGIT	-10	The display value is outside the range.
LCD_R_ERR_MARKNO	-11	The display position setting is outside the range.

Table 3-31 Constants for LCD Codes

Constant name	Defined value	Description
LCD_CODE_NUM_0	0x30	LCD code corresponding to a display of “0”
LCD_CODE_NUM_1	0x31	LCD code corresponding to a display of “1”
LCD_CODE_NUM_2	0x32	LCD code corresponding to a display of “2”
LCD_CODE_NUM_3	0x33	LCD code corresponding to a display of “3”
LCD_CODE_NUM_4	0x34	LCD code corresponding to a display of “4”
LCD_CODE_NUM_5	0x35	LCD code corresponding to a display of “5”
LCD_CODE_NUM_6	0x36	LCD code corresponding to a display of “6”
LCD_CODE_NUM_7	0x37	LCD code corresponding to a display of “7”
LCD_CODE_NUM_8	0x38	LCD code corresponding to a display of “8”
LCD_CODE_NUM_9	0x39	LCD code corresponding to a display of “9”
LCD_CODE_HEX_A	0x41	LCD code corresponding to a display of “A”
LCD_CODE_HEX_B	0x42	LCD code corresponding to a display of “B”
LCD_CODE_HEX_C	0x43	LCD code corresponding to a display of “C”
LCD_CODE_HEX_D	0x44	LCD code corresponding to a display of “D”
LCD_CODE_HEX_E	0x45	LCD code corresponding to a display of “E”
LCD_CODE_HEX_F	0x46	LCD code corresponding to a display of “F”
LCD_CODE_BLANK	0x20	LCD code corresponding to a display of “(blank)”
LCD_CODE_MINUS	0x2D	LCD code corresponding to a display of “-”
LCD_CODE_G	0x47	LCD code corresponding to a display of “G”
LCD_CODE_H	0x48	LCD code corresponding to a display of “H”
LCD_CODE_I	0x49	LCD code corresponding to a display of “I”
LCD_CODE_J	0x4A	LCD code corresponding to a display of “J”

Constant name	Defined value	Description
LCD_CODE_K	0x4B	LCD code corresponding to a display of “K”
LCD_CODE_L	0x4C	LCD code corresponding to a display of “L”
LCD_CODE_M	0x4D	LCD code corresponding to a display of “M”
LCD_CODE_N	0x4E	LCD code corresponding to a display of “N”
LCD_CODE_O	0x4F	LCD code corresponding to a display of “O”
LCD_CODE_P	0x50	LCD code corresponding to a display of “P”
LCD_CODE_Q	0x51	LCD code corresponding to a display of “Q”
LCD_CODE_R	0x52	LCD code corresponding to a display of “R”
LCD_CODE_S	0x53	LCD code corresponding to a display of “S”
LCD_CODE_T	0x54	LCD code corresponding to a display of “T”
LCD_CODE_U	0x55	LCD code corresponding to a display of “U”
LCD_CODE_V	0x56	LCD code corresponding to a display of “V”
LCD_CODE_W	0x57	LCD code corresponding to a display of “W”
LCD_CODE_X	0x58	LCD code corresponding to a display of “X”
LCD_CODE_Y	0x59	LCD code corresponding to a display of “Y”
LCD_CODE_Z	0x5A	LCD code corresponding to a display of “Z”

Table 3-32 MARK Code Constants

Constant name	Defined value	Description
LCD_MARK_DOC1	0x00	“doC1” mark
LCD_MARK_DOF1	0x01	“doF1” mark
LCD_MARK_HPA	0x02	“hpa” mark
LCD_MARK_DOC2	0x03	“doC2” mark
LCD_MARK_DOF2	0x04	“doF2” mark
LCD_MARK_PERCENT	0x05	“%” mark
LCD_MARK_COL1	0x06	“COL1” mark
LCD_MARK_COL2	0x07	“COL2” mark
LCD_MARK_C2	0x08	“C2” mark
LCD_MARK_C1	0x09	“C1” mark
LCD_MARK_PM	0x0A	“PM” mark
LCD_MARK_AM	0x0B	“AM” mark
LCD_MARK_DATE	0x0C	“DATE” mark
LCD_MARK_MONTH	0x0D	“MONTH” mark
LCD_MARK_YEAR	0x0E	“YEAR” mark
LCD_MARK_T1	0x0F	“T1” mark
LCD_MARK_T2	0x10	“T2” mark
LCD_MARK_B1	0x11	“B1” mark
LCD_MARK_B2	0x12	“B2” mark
LCD_MARK_B3	0x13	“B3” mark
LCD_MARK_B4	0x14	“B4” mark
LCD_MARK_E	0x15	“E” mark
LCD_MARK_K	0x16	“K” mark
LCD_MARK_M	0x17	“M” mark
LCD_MARK_PLUS	0x18	“+” mark
LCD_MARK_MINUS	0x19	“-” mark
LCD_MARK_MULTI	0x1A	“∗” mark
LCD_MARK_DIV	0x1B	“÷” mark
LCD_MARK_11H	0x1C	“11H” mark
LCD_MARK_12H	0x1D	“12H” mark
LCD_MARK_13H	0x1E	“13H” mark
LCD_MARK_14H	0x1F	“14H” mark
LCD_MARK_15H	0x20	“15H” mark
LCD_MARK_16H	0x21	“16H” mark
LCD_MARK_17H	0x22	“17H” mark
LCD_MARK_18H	0x23	“18H” mark
LCD_MARK_19H	0x24	“19H” mark
LCD_MARK_20H	0x25	“20H” mark
LCD_MARK_21H	0x26	“21H” mark
LCD_MARK_22H	0x27	“22H” mark
LCD_MARK_Y16	0x28	“Y16” mark
LCD_MARK_Y15	0x29	“Y15” mark
LCD_MARK_Y14	0x2A	“Y14” mark
LCD_MARK_Y13	0x2B	“Y13” mark
LCD_MARK_Y12	0x2C	“Y12” mark
LCD_MARK_Y11	0x2D	“Y11” mark
LCD_MARK_Y10	0x2E	“Y10” mark
LCD_MARK_Y9	0x2F	“Y9” mark
LCD_MARK_Y8	0x30	“Y8” mark
LCD_MARK_Y7	0x31	“Y7” mark
LCD_MARK_Y6	0x32	“Y6” mark
LCD_MARK_Y5	0x33	“Y5” mark

Constant name	Defined value	Description
LCD_MARK_Y4	0x34	“Y4” mark
LCD_MARK_Y3	0x35	“Y3” mark
LCD_MARK_Y2	0x36	“Y2” mark
LCD_MARK_Y1	0x37	“Y1” mark

3.8.5. API Details

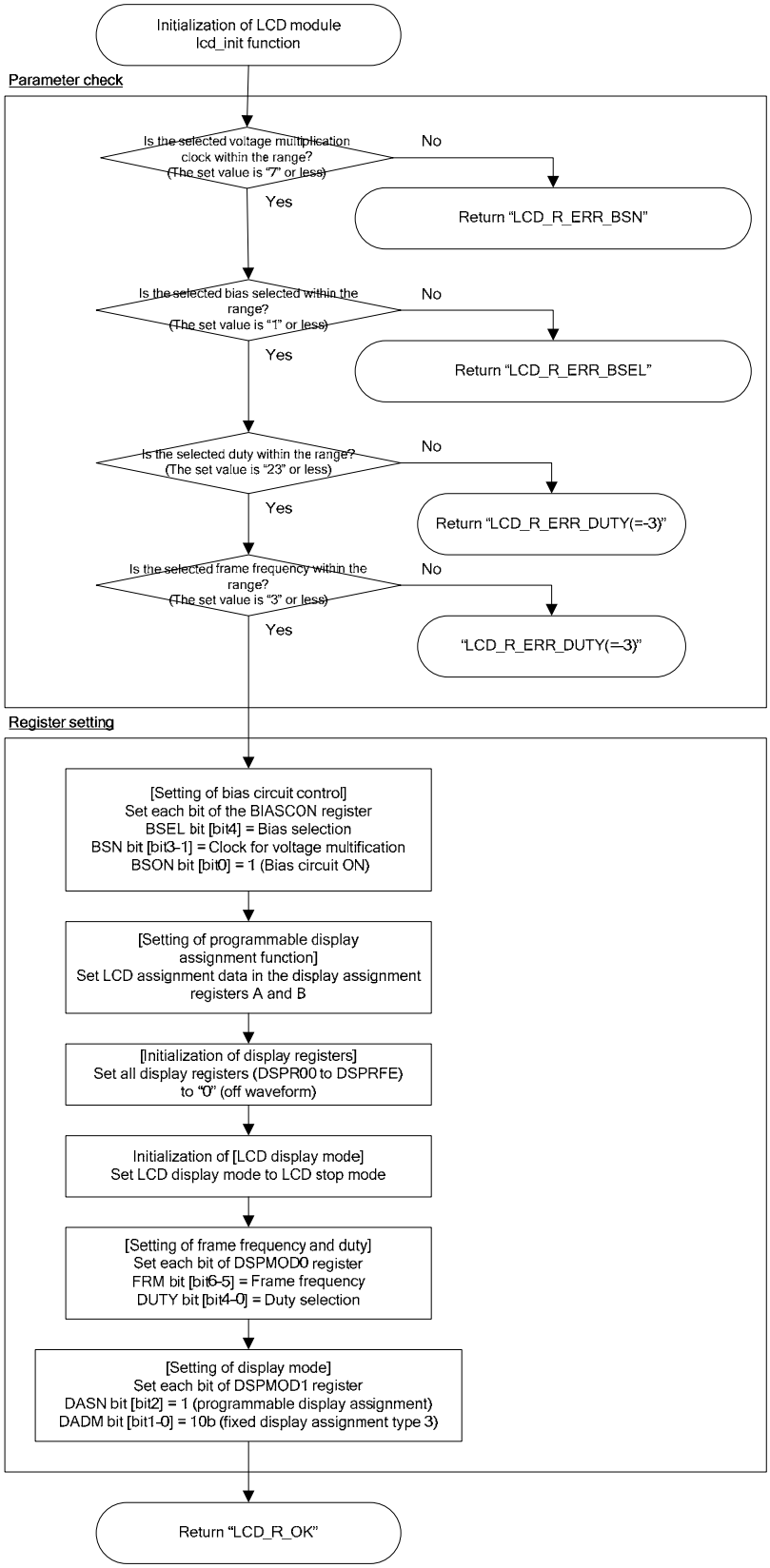
This section describes details of the LCD module.

3.8.5.1. lcd_init Function

This function initializes the LCD driver section of the MCU according to the specification of the LCD panel used. Bias, duty, and frame frequency are set by this function.

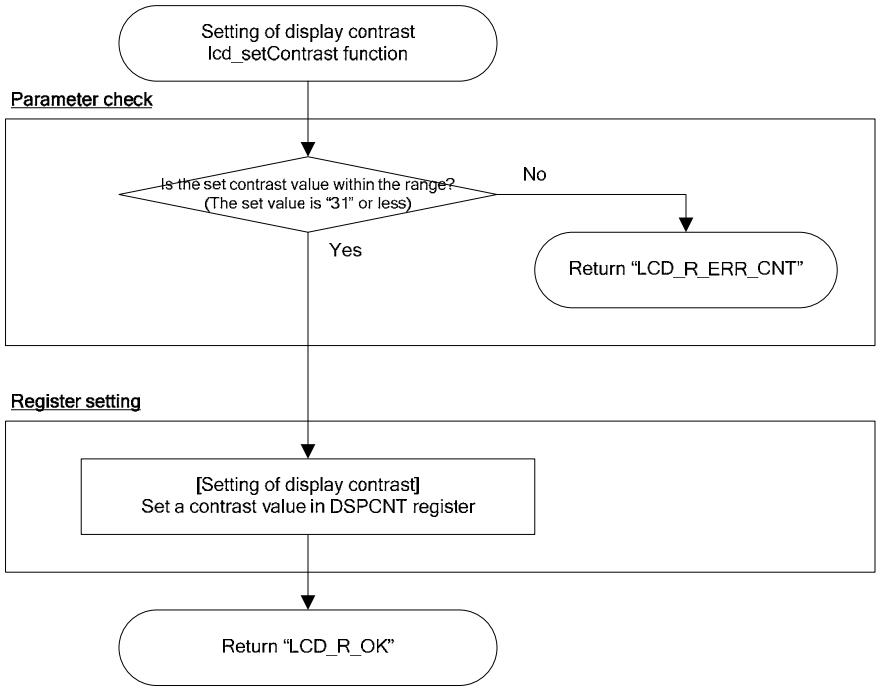
Function name:	int lcd_init(unsigned char bsn, unsigned char bsel, unsigned char duty, unsigned char frm)
Arguments:	unsigned char bsn … Voltage multiplication clock selection for the bias generation circuit 1/1LSCLK(32kHz): LCD_BSN_32KHZ(=0) 1/2LSCLK(16kHz): LCD_BSN_16KHZ(=1) 1/4LSCLK(8kHz): LCD_BSN_8KHZ(=2) 1/8LSCLK(4kHz): LCD_BSN_4KHZ(=3) 1/16LSCLK(2kHz): LCD_BSN_2KHZ(=4) 1/32LSCLK(1kHz): LCD_BSN_1KHZ(=5) 1/64LSCLK(512Hz): LCD_BSN_512HZ(=6) 1/128LSCLK(256Hz): LCD_BSN_256HZ(=7) unsigned char bsel … Bias selection for the bias generation circuit 1/3 bias: LCD_BSEL_DIV3(=0) 1/4 bias: LCD_BSEL_DIV4(=1) unsigned char duty … Duty selection 1/1 duty: LCD_DUTY_DIV1 … (* For details, see Table 3-28.) 1/24 duty: LCD_DUTY_DIV24 unsigned char frm … Selection of frame frequency Reference frequency 64Hz: LCD_FRM_64HZ(=0) Reference frequency 73Hz: LCD_FRM_73HZ(=1) Reference frequency 85Hz: LCD_FRM_85HZ(=2) Reference frequency 102Hz: LCD_FRM_102HZ(=3)
Return values:	int Initialization succeeded: LCD_R_OK(=0) The selected voltage multiplication clock selected for the bias generation circuit is outside the range: LCD_R_ERR_BSN(=-1) The selected bias for the bias generation circuit is outside the range: LCD_R_ERR_BSEL(= -2) The selected duty is outside the range: LCD_R_ERR_DUTY(=-3) The selected frame frequency is outside the range: LCD_R_ERR_FRM(= -4)

Processing: See next page.



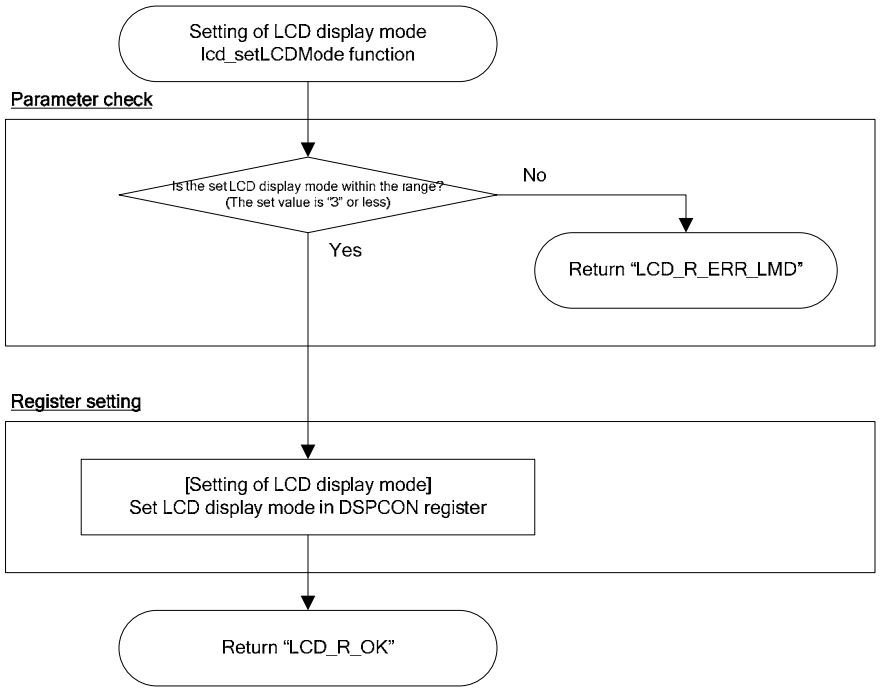
3.8.5.2. lcd_setContrast Function

This function sets the contrast of the LCD. It allows 32 levels of contrast adjustment.

Function name:	int lcd_setContrast(unsigned char cnt)
Arguments:	unsigned char cnt … Contrast value (0 to 31)
Return values:	int Setting succeeded: LCD_R_OK(=0) The set contrast value is outside the range: LCD_R_ERR_CNT (=-6)
Processing:	

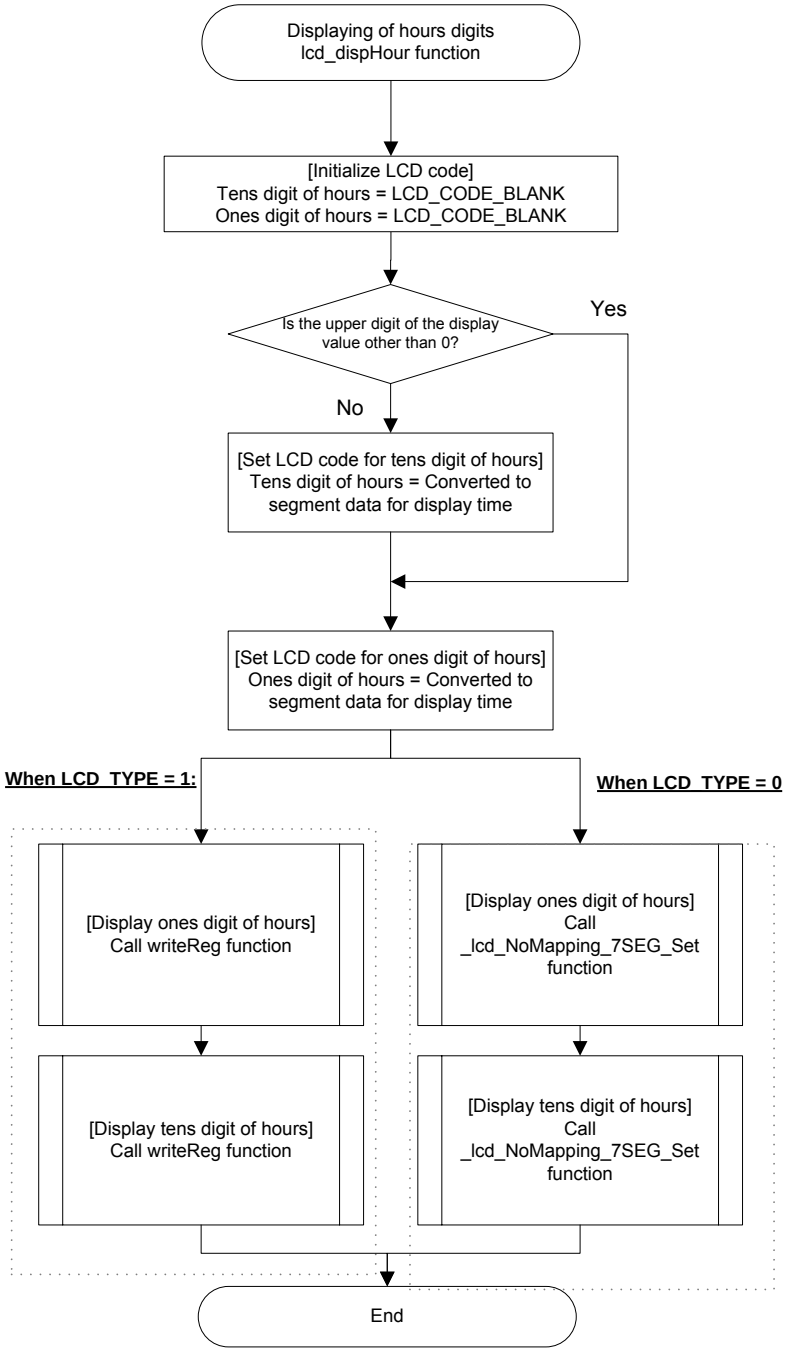
3.8.5.3. lcd_setLCDMode Function

This function sets the display mode of LCD by selecting from among four modes including stop mode and display mode.

Function name:	int lcd_setLCDMode(unsigned char mode)
Arguments:	unsigned char mode … LCD display mode LCD stop mode: LCD_LMD_STOP(=0) LCD all-off mode: LCD_LMD_ALLOFF(=1) LCD display mode: LCD_LMD_NORMAL(=2) LCD all-on mode: LCD_LMD_ALLON(=3)
Return values:	int Setting succeeded: LCD_R_OK(=0) The set LCD display mode is outside the range: LCD_R_ERR_LMD (=-5)
Processing:	

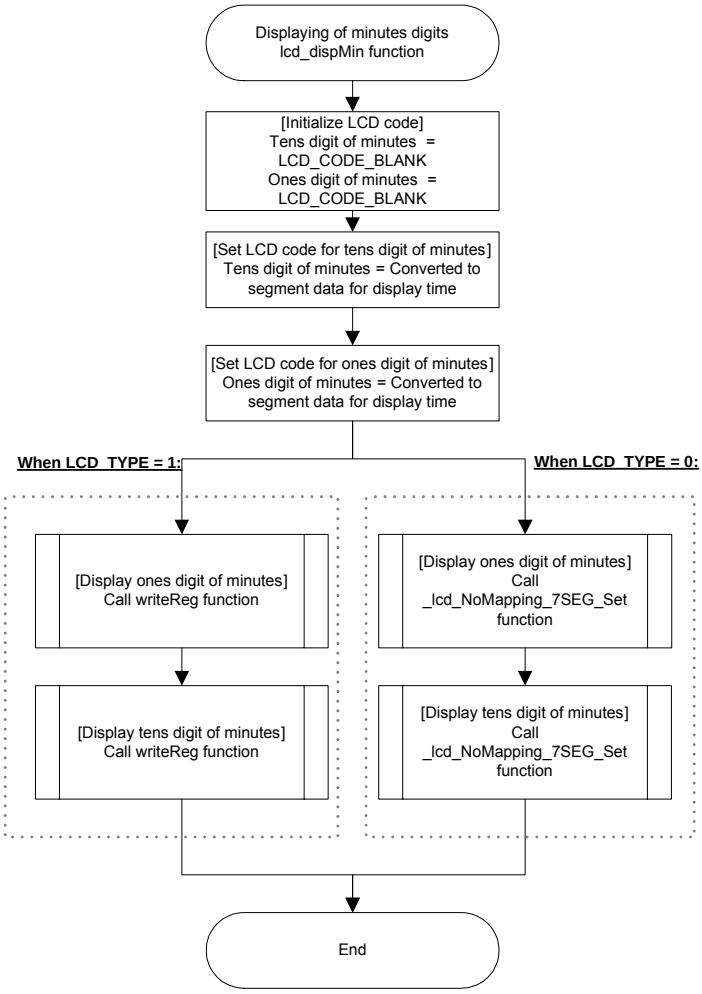
3.8.5.4. lcd_dispHour Function

This function displays hours (0–99 or blank) on the hours digits (2 digits) of the LCD. If a value of 0 to 99 is specified for the “argument “hour”, this function displays hours (the tens digit is zero-suppressed). If the argument “hour” is 100 or more, the hours digits (2 digits) are displayed as blanks.

Function name:	void lcd_dispHour(unsigned char hour)
Arguments:	unsigned char hour … Display value
Return values:	None
Processing:	

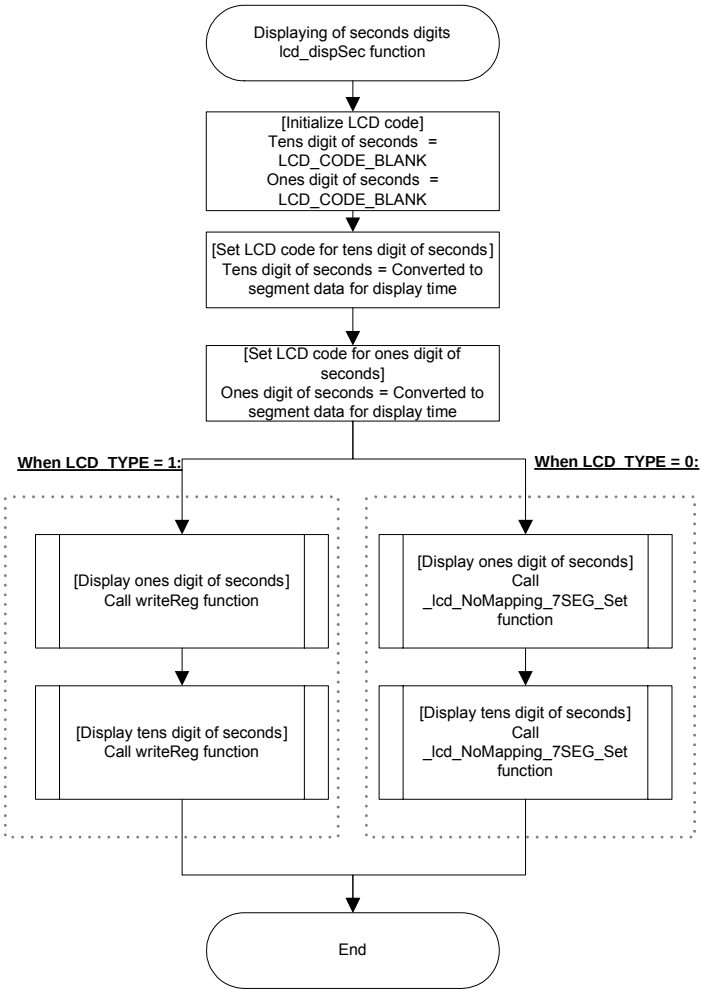
3.8.5.5. lcd_dispMin Function

This function displays minutes (0–99 or blank) on the minutes digits (2 digits) of the LCD. If a value of 0 to 99 is specified for the argument “min”, this function displays minutes. If the argument “min” is 100 or more, the minutes digits (2 digits) are displayed as blanks.

Function name:	void lcd_dispMin(unsigned char min)
Arguments:	unsigned char min … Display value
Return values:	None
Processing:	

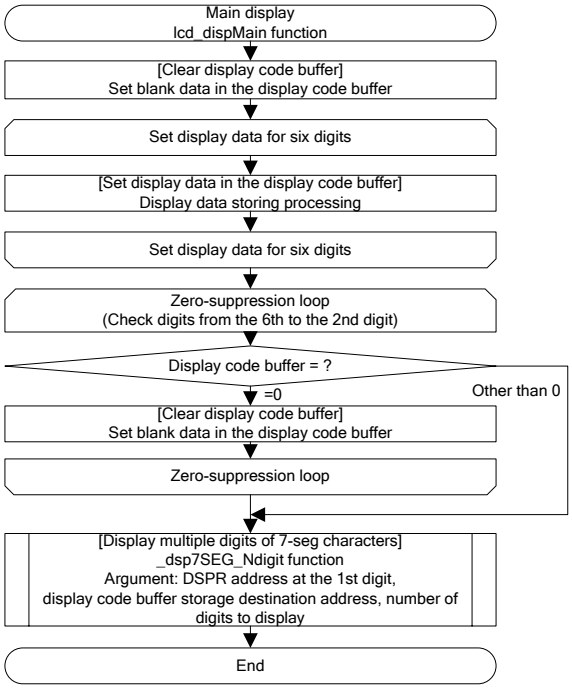
3.8.5.6. lcd_dispSec Function

This function displays seconds (0–99 or blank) on the seconds digits (2 digits) of the LCD. If a value of 0 to 99 is specified for the argument “sec”, this function displays seconds. If the argument “sec” is 100 or more, the seconds digits (2-digit) are displayed as blanks.

Function name:	void lcd_dispSec(unsigned char sec)
Arguments:	unsigned char sec … Display value
Return values:	None
Processing:	

3.8.5.7. lcd_dispMain Function

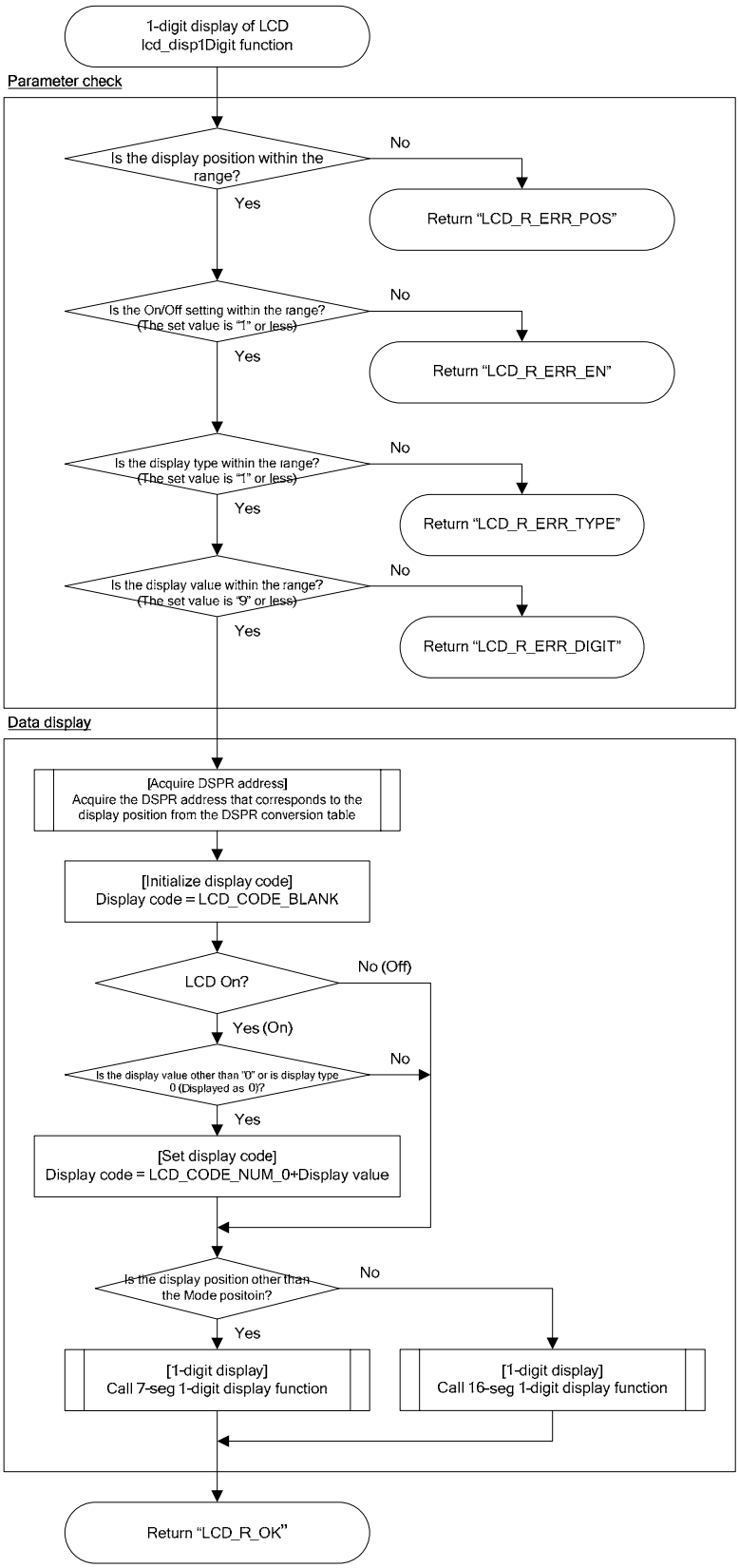
This function displays figures into the 12 digits (by 8-segment characters) on the lower part of the LCD panel. It generates font data for display from numeric data and then displays it on the LCD.

Function name:	void lcd_dispMain(signed long num)
Arguments:	signed long num … Display value
Return values:	None
Processing:	

3.8.5.8. lcd_disp1Digit Function

This function displays data onto one digit of the LCD panel. The function sets the digit position, LCD On/Off, and zero suppression enable/disable. It then generates font data for display from numeric data and writes display data to the target digit to display the data.

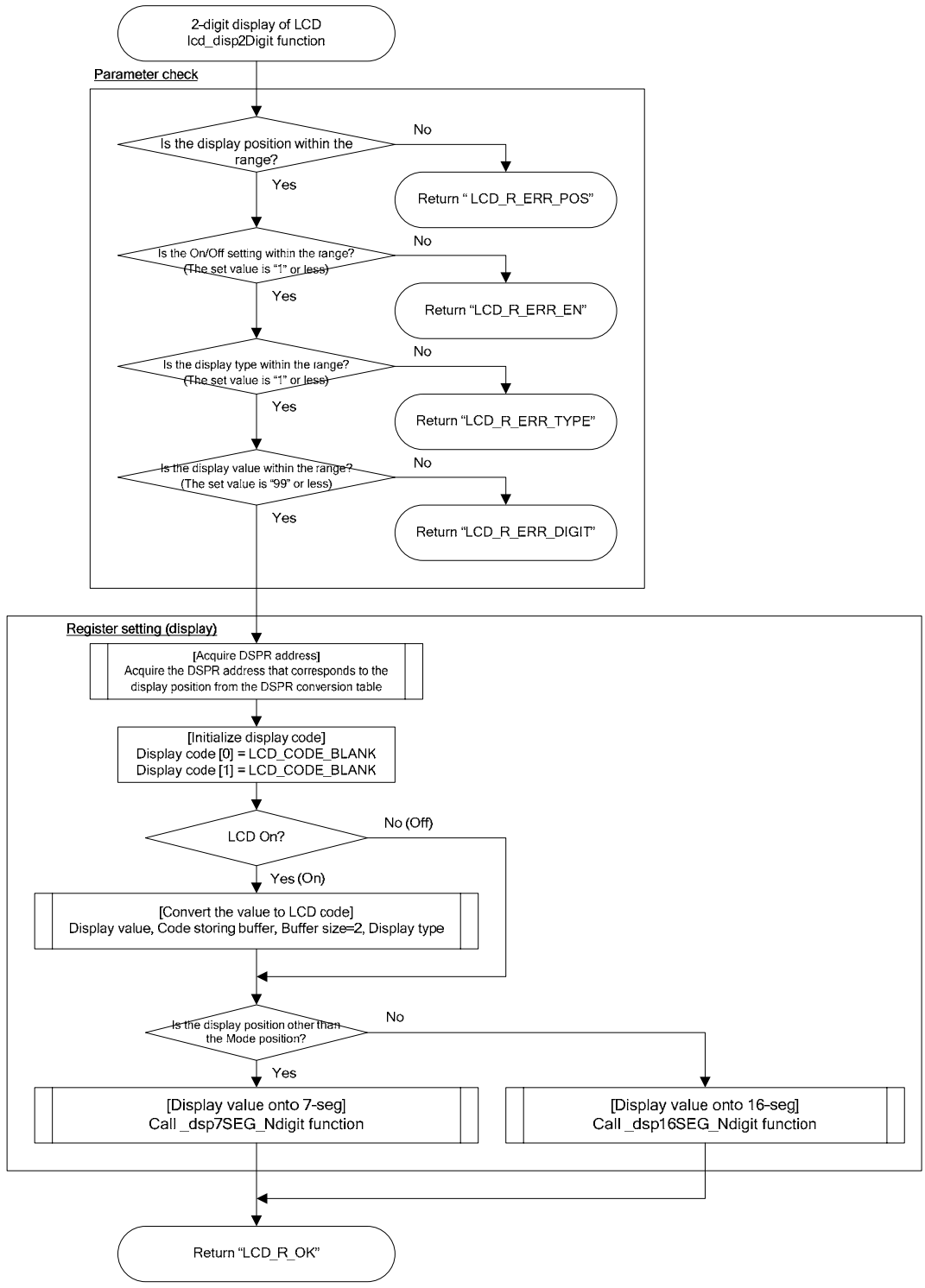
Function name:	int lcd_disp1Digit(unsigned char pos, unsigned char en, unsigned char type, unsigned char digit)
Arguments:	unsigned int pos … Display position 1st digit of the 12 digits on the lower section: LCD_POS_MAIN_01(=0x00) 2nd digit of the 12 digits on the lower section: LCD_POS_MAIN_02(=0x01) 3rd digit of the 12 digits on the lower section: LCD_POS_MAIN_03(=0x02) 4th digit of the 12 digits on the lower section: LCD_POS_MAIN_04(=0x03) 5th digit of the 12 digits on the lower section: LCD_POS_MAIN_05(=0x04) 6th digit of the 12 digits on the lower section: LCD_POS_MAIN_06(=0x05) 7th digit of the 12 digits on the lower section: LCD_POS_MAIN_07(=0x06) 8th digit of the 12 digits on the lower section: LCD_POS_MAIN_08(=0x07) 9th digit of the 12 digits on the lower section: LCD_POS_MAIN_09(=0x08) 10th digit of the 12 digits on the lower section: LCD_POS_MAIN_10(=0x09) 11th digit of the 12 digits on the lower section: LCD_POS_MAIN_11(=0x0A) 12th digit of the 12 digits on the lower section: LCD_POS_MAIN_12(=0x0B) Ones digit of second: LCD_POS_SEC_01(=0x10) Tens digit of second: LCD_POS_SEC_10(=0x11) Ones digit of minute: LCD_POS_MIN_01(=0x20) Tens digit of minute: LCD_POS_MIN_10(=0x21) Ones digit of hour: LCD_POS_HOUR_01(=0x30) Tens digit of hour: LCD_POS_HOUR_10(=0x31) 1st digit of Mode (upper-left four digits on the upper section): LCD_POS_MODE_01(=0x40) 2nd digit of Mode (upper-left four digits on the upper section): LCD_POS_MODE_02(=0x41) 3rd digit of Mode (upper-left four digits on the upper section): LCD_POS_MODE_03(=0x42) 4th digit of Mode (upper-left four digits on the upper section): LCD_POS_MODE_04(=0x43) unsigned char en … On/Off LCD Off: LCD_TURNOFF(=0) LCD On: LCD_TURNON(=1) unsigned char type … Display type Display a value of 0 as “0”: LCD_TYPE_0_DISP(=0) Display a value of 0 as “blank”: LCD_TYPE_0_BLANK(=1) unsigned char digit. … Display value(0 to 9)
Return values:	int Display succeeded: LCD_R_OK(=0) Display position setting is outside the range: LCD_R_ERR_POS(=-7) On/Off setting is outside the range: LCD_R_ERR_EN(=-8) Display type is outside the range: LCD_R_ERR_TYPE(=-9) Display value is outside the range: LCD_R_ERR_DIGIT(=-10)
Processing:	See next page.


1-digit display of LCD
lcd_disp1Digit function
Parameter check
Is the display position within the range?
No
Return "LCD_R_ERR_POS"
Yes
Is the On/Off setting within the range? (The set value is "1" or less)
No
Return "LCD_R_ERR_EN"
Yes
Is the display type within the range? (The set value is "1" or less)
No
Return "LCD_R_ERR_TYPE"
Yes
Is the display value within the range? (The set value is "9" or less)
No
Return "LCD_R_ERR_DIGIT"
Yes
Data display
[Acquire DSPR address]
Acquire the DSPR address that corresponds to the display position from the DSPR conversion table
[Initialize display code]
Display code = LCD_CODE_BLANK
LCD On?
No (Off)
Yes (On)
Is the display value other than "0" or is display type 0 (Displayed as 0)?
No
Yes
[Set display code]
Display code = LCD_CODE_NUM_0+Display value
Is the display position other than the Mode positoin?
No
Yes
[1-digit display]
Call 7-seg 1-digit display function
[1-digit display]
Call 16-seg 1-digit display function
Return "LCD_R_OK"

3.8.5.9. lcd_disp2Digit Function

This function displays data onto two consecutive digits of the LCD panel. The function sets the digit position, LCD On/Off, and zero suppression enable/disable for upper digits. It then generates font data for display from 2-digit numeric data and writes display data to the two digits to display the data.

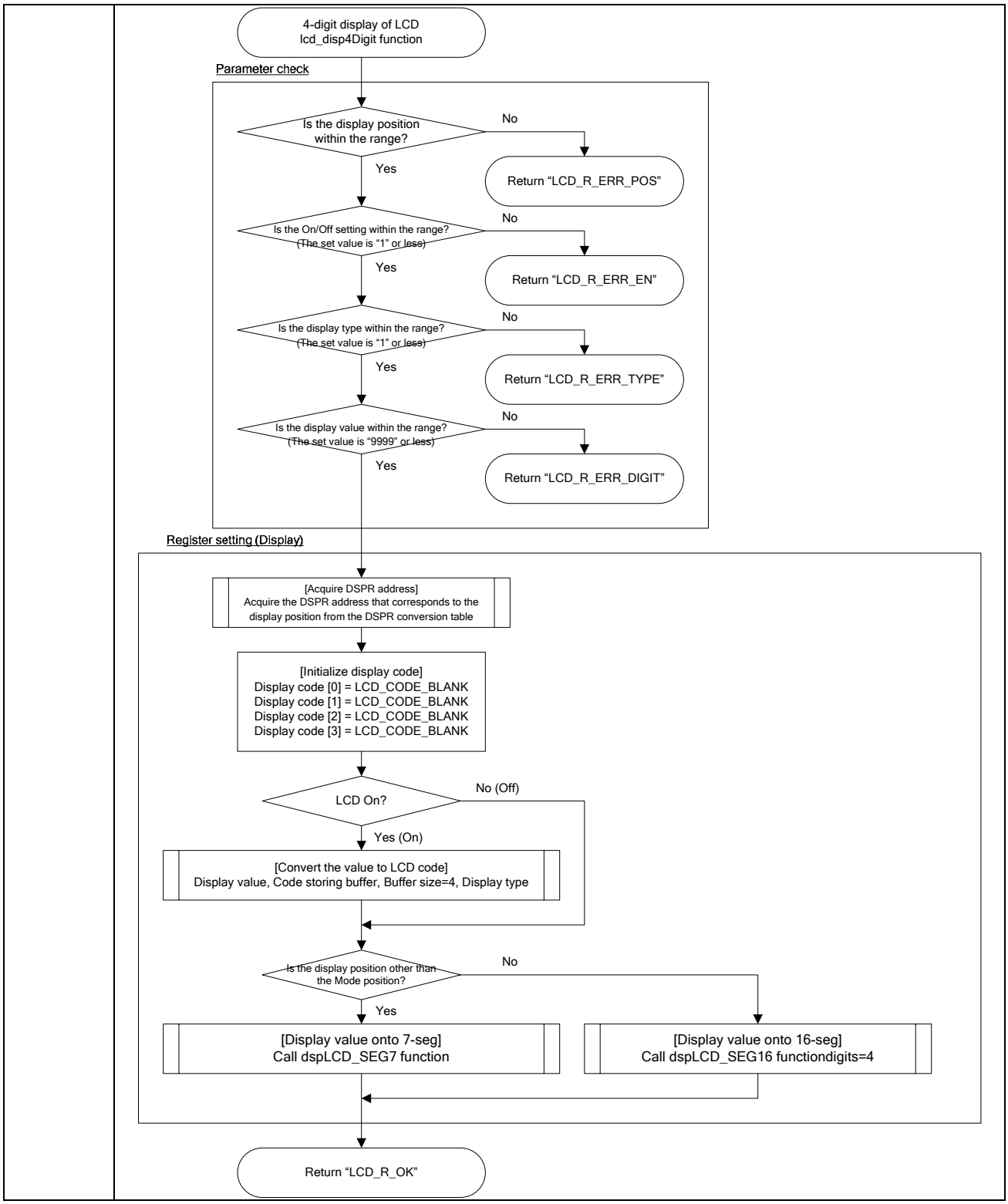
Function name:	int lcd_disp2Digit(unsigned char pos, unsigned char en, unsigned char type, unsigned char digit)
Arguments:	unsigned int pos … Display position 1st and 2nd digits of the 12 digits on the lower section: LCD_POS_MAIN_01(=0x00) 3rd and 4th digits of the 12 digits on the lower section: LCD_POS_MAIN_03(=0x02) 5th and 6th digits of the 12 digits on the lower section: LCD_POS_MAIN_05(=0x04) 7th and 8th digits of the 12 digits on the lower section: LCD_POS_MAIN_07(=0x06) 9th and 10th digits of the 12 digits on the lower section: LCD_POS_MAIN_09(=0x08) 11th and 12th digits of the 12 digits on the lower section: LCD_POS_MAIN_11(=0x0A) Second: LCD_POS_SEC(=0x10) Minute: LCD_POS_MIN(=0x20) Hour: LCD_POS_HOUR(=0x30) Lower 2 digits for mode display (upper-left 4 digits of the upper section): LCD_POS_MODE_LOW(=0x40) Upper2 digits for mode display (upper-left 4 digits of the upper section): LCD_POS_MODE_HIGH(=0x42) unsigned char en … On/Off LCD Off: LCD_TURNOFF(=0) LCD On: LCD_TURNON(=1) unsigned char type … Display type Display a high-order digit 0 in the two digits as a “0”: LCD_TYPE_0_DISP(=0) Display a high-order digit 0 in the two digits as a “blank”: LCD_TYPE_0_BLANK(=1) unsigned char digit. … Display value(0 to 99)
Return values:	int Display succeeded: LCD_R_OK(=0) Display position setting is outside the range: LCD_R_ERR_POS(=-7) On/Off setting is outside the range: LCD_R_ERR_EN(=-8) Display type is outside the range: LCD_R_ERR_TYPE(=-9) Display value is outside the range: LCD_R_ERR_DIGIT(=-10)
Processing:	See next page.


2-digit display of LCD
lcd_disp2Digit function
Parameter check
Is the display position within the range?
No
Return " LCD_R_ERR_POS"
Yes
Is the On/Off setting within the range? (The set value is "1" or less)
No
Return "LCD_R_ERR_EN"
Yes
Is the display type within the range? (The set value is "1" or less)
No
Return "LCD_R_ERR_TYPE"
Yes
Is the display value within the range? (The set value is "99" or less)
No
Return "LCD_R_ERR_DIGIT"
Yes
Register setting (display)
[Acquire DSPR address]
Acquire the DSPR address that corresponds to the display position from the DSPR conversion table
[Initialize display code]
Display code [0] = LCD_CODE_BLANK
Display code [1] = LCD_CODE_BLANK
LCD On?
No (Off)
Yes (On)
[Convert the value to LCD code]
Display value, Code storing buffer, Buffer size=2, Display type
Is the display position other than the Mode position?
No
Yes
[Display value onto 7-seg]
Call _dsp7SEG_Ndigit function
[Display value onto 16-seg]
Call _dsp16SEG_Ndigit function
Return "LCD_R_OK"

3.8.5.10. lcd_disp4Digit Function

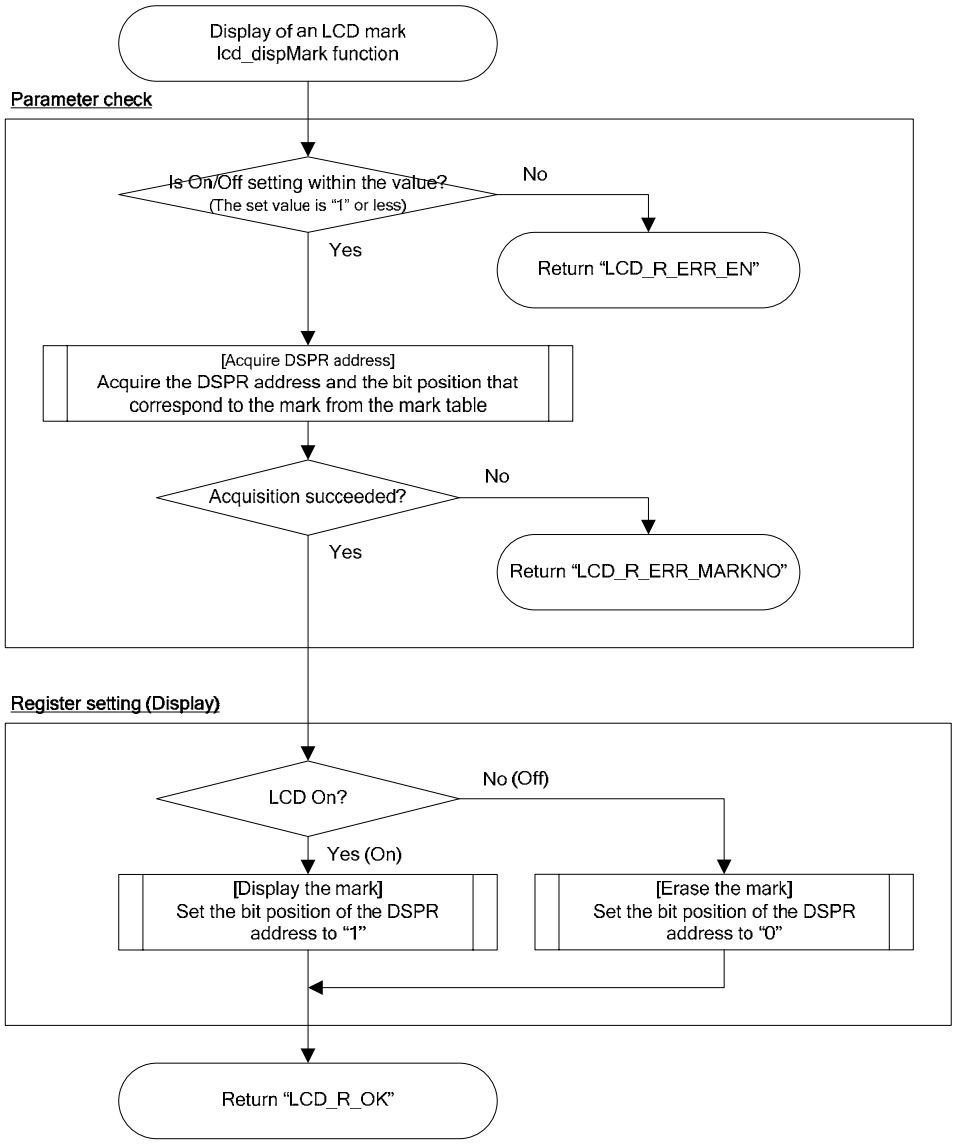
This function displays data onto four consecutive digits of the LCD panel. The function sets the digit position, LCD On/Off, and zero suppression enable/disable for high-order digits. It then generates font data for display from 4-digit numeric data and writes display data to the four digits to display the data.

Function name:	int lcd_disp4Digit(unsigned char pos, unsigned char en, unsigned char type, unsigned short digit)
Arguments:	unsigned int pos … Display position 1st to 4th digits of the 12 digits on the lower section: LCD_POS_MAIN_01(=0x00) 5th to 8th digits of the 12 digits on the lower section: LCD_POS_MAIN_05(=0x04) 9th to 12th digits of the 12 digits on the lower section: LCD_POS_MAIN_09(=0x08) 1st to 4th digits for mode display (upper-left 4 digits on the upper section): LCD_POS_MODE_01(=0x40) unsigned char en … On/Off LCD Off: LCD_TURNOFF(=0) LCD On: LCD_TURNON(=1) unsigned char type … Display type Display 0s of the high-order digits as “0s”: LCD_TYPE_0_DISP(=0) Display 0s of the high-order digits as “blanks”: LCD_TYPE_0_BLANK(=1) unsigned short digit. … Display value(0 to 9999)
Return values:	int Display succeeded: LCD_R_OK(=0) Display position setting is outside the range: LCD_R_ERR_POS(=-7) On/Off setting is outside the range: LCD_R_ERR_EN(=-8) Display type is outside the range: LCD_R_ERR_TYPE(=-9) Display value is outside the range: LCD_R_ERR_DIGIT(=-10)
Processing:	See next page.


4-digit display of LCD
lcd_disp4Digit function
Parameter check
Is the display position within the range?
No
Return "LCD_R_ERR_POS"
Yes
Is the On/Off setting within the range? (The set value is "1" or less)
No
Return "LCD_R_ERR_EN"
Yes
Is the display type within the range? (The set value is "1" or less)
No
Return "LCD_R_ERR_TYPE"
Yes
Is the display value within the range? (The set value is "9999" or less)
No
Return "LCD_R_ERR_DIGIT"
Yes
Register setting (Display)
[Acquire DSPR address]
Acquire the DSPR address that corresponds to the display position from the DSPR conversion table
[Initialize display code]
Display code [0] = LCD_CODE_BLANK
Display code [1] = LCD_CODE_BLANK
Display code [2] = LCD_CODE_BLANK
Display code [3] = LCD_CODE_BLANK
LCD On?
No (Off)
Yes (On)
[Convert the value to LCD code]
Display value, Code storing buffer, Buffer size=4, Display type
Is the display position other than the Mode position?
No
Yes
[Display value onto 7-seg]
Call dspLCD_SEG7 function
[Display value onto 16-seg]
Call dspLCD_SEG16 functiondigits=4
Return "LCD_R_OK"

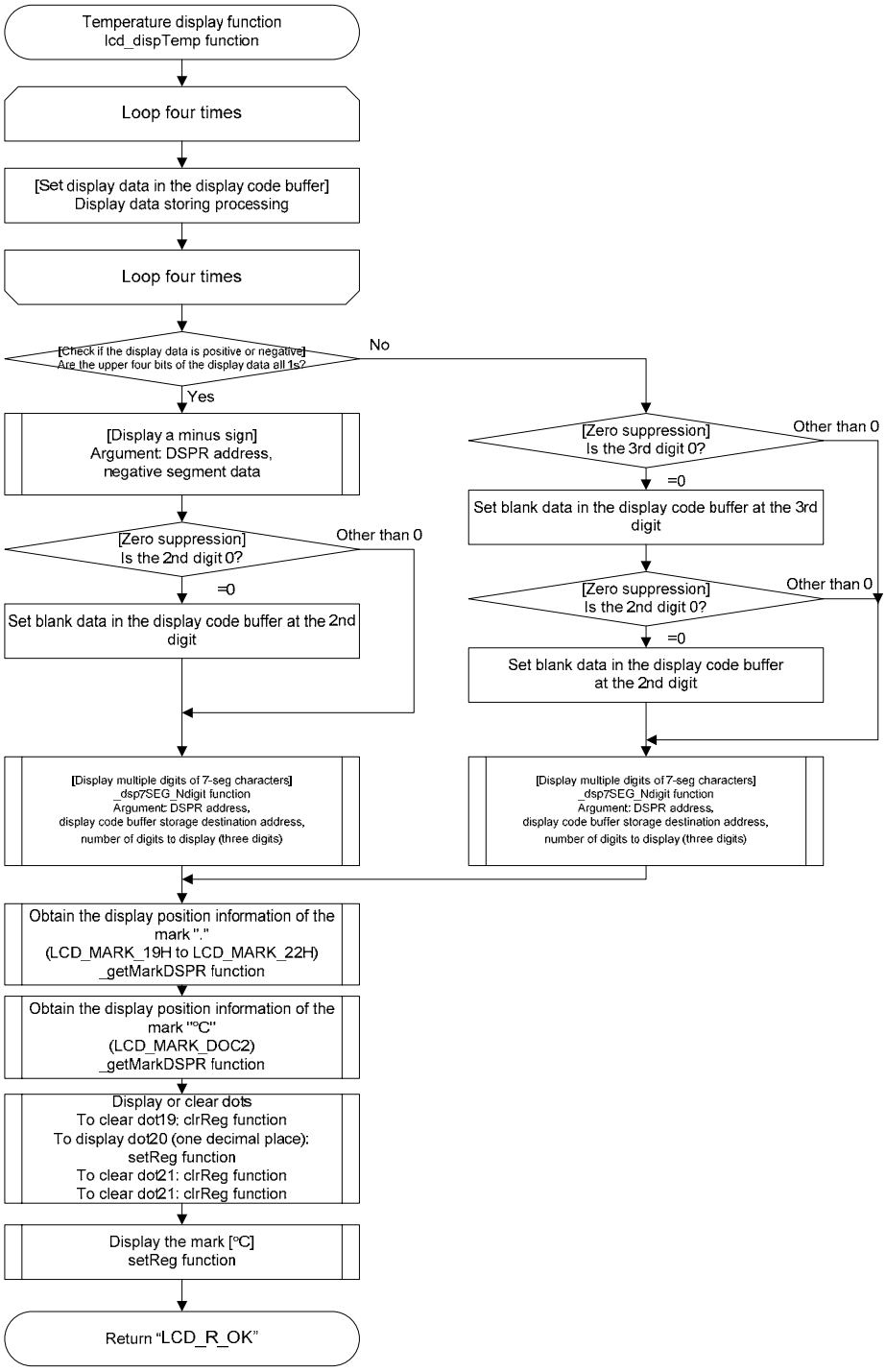
3.8.5.11. lcd_dispMark Function

This function displays one of the LCD marks. It turns on or off the target mark based on the of the mark position setting and On/Off setting.

Function name:	int lcd_dispMark(unsigned char markNo, unsigned char en)
Arguments:	unsigned char markNo … Mark No. to be displayed See Table 3-32, "Constants for Mark Codes." unsigned char en … On/Off Off: LCD_TURNOFF(=0) On: LCD_TURNON(=1)
Return values:	int Display succeeded: LCD_R_OK(=0) On/Off setting is outside the range: LCD_R_ERR_EN(=-8) Mark No. is outside the range: LCD_R_ERR_MARKNO(=-11)
Processing:	

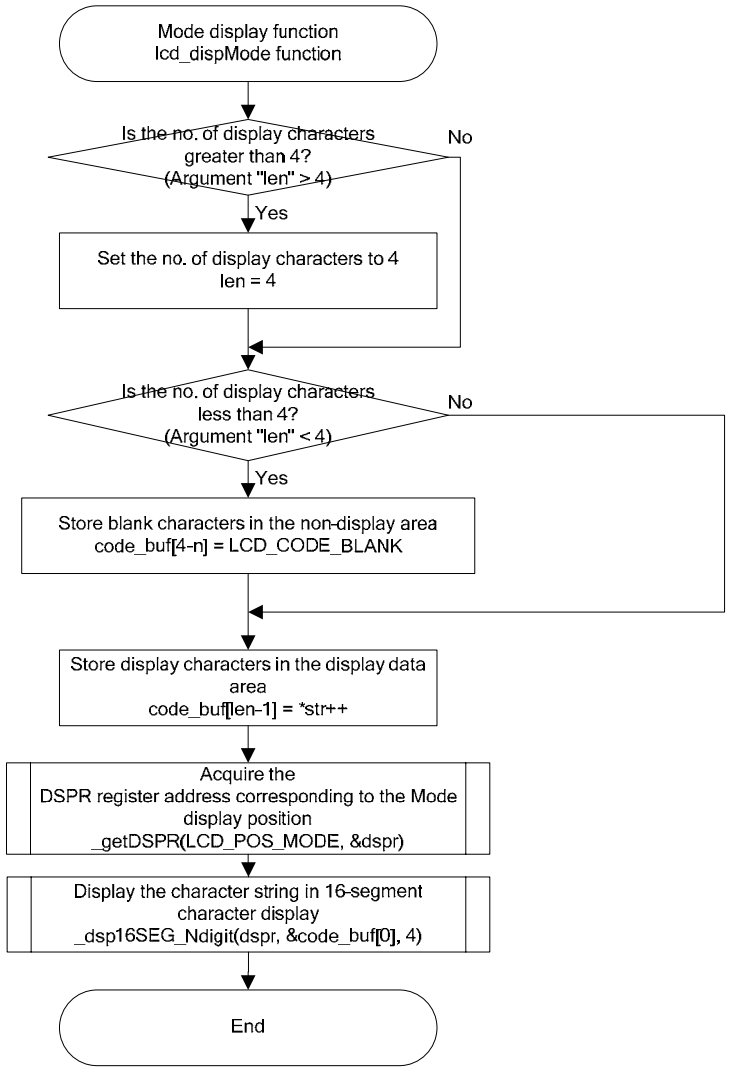
3.8.5.12. lcd_dispTemp Function

This function displays the temperature value (XXX.X[°C]).

Function name:	int lcd_dispTemp(short digit)
Arguments:	short digit … Temperature value (9999 to -999) * Specify the value obtained by multiplying the actual temperature by 10 for the temperature value. * Temperatures are displayed as “999.9°C” to “-99.9°C”. * The position of the decimal point is fixed to XXX.X.
Return values:	int Display succeeded: LCD_R_OK(=0) Display value is outside the range: LCD_R_ERR_DIGIT(=-10) Mark No. is outside the range: LCD_R_ERR_MARKNO(=-11)
Processing:	

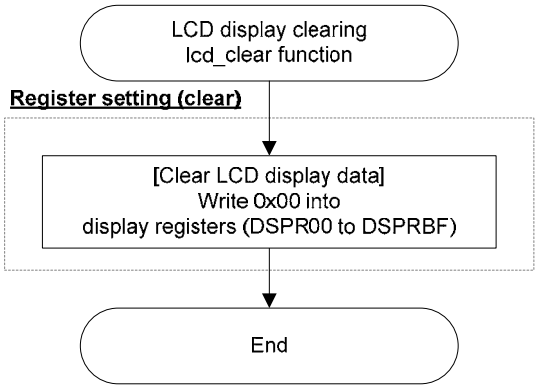
3.8.5.13. lcd_dispMode Function

This function displays a character string (numerals, alphabets, minus sign, blanks) on the "Mode of LCD" display section (the four digits on the upper left of the LCD panel) of the LCD.

Function name:	void lcd_dispMode(char* str, unsigned char len)
Arguments:	char* str … Character string to display For displayable characters, see Table 3-31, "Constants for LCD Codes." Characters that are not in the table are displayed as blanks if specified. unsigned char len … Character count (0 to 4) If 0 is specified, four blank characters are displayed. If any value greater than 4 is specified, up to the first four characters will be displayed.
Return values:	None
Processing:	

3.8.5.14. lcd_dispClear Function

This function clears the display on the LCD panel.

Function name:	void lcd_clear(void)
Arguments:	None
Return values:	None
Processing:	

3.9. Key Read-In Module

3.9.1. Overview of Functions

The key read-in module controls key read-in operation.
Key read-in is achieved by APIs that perform initialization (input mode), read-in start/stop, and key status acquisition.

3.9.2. Key Read-In Timing Diagram

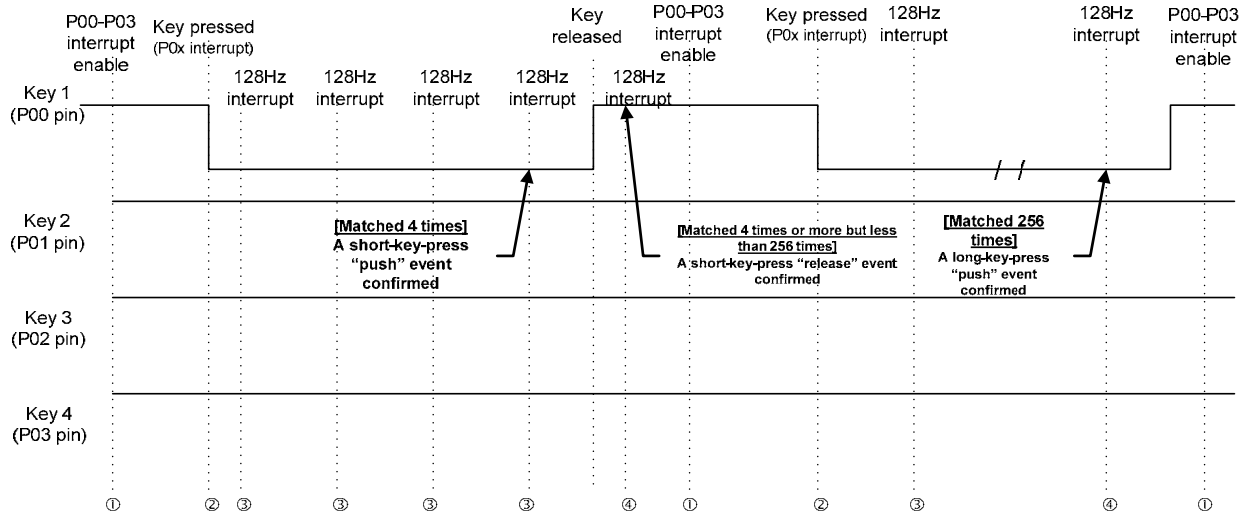


Figure 3-1 Key Read-In Timing Diagram

① Enable interrupts to port P00–P03 pins.
Specify falling edge interrupt for these ports as the interrupt generation condition.

② Interrupt occurs (P0xINT: x=0-3):
P0xINT occurs when a key is pressed. Disables interrupts to other ports.
Chattering absorption timer 128Hz interrupt starts (approx. 7.8 ms).
This 128Hz interrupt is generated by the time base counter. Note that the interval of this interrupt is not synchronous with P0xINT.

③ Absorb chattering (INT128Hz: approx. 7.8 ms):
Key read-in is input to the port until matching of the same key is detected four times or more in a row.
When matching of the same key is detected four times in a row (a short-key-press "push" event is confirmed): Generates a key status of a short-key-press "push" and continues the processing of ④.
If, after matching of the same key is found less than four times, the key is released, processing returns to ①.

④ Key event confirmed.
If, during matching of the same key occurring four times or more but less than 256 times, the key is released (a short-key-press "release" event is confirmed): Generates a key status of a short-key-press "release" and returns to the processing of ①.
When matching of the same key is detected 256 times (when holding it down for approx. 2 seconds) (a long-key-press event is confirmed): Generates a key status of a long key press and continues processing until the key is released. Processing returns to ① when the key is released.

3.9.3. List of APIs

The following table lists the key read-in module APIs.

Table 3-33 Key Read-In Module APIs

Function name	Description
key_init function	Sets key read-in port (pull-up input, interrupt at the falling edge, no sampling).
key_start function	Starts key read-in processing.
key_stop function	Stops key read-in processing.
key_getEvent function	Acquires a key event.

3.9.4. List of Constants

The following table lists the constants used in the key read-in module.

Table 3-34 Constants for Return Values

Constant name	Defined value	Description
NO_EVENT	0x00	No event has been generated
KEY1_SHORT_PUSH _EVENT	0x01	A short S1 key press “push” event generated
KEY2_SHORT_PUSH_EVENT	0x02	A short S2 key press “push” event generated
KEY3_SHORT_PUSH_EVENT	0x03	A short S3 key press “push” event generated
KEY4_SHORT_PUSH_EVENT	0x04	A short S4 key press “push” event generated
KEY1_SHORT_RELEASE_EVENT	0x11	A short S1 key press “release” event generated
KEY2_SHORT_RELEASE_EVENT	0x12	A short S2 key press “release” event generated
KEY3_SHORT_RELEASE_EVENT	0x13	A short S3 key press “release” event generated
KEY4_SHORT_RELEASE_EVENT	0x14	A short S4 key press “release” event generated
KEY1_LONG_EVENT	0x21	A long S1 key press event generated
KEY2_LONG_EVENT	0x22	A long S2 key press event generated
KEY3_LONG_EVENT	0x23	A long S3 key press event generated
KEY4_LONG_EVENT	0x24	A long S4 key press event generated

3.9.5. List of Variables

The following table shows the variables used in the key read-in module.

Table 3-35 Variables

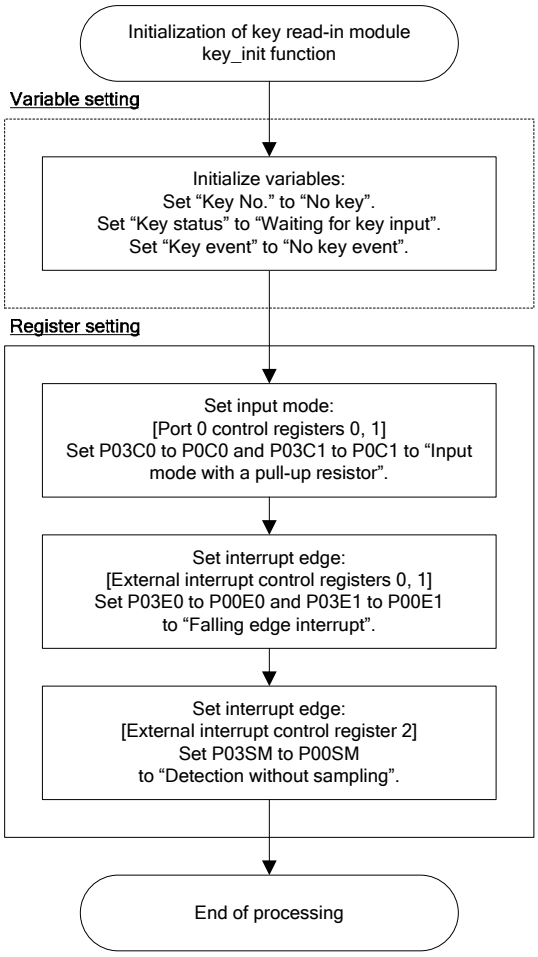
Variable name	Initial value	Description
static unsigned _sKeyNo	0	Variable used to manage the current key status
static unsigned _sKeyStatus	0	Variable used to manage the key read-in status
static unsigned _sKeyEvent	0	Variable used to manage the “confirmed” status
static unsigned _sKeyOnCnt	0	Variable used to manage the number of times a key was pressed

3.9.6. API Details

This section describes the details of the key read-in module.

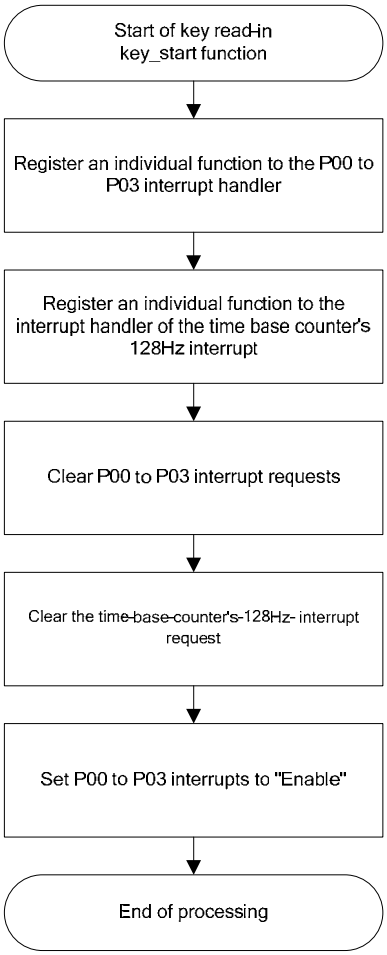
3.9.6.1. key_init Function

This function initializes the key read-in module.

Function name:	void key_init(void)
Arguments:	None
Return values:	None
Processing:	

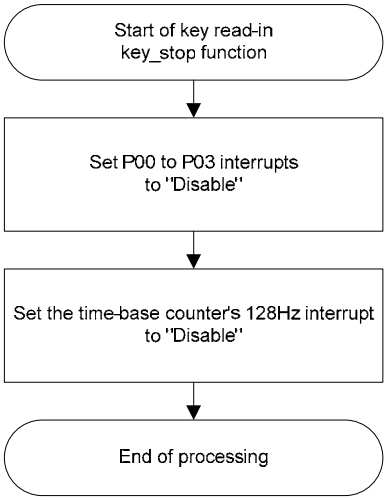
3.9.6.2. key_start Function

This function starts key-read in operation.

Function name:	void key_start(void)
Arguments:	None
Return values:	None
Processing:	

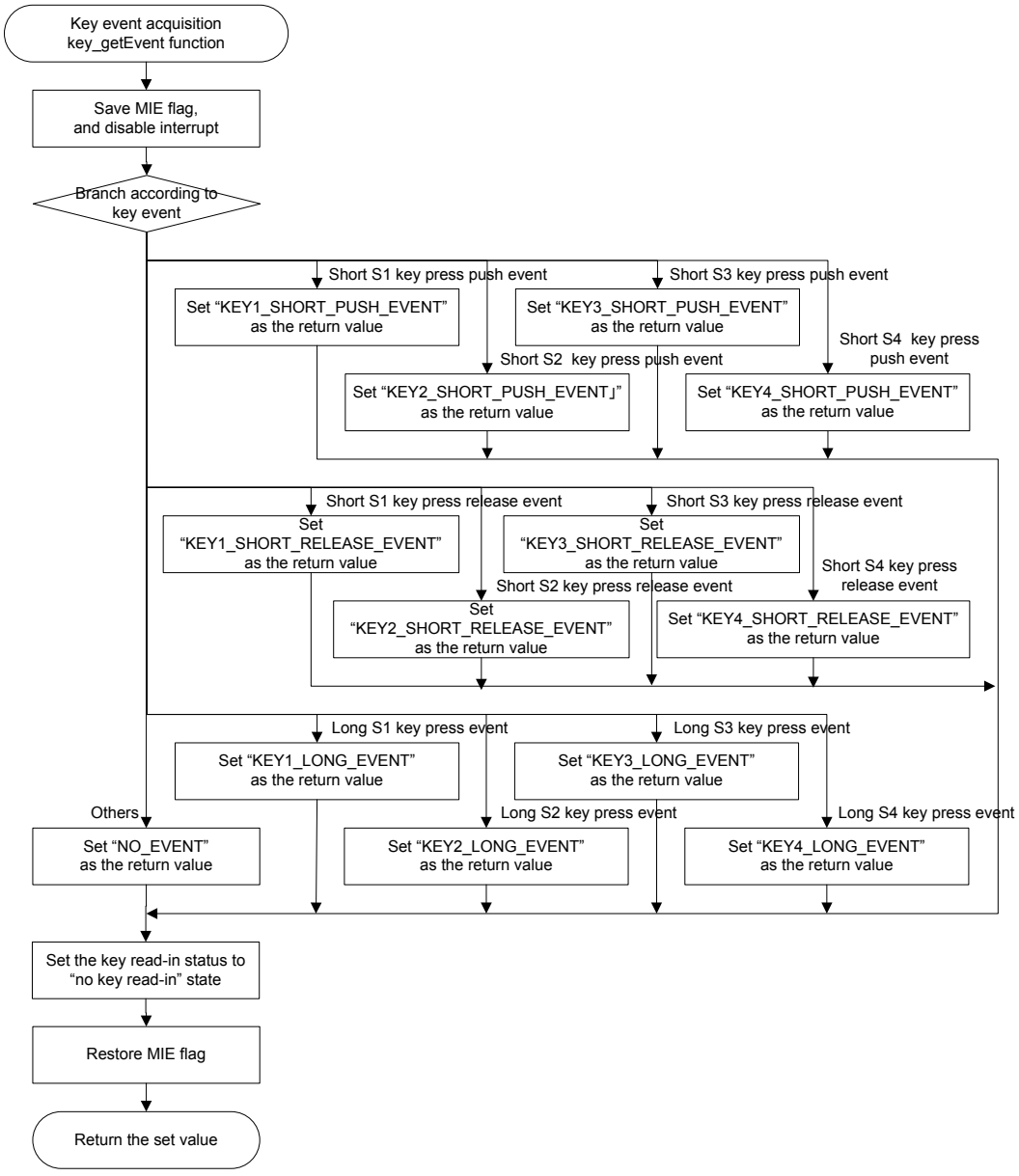
3.9.6.3. key_stop Function

This function stops key read-in operation.

Function name:	void key_stop(void)
Arguments:	None
Return values:	None
Processing:	

3.9.6.4. key_getEvent Function

This function acquires key events.

Function name:	unsigned char key_getEvent(void)
Arguments:	None
Return values:	unsigned cahr Short S1 key press push event: KEY1_SHORT_PUSH_EVENT Short S2 key press push event: KEY2_SHORT_PUSH_EVENT Short S3 key press push event: KEY3_SHORT_PUSH_EVENT Short S4 key press push event: KEY4_SHORT_PUSH_EVENT Short S1 key press release event: KEY1_SHORT_RELEASE_EVENT Short S2 key press release event: KEY2_SHORT_RELEASE_EVENT Short S3 key press release event: KEY3_SHORT_RELEASE_EVENT Short S4 key press release event: KEY4_SHORT_RELEASE_EVENT Long S1 key press event: KEY1_LONG_EVENT Long S2 key press event: KEY2_LONG_EVENT Long S3 key press event: KEY3_LONG_EVENT Long S3 key press event: KEY4_LONG_EVENT No key event has been confirmed for any key: NO_EVENT
Processing:	

3.10. Melody Module

3.10.1. Overview of Functions

The melody module controls the melody driver of the MCU.
This module can control the following: Initialization (melody/buzzer output stop, port secondary function setting), melody output start/stop, melody output continuation, melody output interrupt request acquisition and clear, and buzzer output start/stop.
Note: The melody module is available only in the ML610Q431/Q432. The ML610Q411/Q412/Q415 is not provided with the melody function.

3.10.2. List of APIs

The following table lists the melody module APIs.

Table 3-36 Melody Module APIs

Function name	Description
melody_init function	Initializes the melody control module (stops melody/buzzer output; port secondary function setting).
melody_start function	Starts melody output.
melody_stop function	Stops melody output.
melody_continue function	Performs melody output continuation processing.
melody_checkoutput function	Checks the presence or absence of melody output.
melody_checkIRQ function	Checks the presence or absence of a melody output interrupt request.
melody_clearIRQ function	Clears melody output interrupt request.
buzzer_start function	Starts buzzer output.
buzzer_stop function	Stops buzzer output.

3.10.3. List of Constants

The following table lists the constants used in the melody module.

Table 3-37 Melody Module Constants

Constant name	Defined value	Description
MELODY_TEMPO_480	1	♪= 480
MELODY_TEMPO_320	2	♪=320
MELODY_TEMPO_240	3	♪=240
MELODY_TEMPO_192	4	♪=192
MELODY_TEMPO_160	5	♪=160
MELODY_TEMPO_137	6	♪≅137
MELODY_TEMPO_120	7	♪=120
MELODY_TEMPO_107	8	♪≅107
MELODY_TEMPO_96	9	♪=96
MELODY_TEMPO_87	10	♪≅87
MELODY_TEMPO_80	11	♪=80
MELODY_TEMPO_74	12	♪≅74
MELODY_TEMPO_69	13	♪≅69
MELODY_TEMPO_64	14	♪=64
MELODY_TEMPO_60	15	♪=60
BUZZER_TYPE0	0	Specifies intermittent sound 1 as the buzzer output sound.
BUZZER_TYPE1	1	Specifies intermittent sound 2 as the buzzer output sound.
BUZZER_TYPE2	2	Specifies single sound as the buzzer output sound.
BUZZER_TYPE3	3	Specifies continuous sound as the buzzer output.
BUZZER_FREQ_4096HZ	0	Specifies 4.096kHz as the buzzer output frequency.
BUZZER_FREQ_2048HZ	1	Specifies 2.048 kHz as the buzzer output frequency.
BUZZER_FREQ_1365HZ	2	Specifies 1.365 kHz as the buzzer output frequency.
BUZZER_FREQ_1024HZ	3	Specifies 1.024 kHz as the buzzer output frequency.
BUZZER_FREQ_819HZ	4	Specifies 819 kHz as the buzzer output frequency.
BUZZER_FREQ_683HZ	5	Specifies 683 kHz as the buzzer output frequency.
BUZZER_FREQ_585HZ	6	Specifies 1585 kHz as the buzzer output frequency.
BUZZER_FREQ_512HZ	7	Specifies 512 kHz as the buzzer output frequency.
BUZZER_DUTY_1_16	1	Specifies 1/16 duty as the buzzer output duty.
BUZZER_DUTY_2_16	2	Specifies 2/16 duty as the buzzer output duty.
BUZZER_DUTY_3_16	3	Specifies 3/16 duty as the buzzer output duty.
BUZZER_DUTY_4_16	4	Specifies 4/16 duty as the buzzer output duty.
BUZZER_DUTY_5_16	5	Specifies 5/16 duty as the buzzer output duty.
BUZZER_DUTY_6_16	6	Specifies 6/16 duty as the buzzer output duty.
BUZZER_DUTY_7_16	7	Specifies 7/16 duty as the buzzer output duty.
BUZZER_DUTY_8_16	8	Specifies 8/16 duty as the buzzer output duty.
BUZZER_DUTY_9_16	9	Specifies 9/16 duty as the buzzer output duty.
BUZZER_DUTY_10_16	10	Specifies 10/16 duty as the buzzer output duty.
BUZZER_DUTY_11_16	11	Specifies 11/16 duty as the buzzer output duty.
BUZZER_DUTY_12_16	12	Specifies 12/16 duty as the buzzer output duty.
BUZZER_DUTY_13_16	13	Specifies 13/16 duty as the buzzer output duty.
BUZZER_DUTY_14_16	14	Specifies 14/16 duty as the buzzer output duty.
BUZZER_DUTY_15_16	15	Specifies 15/16 duty as the buzzer output duty.
BUZZER_DUTY_16_16	16	Specifies 16/16 duty as the buzzer output duty.

Table 3-38 List of Constants for Return Values

Constant name	Defined value	Description
MELODY_R_OK	0	Processing succeeded.
MELODY_R_IRQ	1	Melody output interrupt request is present.
MELODY_R_NON_IRQ	0	No melody output complete interrupt request is present.
MELODY_R_OUTPUT	1	Melody is being output.
MELODY_R_STOP	0	Melody output is stopped.
BUZZER_R_OK	0	Processing succeeded.
BUZZER_R_ERR_TYPE	-1	Buzzer type is outside the range.
BUZZER_R_ERR_FREQUENCY	-2	Buzzer output frequency is outside the range.
BUZZER_R_ERR_DUTY	-3	Buzzer output duty is outside the range.

3.10.4. Structure

This section describes the structures used in the melody module.

■ Melody output data control parameters

```
typedef struct {
    unsigned short *    data;    // Pointer to the melody data storage area
    unsigned int        size;    // Melody data size (the number of note data items)
} tMelodyCtrlParam;
```

3.10.5. List of Variables

This section describes the variables used in the melody module.

Variable name	Initial value	Description
static tMelodyCtrlParam_gsCtrlParam	data: NULL size: 0	Variable used to manage melody output data

3.10.6. Melody Output Data Format

The following note data is generated continuously to create melody output data.

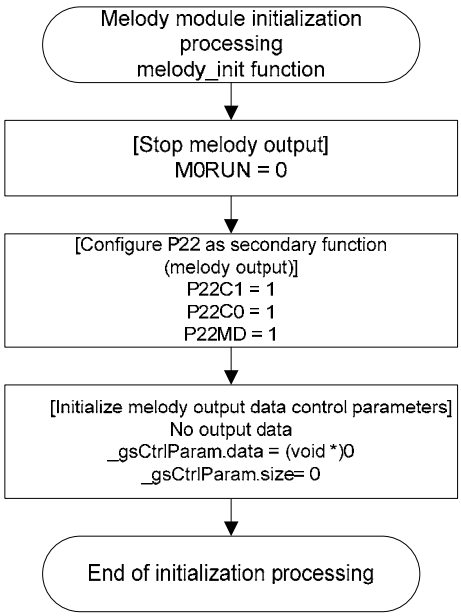
Note data (16-bit)	
Tone length code (8-bit)	Scale code (8-bit)

3.10.7. Details of APIs

This section describes details of the melody module APIs.

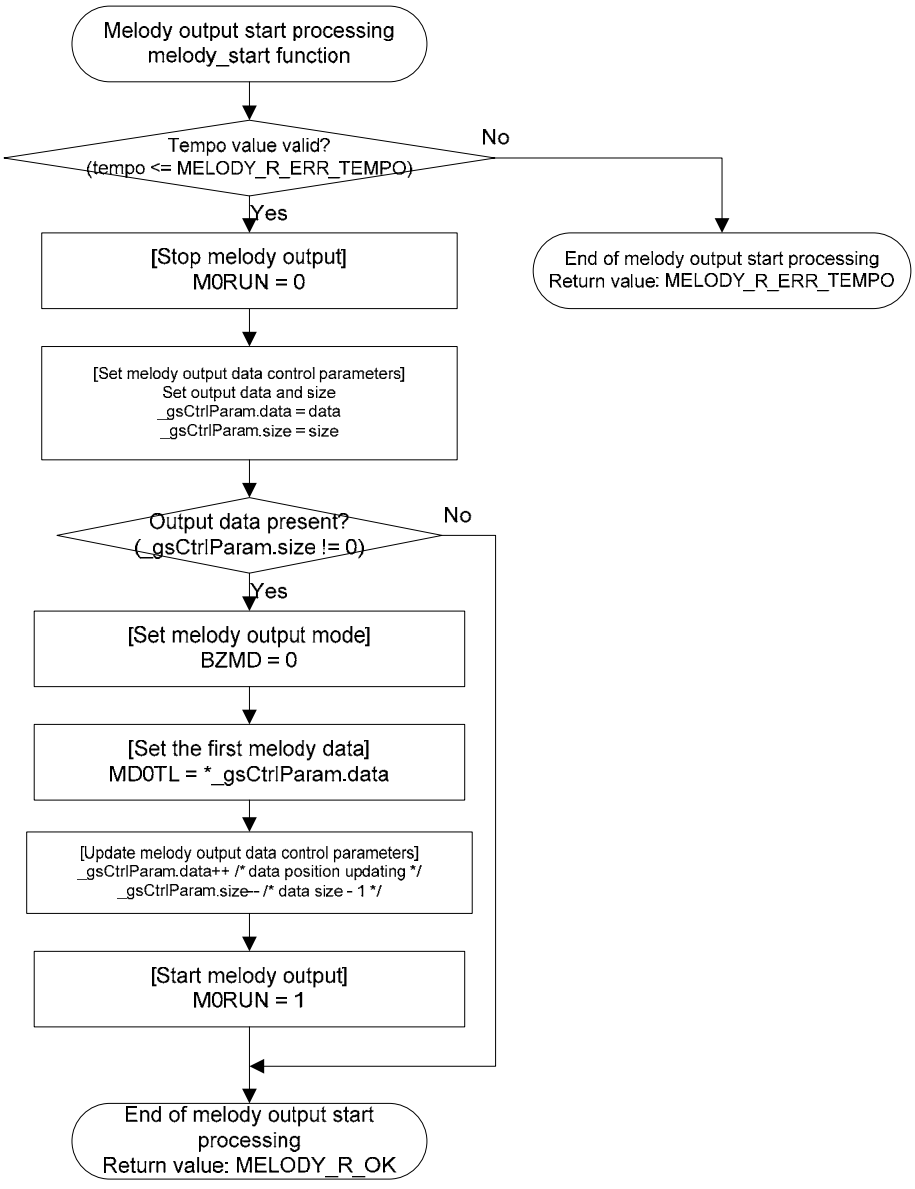
3.10.7.1. melody_init Function

This function initializes the melody module. In the initialization, melody/buzzer output of the MCU is stopped, port 2 (P22), which is used in melody output, is configured as its secondary function, and variables that controls melody output are initialized.

Function name:	void melody_init(void)
Arguments:	None
Return values:	None
Processing:	

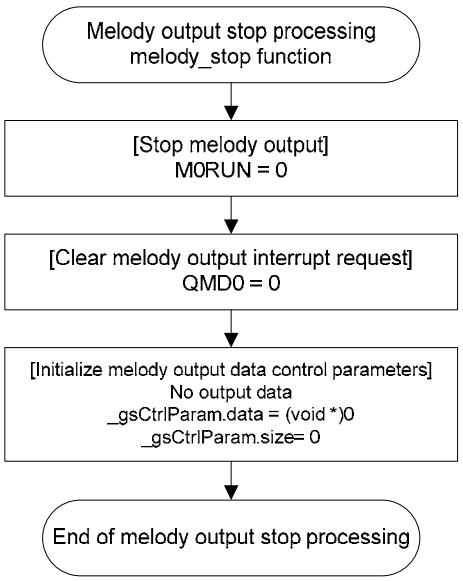
3.10.7.2. melody_start Function

This function performs melody output start processing. Melody output is started by setting values in the parameters that control melody output data and setting the first note data in the MCU. The second and the subsequent melody output (note data setting) is performed by the melody_continue.
If this function is called during melody output, the melody output being processed is canceled and new melody output processing is started.

Function name:	void melody_start(unsigned char tempo, unsigned short *data, unsigned int size)
Arguments:	unsigned char tempo … Specify tempo unsigned short* data … Pointer to the area that contains melody data unsigned int size …. Melody data size (the number of note data items)
Return values:	None
Processing:	

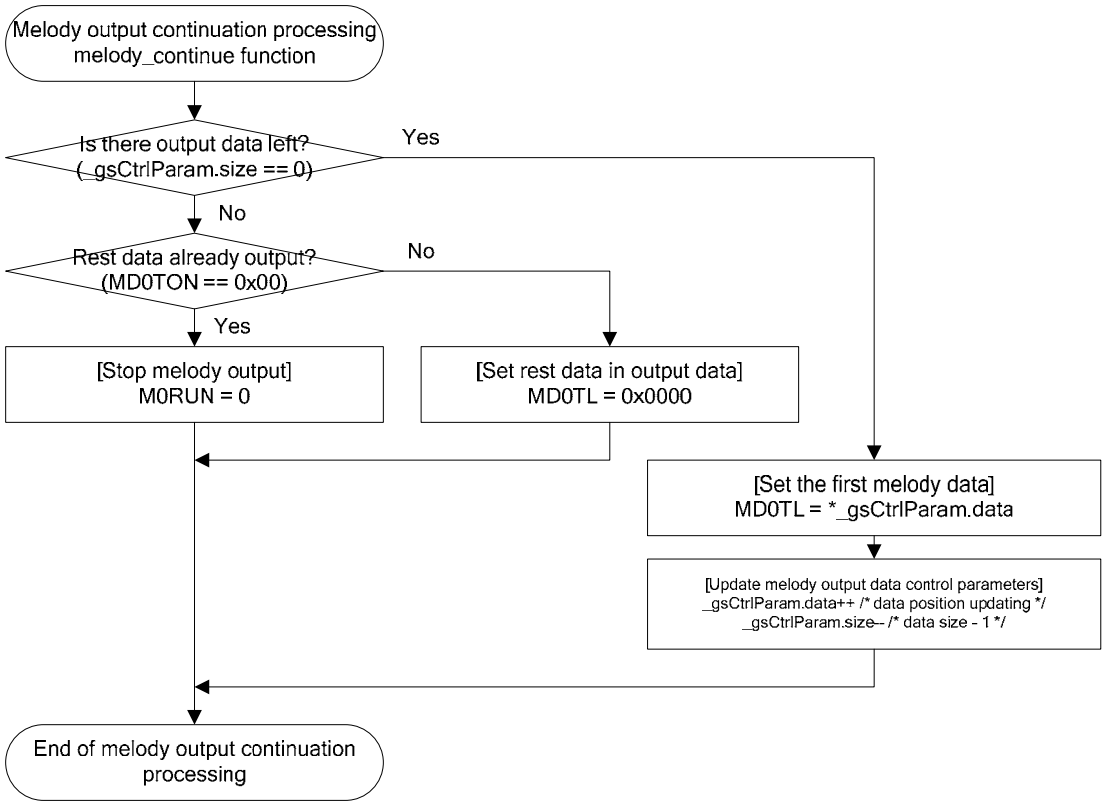
3.10.7.3. melody_stop Function

This function performs melody output stop processing. In the stop processing, melody output of the MCU is stopped, melody output interrupt request is cleared, and parameters that control melody output are initialized.

Function name:	void melody_stop(void)
Arguments:	None
Return values:	None
Processing:	

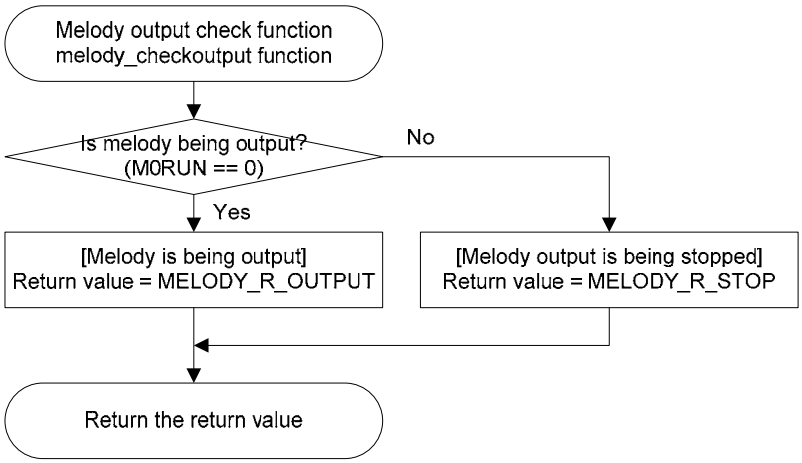
3.10.7.4. melody_continue Function

This function performs melody output continuation processing. If there is any data remaining, the next data is set in the MCU. When no data is left, rest data is set in the MCU. After the rest data is output, the melody output of the MCU is stopped.

Function name:	void melody_continue(void)
Arguments:	None
Return values:	None
Processing:	

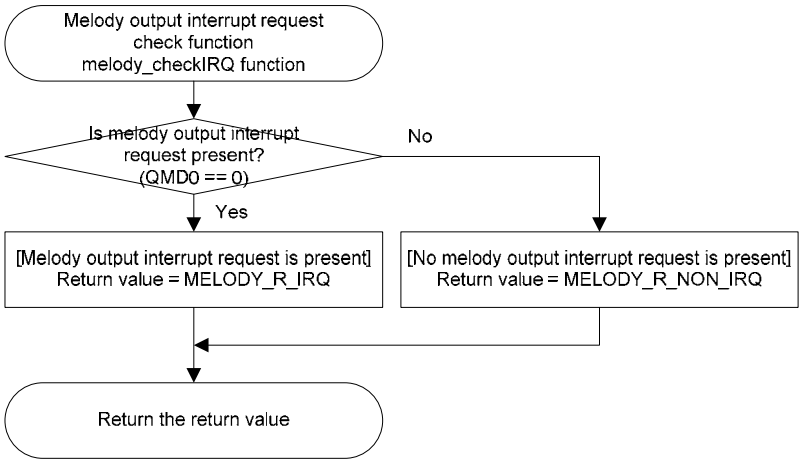
3.10.7.5. melody_checkoutput Function

This function checks the presence or absence of melody output.

Function name:	int melody_checkoutput(void)
Arguments:	None
Return values:	int MELODY_R_OUTPUT(=1): Melody is being output. MELODY_R_STOP(=0): Melody output is being stopped.
Processing:	

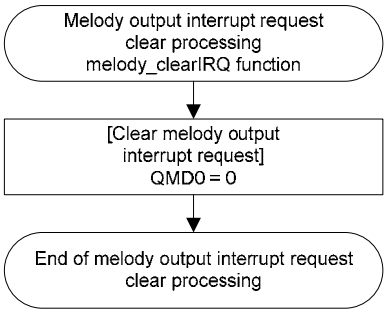
3.10.7.6. melody_checkIRQ Function

This function checks the presence or absence of the melody output interrupt request. Return values are only valid when the melody output interrupt of the MCU is not used.

Function name:	int melody_checkIRQ(void)
Arguments:	None
Return values:	int MELODY_R_IRQ(=1): Melody output interrupt request present MELODY_R_NON_IRQ(=0): No melody output interrupt request present
Processing:	

3.10.7.7. melody_clearIRQ Function

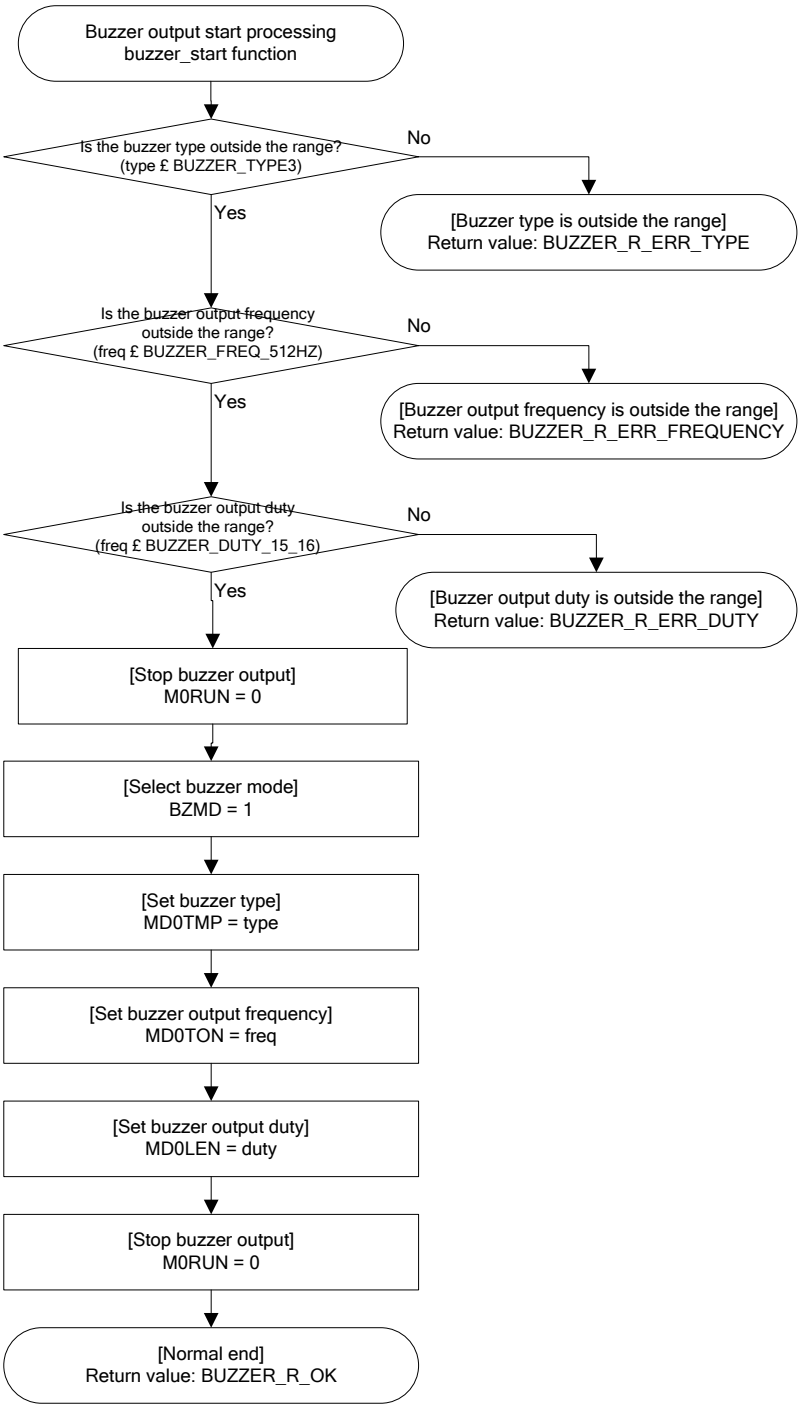
This function clears melody output interrupt request. Use this function only when the melody output interrupt of the MCU is not used.

Function name:	void melody_clearIRQ(void)
Arguments:	None
Return values:	int MELODY_R_IRQ(=1): Melody output interrupt request is present MELODY_R_NON_IRQ(=0): No melody output interrupt request is present
Processing:	

3.10.7.8. buzzer_start Function

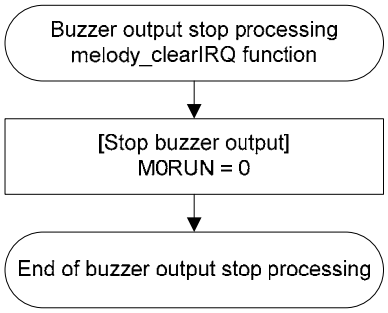
This function starts buzzer output.

Function name:	int buzzer_start(unsigned char type, unsigned char freq, unsigned char duty)
Arguments:	unsigned char type … Buzzer output sound selection BUZZER_TYPE0(=0): Intermittent sound 1 BUZZER_TYPE1(=1): Intermittent sound 2 BUZZER_TYPE2(=2): Single sound BUZZER_TYPE3(=3): Continuous sound unsigned char freq … Buzzer output frequency selection BUZZER_FREQ_4096HZ(=0) : 4.096kHz BUZZER_FREQ_2048HZ(=1) : 2.048kHz BUZZER_FREQ_1365HZ(=2) : 1.365kHz BUZZER_FREQ_1024HZ(=3) : 1.024kHz BUZZER_FREQ_819HZ(=4) : 819Hz BUZZER_FREQ_683HZ(=5) : 683Hz BUZZER_FREQ_585HZ(=6) : 585Hz BUZZER_FREQ_512HZ(=7) : 512Hz unsigned char duty … Buzzer output duty BUZZER_DUTY_1_16(=1) : 1/16DUTY BUZZER_DUTY_2_16(=2) : 2/16DUTY BUZZER_DUTY_3_16(=3) : 3/16DUTY BUZZER_DUTY_4_16(=4) : 4/16DUTY BUZZER_DUTY_5_16(=5) : 5/16DUTY BUZZER_DUTY_6_16(=6) : 6/16DUTY BUZZER_DUTY_7_16(=7) : 7/16DUTY BUZZER_DUTY_8_16(=8) : 8/16DUTY BUZZER_DUTY_9_16(=9) : 9/16DUTY BUZZER_DUTY_10_16(=10) : 10/16DUTY BUZZER_DUTY_11_16(=11) : 11/16DUTY BUZZER_DUTY_12_16(=12) : 12/16DUTY BUZZER_DUTY_13_16(=13) : 13/16DUTY BUZZER_DUTY_14_16(=14) : 14/16DUTY BUZZER_DUTY_15_16(=15) : 15/16DUTY BUZZER_DUTY_16_16(=16) : 16/16DUTY
Return values:	int BUZZER_R_OK(=0): Processing succeeded BUZZER_R_ERR_TYPE(=-1): The buzzer type selected is outside the range. BUZZER_R_ERR_FREQUENCY(=-2): The buzzer frequency selected is outside the range. BUZZER_R_ERR_DUTY(=-3): The buzzer output duty selected is outside the range.
Processing:	See next page.


Buzzer output start processing
buzzer_start function
Is the buzzer type outside the range?
(type £ BUZZER_TYPE3)
No
[Buzzer type is outside the range]
Return value: BUZZER_R_ERR_TYPE
Yes
Is the buzzer output frequency
outside the range?
(freq £ BUZZER_FREQ_512HZ)
No
[Buzzer output frequency is outside the range]
Return value: BUZZER_R_ERR_FREQUENCY
Yes
Is the buzzer output duty
outside the range?
(freq £ BUZZER_DUTY_15_16)
No
[Buzzer output duty is outside the range]
Return value: BUZZER_R_ERR_DUTY
Yes
[Stop buzzer output]
M0RUN = 0
[Select buzzer mode]
BZMD = 1
[Set buzzer type]
MD0TMP = type
[Set buzzer output frequency]
MD0TON = freq
[Set buzzer output duty]
MD0LEN = duty
[Stop buzzer output]
M0RUN = 0
[Normal end]
Return value: BUZZER_R_OK

3.10.7.9. buzzer_stop Function

This function stops buzzer output.

Function name:	void buzzer_stop(void)
Arguments:	None
Return values:	None
Processing:	

3.11. Real Time Clock Control Module

3.11.1. Overview of Functions

The real time clock control module controls the real time clock of the MCU.
Controllable items are: Setting/obtaining/starting/stopping the date (year, month, day, day of the week) counting function and clock time (hour, minute, second) counting function; setting an interval between periodic interrupts; setting/obtaining alarm 0 (comparing day of the week, hour, and minute) and alarm 1 (comparing month, day, hour, and minute).

* This module is available in the ML610Q431/Q432. The ML610Q411/Q412/Q415 is not provided with the real time clock function.

3.11.2. List of APIs

The following table lists the real time clock control module APIs.

Table 3-39 Real Time Clock Control Module APIs

Function name	Description
rtc_setTime function	Sets date (year, month, day, day of the week) and clock time (hour, minute, second).
rtc_getTime function	Obtains date (year, month, day, day of the week) and clock time (hour, minute, second).
rtc_start function	Starts RTC operation.
rtc_stop function	Stops RTC operation.
rtc_setRegularInt function	Selects the interval between periodic interrupts.
rtc_setAlarm0 function	Sets alarm 0 (day of the week, hour, minute).
rtc_setAlarm1 function	Sets alarm 1 (month, day, hour, minute).
rtc_getAlarm0 function	Obtains the setting of alarm 0 (day of the week, hour, minute).
rtc_getAlarm1 function	Obtains the setting of alarm 1 (month, day, hour, minute).

3.11.3. List of Constants

The following tables list the constants used in the real time clock control module.

Table 3-40 List of Constants for Arguments

Constant name	Defined value	Description
RTC_RIN_DISABLE	0	Periodic interrupt: Disabled
RTC_RIN_0_5_SEC	1	Periodic interrupt: Enables 0.5-second interrupt
RTC_RIN_1_0_SEC	2	Periodic interrupt: Enables 1-second interrupt
RTC_RIN_1_0_MIN	3	Periodic interrupt: Enables 1-minute interrupt
RTC_ALEN_DIS	0	Disables the alarm function.
RTC_ALEN_ENA	1	Enables the alarm function.

Table 3-41 List of Constants for Return Values

Constant name	Defined value	Description
RTC_R_OK	0	Processing succeeded
RTC_R_ERR_SEC	-1	The second setting is outside the range
RTC_R_ERR_MIN	-2	The minute setting is outside the range
RTC_R_ERR_HOUR	-3	The hour setting is outside the range
RTC_R_ERR_WEEK	-4	The day of the week setting is outside the range
RTC_R_ERR_DAY	-5	The day setting is outside the range
RTC_R_ERR_MON	-6	The month setting is outside the range
RTC_R_ERR_YEAR	-7	The year setting is outside the range
RTC_R_ERR_ALEN	-8	The alarm enable/disable setting is outside the range
RTC_R_ERR_RIN	-9	The periodic interrupt setting is outside the range
RTC_R_ERR_GETTIME	-10	Obtaining of date and clock time failed

3.11.4. Structure

This section describes the structures used in the real time clock control module. Each parameter value must be specified not by binary-coded decimal (BCD) numbers but by decimal numbers.

■ Date and watch setting parameters

```
typedef struct {
    unsigned char    sec;     // Second data (0 to 59)
    unsigned char    min;     // Minute data (0 to 59)
    unsigned char    hour;    // Hour data (0 to 23)
    unsigned char    week;    // Day of the week data (0 to 7)
    unsigned char    day;     // Day data (0 to 31)
    unsigned char    mon;     // Month data (0 to 12)
    unsigned char    year;    // Year data (0 to 99)
} tRtcTime;
```

■ Alarm 0 setting parameters

```
typedef struct {
    unsigned char    min;     // Minute data (0 to 59)
    unsigned char    hour;    // Hour data (0 to 23)
    unsigned char    week;    // Day of the week data (0 to 7)   * If 0 is specified, no day-of-the-week data is used for comparison.
} tRtcAlarm0;
```

■ Alarm 1 setting parameters

```
typedef struct {
    unsigned char    min;     // Minute data (0 to 59)
    unsigned char    hour;    // Hour data (0 to 23)
    unsigned char    day;     // Day data (0 to 31)    * If 0 is specified, no day data is used for comparison.
    unsigned char    mon;     // Month data (0 to 12)    * If 0 is specified, no month data is used for comparison.
} tRtcAlarm1;
```

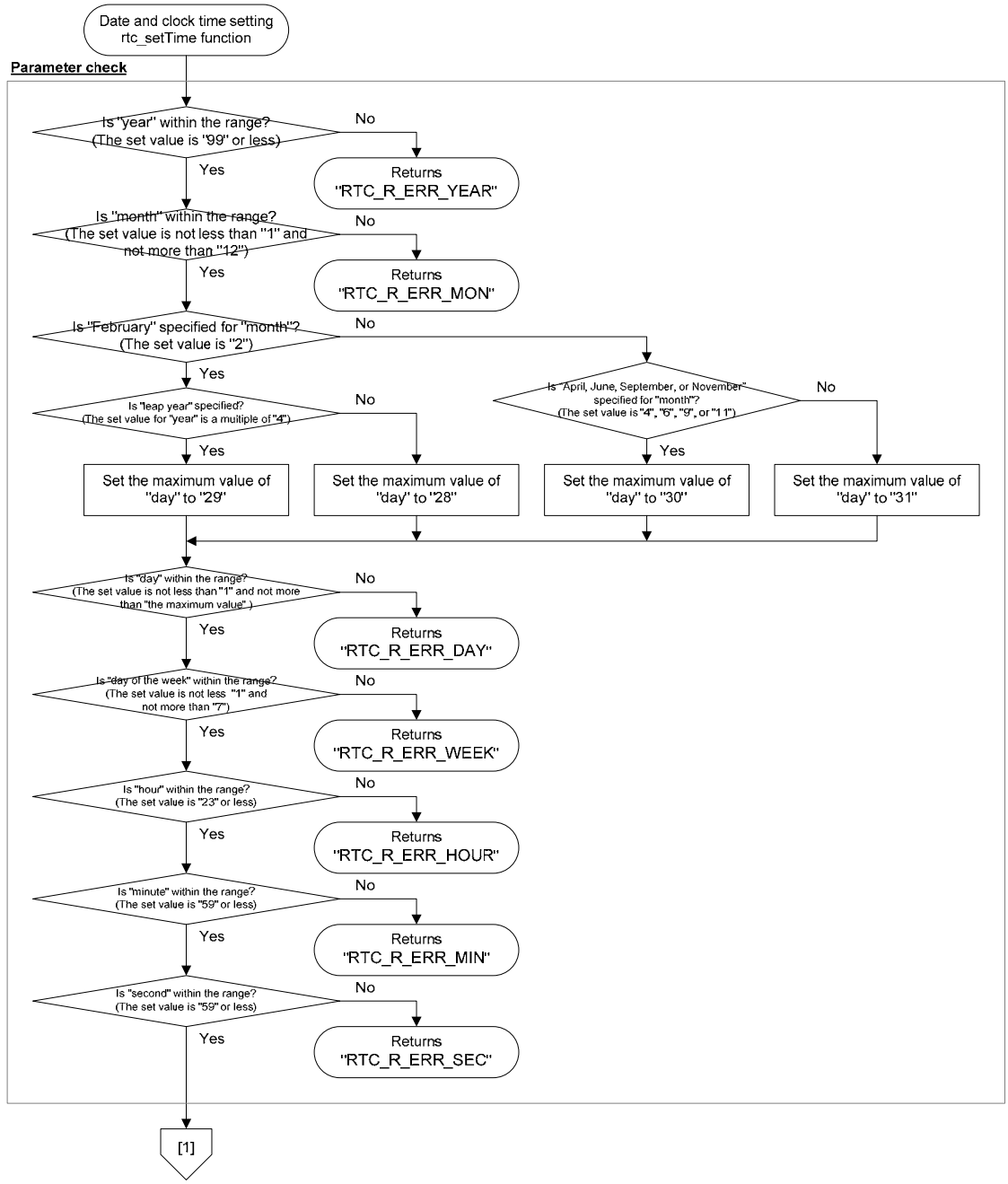
3.11.5. Details of APIs

This section describes details of the time clock control module APIs.

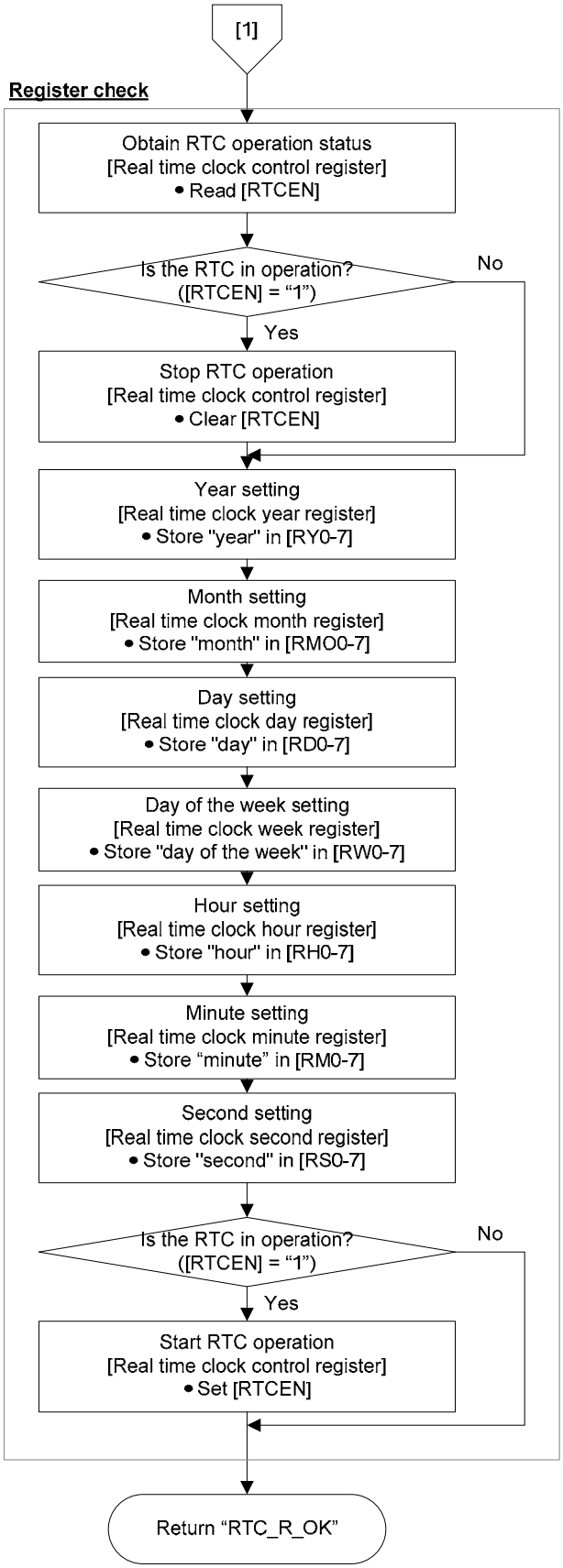
3.11.5.1. rtc_setTime Function

This function sets date (year, month, day, day of the week) and clock time (hour, minute, second). If this function is executed during operation of the RTC, the RTC is once put into a stopped state within this function. Then, data and clock time are set and after that, the RTC is reactivated.

Function name:	int rtc_setTime(tRtcTime *prm)
Arguments:	tRtcTime *prm … Pointer to the area that contains the date (year, month, day, day of the week) and clock time (hour, minute, second) to be set
Return values:	int Setting succeeded: RTC_R_OK(=0) The second setting is outside the range: RTC_R_ERR_SEC(=-1) The minute setting is outside the range: RTC_R_ERR_MIN(=-2) The hour setting is outside the range: RTC_R_ERR_HOUR(=-3) The day of the week setting is outside the range: RTC_R_ERR_WEEK(=-4) The day setting is outside the range: RTC_R_ERR_DAY(=-5) The month setting is outside the range: RTC_R_ERR_MON(=-6) The year setting is outside the range: RTC_R_ERR_YEAR(=-7)
Processing:	See next page.

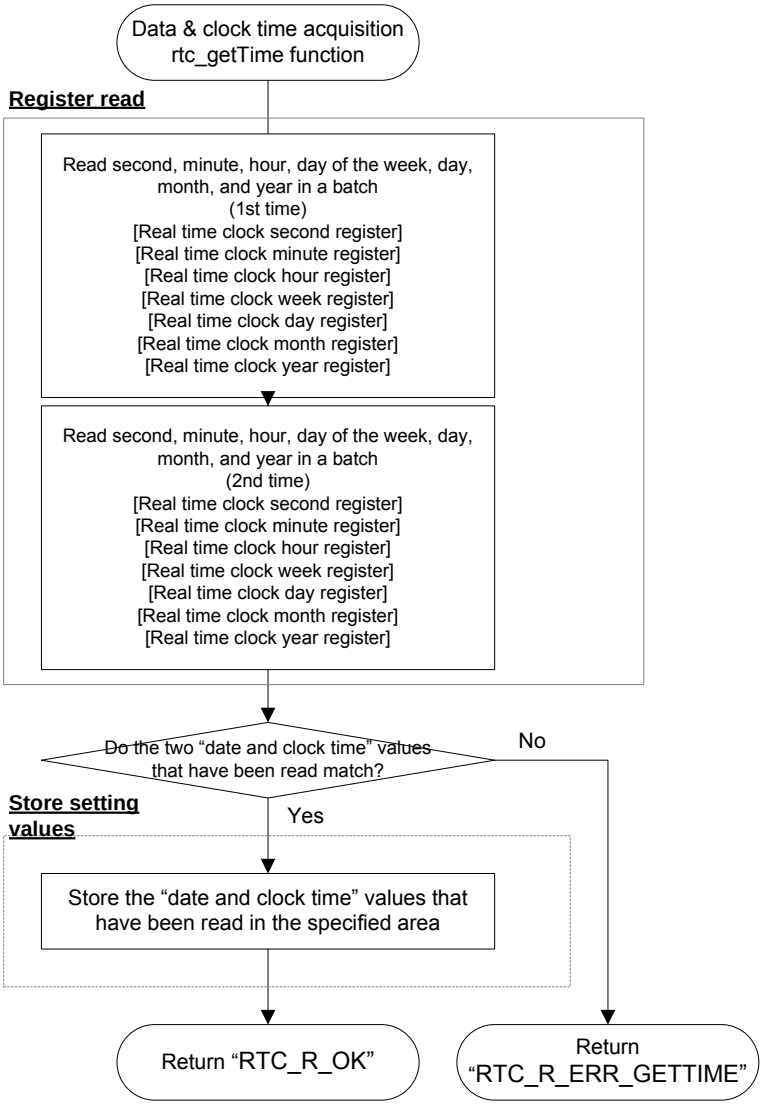


Continued on next page


[1]
Register check
Obtain RTC operation status
[Real time clock control register]
• Read [RTCEN]
Is the RTC in operation?
([RTCEN] = "1")
No
Yes
Stop RTC operation
[Real time clock control register]
• Clear [RTCEN]
Year setting
[Real time clock year register]
• Store "year" in [RY0-7]
Month setting
[Real time clock month register]
• Store "month" in [RMO0-7]
Day setting
[Real time clock day register]
• Store "day" in [RD0-7]
Day of the week setting
[Real time clock week register]
• Store "day of the week" in [RW0-7]
Hour setting
[Real time clock hour register]
• Store "hour" in [RH0-7]
Minute setting
[Real time clock minute register]
• Store "minute" in [RM0-7]
Second setting
[Real time clock second register]
• Store "second" in [RS0-7]
Is the RTC in operation?
([RTCEN] = "1")
No
Yes
Start RTC operation
[Real time clock control register]
• Set [RTCEN]
Return "RTC_R_OK"

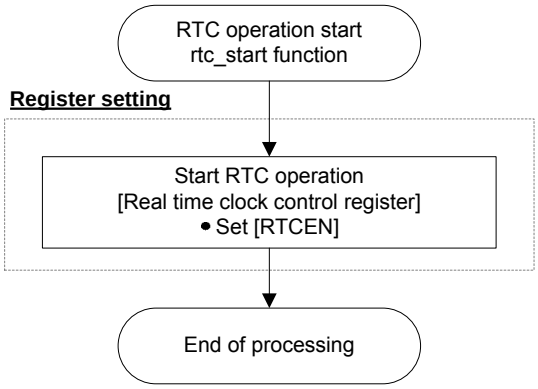
3.11.5.2. rtc_getTime Function

This function acquires date (year, month, day, day of the week) and clock time (hour, minute, second). In order to prevent reading of undefined data during counting, read every register twice and check that the read two values match.

Function name:	int rtc_getTime(tRtcTime *prm)
Arguments:	tRtcTime *prm … Pointer to the area that stores date (year, month, day, day of the week) and clock time (hour, minute, second)
Return values:	int Acquisition succeeded (values match): RTC_R_OK(=0) Acquisition failed (values do not match): RTC_R_ERR_GETTIME(=-10)
Processing:	

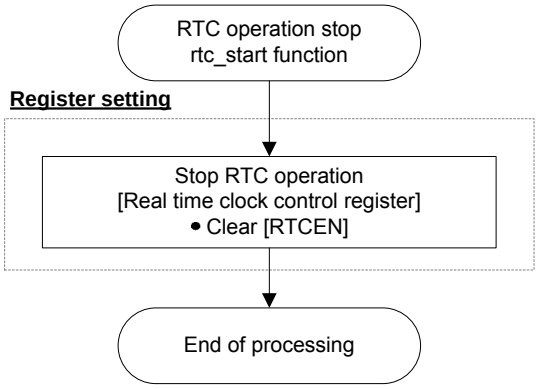
3.11.5.3. rtc_start Function

This function starts RTC operation.

Function name:	void rtc_start(void)
Arguments:	None
Return values:	None
Processing:	

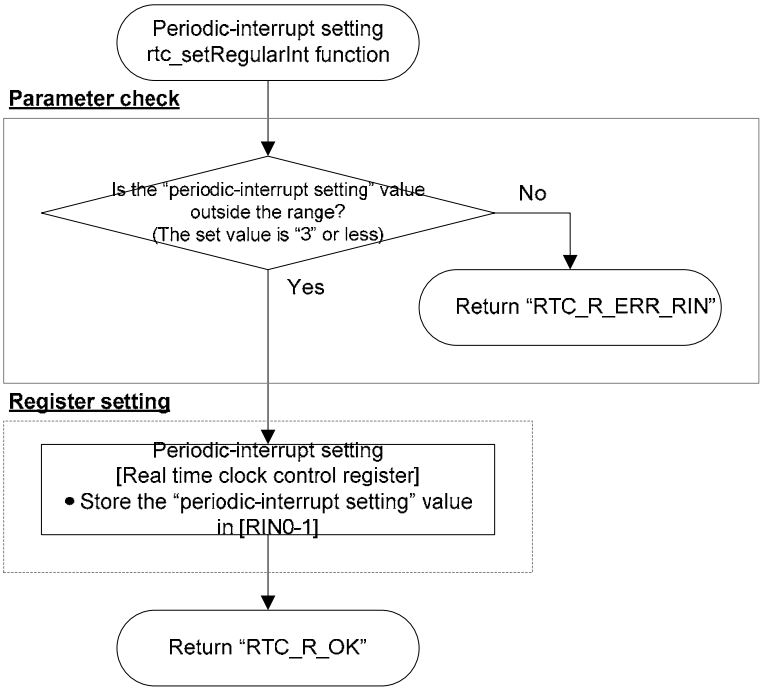
3.11.5.4. rtc_stop Function

This function stops RTC operation.

Function name:	void rtc_stop(void)
Arguments:	None
Return values:	None
Processing:	

3.11.5.5. rtc_setRegularInt Function

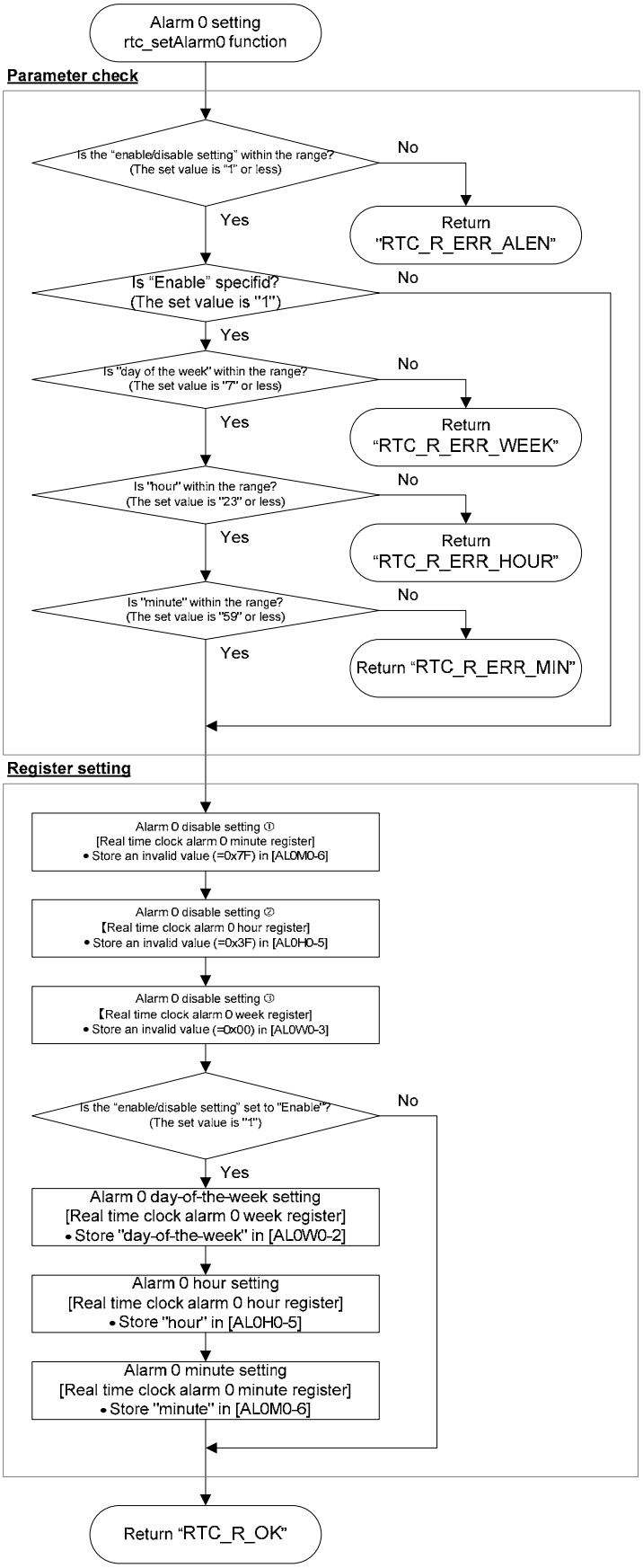
This function sets settings for periodic interrupt.

Function name:	int rtc_setRegularInt(unsigned char rin)
Arguments:	unsigned char rin … Periodic-interrupt setting Disable periodic interrupts: RTC_RIN_DISABLE(=0) Enable 0.5-second interrupt: RTC_RIN_0_5_SEC(=1) Enable 1-second interrupts: RTC_RIN_1_0_SEC(=2) Enable 1-minute interrupts: RTC_RIN_1_0_MIN(=3)
Return values:	int Setting succeeded: RTC_R_OK(=0) Periodic interrupt setting is outside the range: RTC_R_ERR_RIN(=-9)
Processing:	

3.11.5.6. rtc_setAlarm0 Function

This function sets alarm 0.

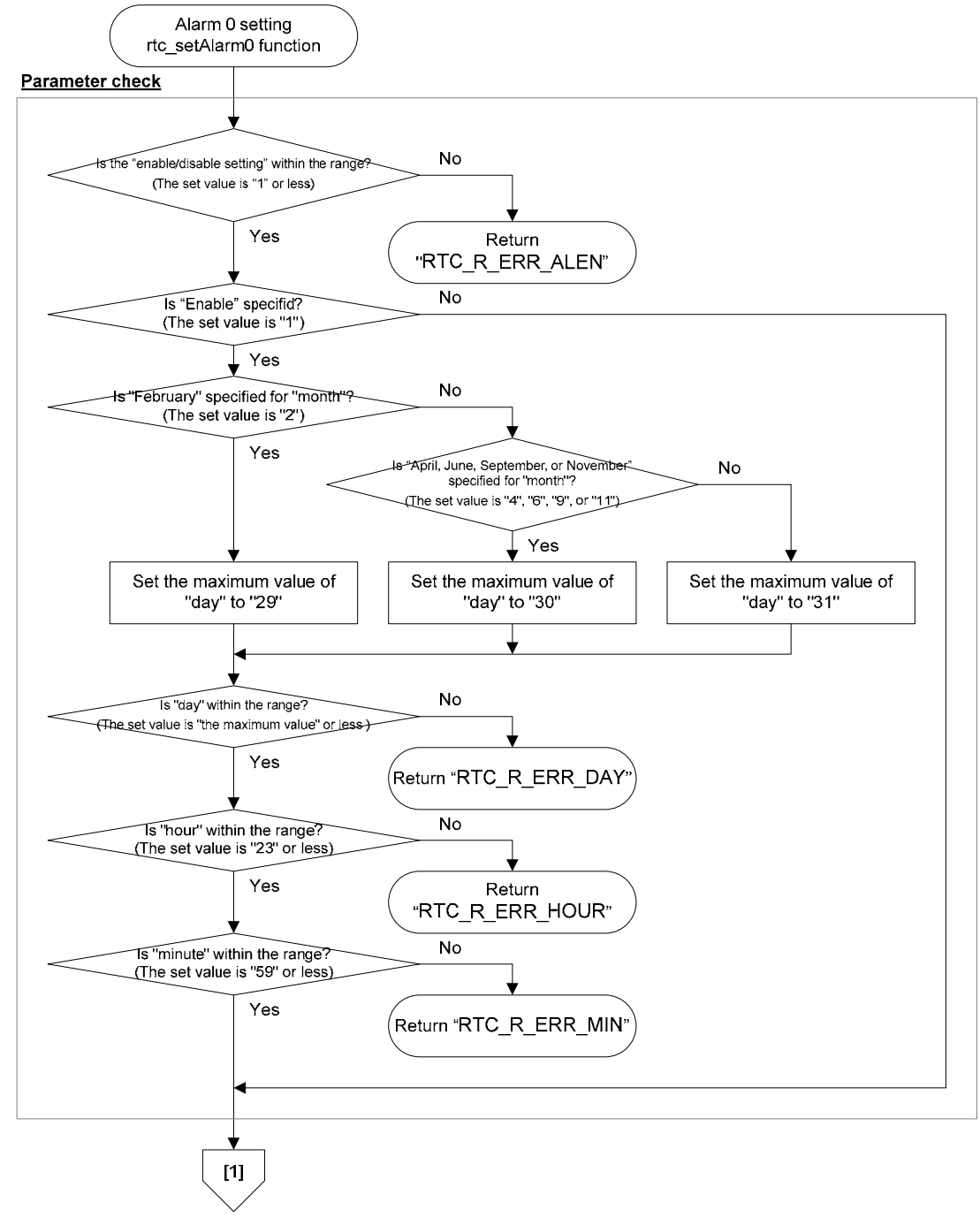
Function name:	int rtc_setAlarm0(unsigned char alen, tRtcAlarm0* prm)
Arguments:	unsigned char alen … Alarm 0 enable/disable setting Disable alarm 0: RTC_ALEN_DIS(=0) Enable alarm 0: RTC_ALEN_ENA(=1) tRtcAlarm0* prm …Pointer to the area that contains the alarm 0 setting (day of the week, hour, minute) - If 0 is specified for day of the week, no day-of-the-week data is used as comparison data for alarm 0.
Return values:	int Setting succeeded: RTC_R_OK(=0) The minute setting is outside the range: RTC_R_ERR_MIN(=-2) The hour setting is outside the range: RTC_R_ERR_HOUR(=-3) The day of the week setting is outside the range: RTC_R_ERR_WEEK(=-4) The enable/disable setting is outside the range: RTC_R_ERR_ALEN(=-9)
Processing:	See next page.


Alarm 0 setting
rtc_setAlarm0 function
Parameter check
Is the "enable/disable setting" within the range?
(The set value is "1" or less)
No
Return
"RTC_R_ERR_ALEN"
Yes
Is "Enable" specified?
(The set value is "1")
No
Yes
Is "day of the week" within the range?
(The set value is "7" or less)
No
Return
"RTC_R_ERR_WEEK"
Yes
Is "hour" within the range?
(The set value is "23" or less)
No
Return
"RTC_R_ERR_HOUR"
Yes
Is "minute" within the range?
(The set value is "59" or less)
No
Return "RTC_R_ERR_MIN"
Yes
Register setting
Alarm 0 disable setting ①
[Real time clock alarm 0 minute register]
• Store an invalid value (=0x7F) in [AL0M0-6]
Alarm 0 disable setting ②
【Real time clock alarm 0 hour register]
• Store an invalid value (=0x3F) in [AL0H0-5]
Alarm 0 disable setting ③
【Real time clock alarm 0 week register]
• Store an invalid value (=0x00) in [AL0W0-3]
Is the "enable/disable setting" set to "Enable"?
(The set value is "1")
No
Yes
Alarm 0 day-of-the-week setting
[Real time clock alarm 0 week register]
• Store "day-of-the-week" in [AL0W0-2]
Alarm 0 hour setting
[Real time clock alarm 0 hour register]
• Store "hour" in [AL0H0-5]
Alarm 0 minute setting
[Real time clock alarm 0 minute register]
• Store "minute" in [AL0M0-6]
Return "RTC_R_OK"

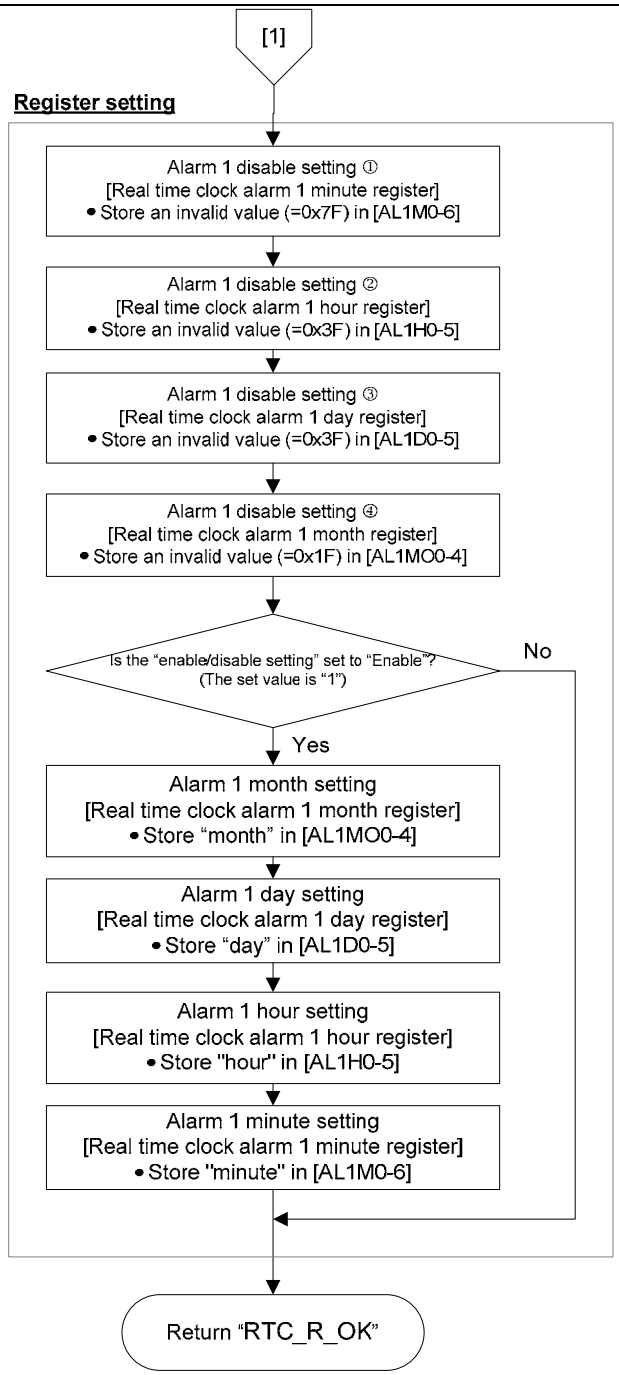
3.11.5.7. rtc_setAlarm1 Function

This function sets alarm 1.

Function name:	int rtc_setAlarm1(unsigned char alen, tRtcAlarm1* prm)
Arguments:	unsigned char alen … Alarm 1 enable/disable setting Disable alarm 1: RTC_ALEN_DIS(=0) Enable alarm 1: RTC_ALEN_ENA(=1) tRtcAlarm1* prm …Pointer to the area that contains the alarm 1 setting (month, day, hour, minute) - If 0 is specified for month, no month data is used as comparison data for alarm 1. - If 0 is specified for day, no day data is used as comparison data for alarm 1.
Return values:	int Setting succeeded: RTC_R_OK(=0) The minute setting is outside the range: RTC_R_ERR_MIN(=-2) The hour setting is outside the range: RTC_R_ERR_HOUR(=-3) The day setting is outside the range: RTC_R_ERR_DAY(=-5) The month setting is outside the range: RTC_R_ERR_MON(=-6) The enable/disable setting is outside the range: RTC_R_ERR_ALEN(=-9)
Processing:	See next page.

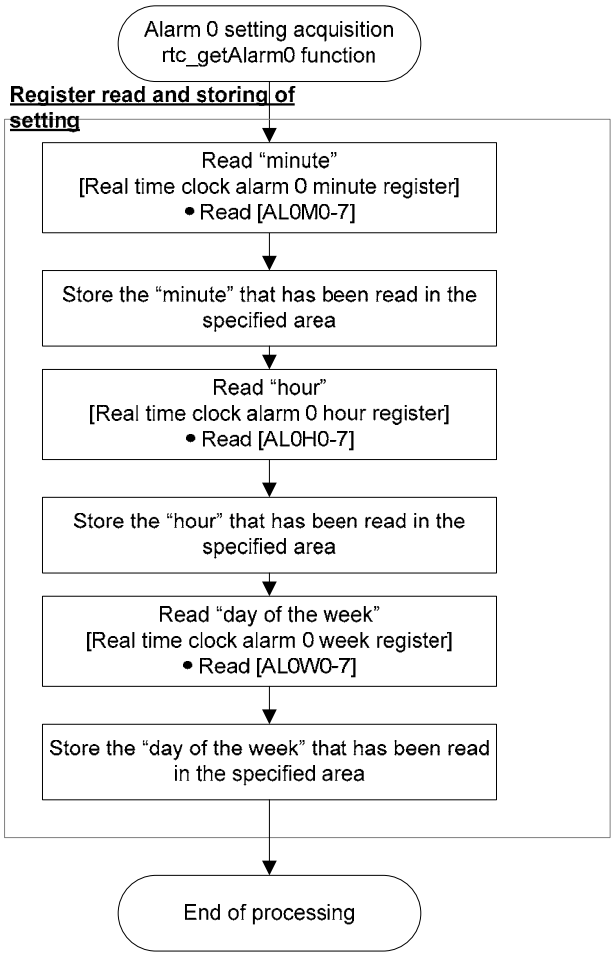


Continued on next page.


[1]
Register setting
Alarm 1 disable setting ①
[Real time clock alarm 1 minute register]
• Store an invalid value (=0x7F) in [AL1M0-6]
Alarm 1 disable setting ②
[Real time clock alarm 1 hour register]
• Store an invalid value (=0x3F) in [AL1H0-5]
Alarm 1 disable setting ③
[Real time clock alarm 1 day register]
• Store an invalid value (=0x3F) in [AL1D0-5]
Alarm 1 disable setting ④
[Real time clock alarm 1 month register]
• Store an invalid value (=0x1F) in [AL1MO0-4]
Is the "enable/disable setting" set to "Enable"?
(The set value is "1")
No
Yes
Alarm 1 month setting
[Real time clock alarm 1 month register]
• Store "month" in [AL1MO0-4]
Alarm 1 day setting
[Real time clock alarm 1 day register]
• Store "day" in [AL1D0-5]
Alarm 1 hour setting
[Real time clock alarm 1 hour register]
• Store "hour" in [AL1H0-5]
Alarm 1 minute setting
[Real time clock alarm 1 minute register]
• Store "minute" in [AL1M0-6]
Return "RTC_R_OK"

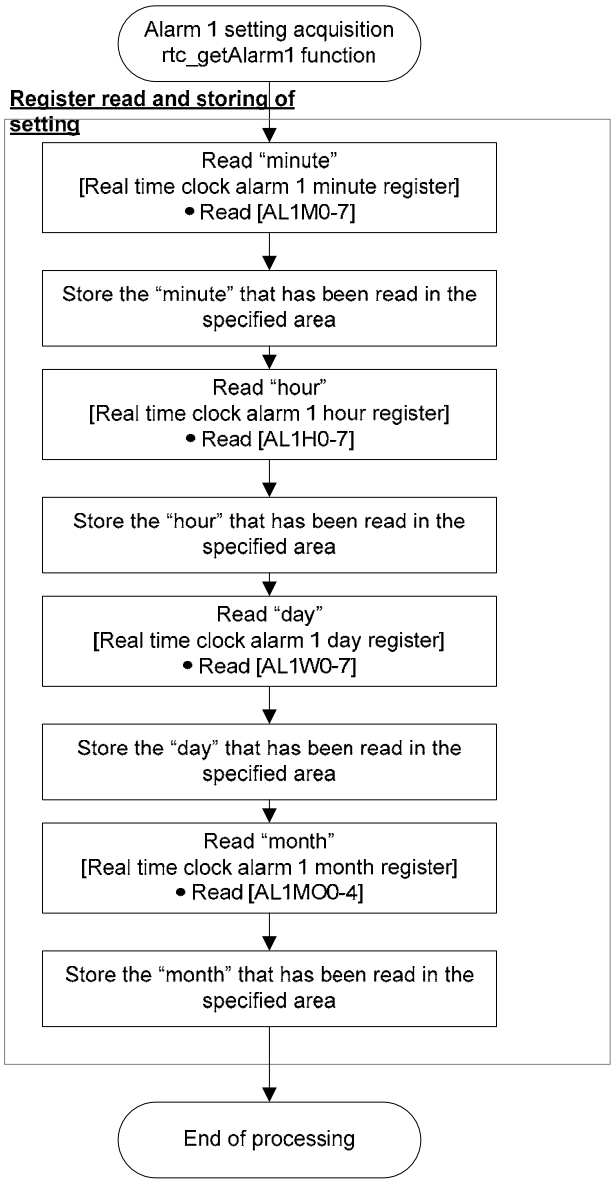
3.11.5.8. rtc_getAlarm0 Function

This function acquires the alarm 0 setting.

Function name:	void rtc_getAlarm0(tRtcAlarm0 *prm)
Arguments:	tRtcAlarm0 *prm … Pointer to the area that stores the alarm 0 setting (day of the week, hour, minute)
Return values:	None
Processing:	

3.11.5.9. rtc_getAlarm1 Function

This function acquires the alarm 1 setting.

Function name:	void rtc_getAlarm1(tRtcAlarm1*prm)
Arguments:	tRtcAlarm1 *prm … Pointer to the area that stores the alarm 1 setting (month, day, hour, minute)
Return values:	None
Processing:	

3.12. Timer Module

3.12.1. Overview of Functions

The timer module controls the timer of the MCU.
Control of the timer is achieved by APIs that perform initialization (operating clock setting, 8/16-bit mode setting, overflow interval specification), timer start/stop, and timer overflow flag acquisition and clear.

3.12.2. List of APIs

The following table lists the timer module APIs.

Table 3-42 Timer Module APIs

Function name	Description
tm_init function	Executes operating clock setting, 8-bit or 16-bit mode setting, and overflow interval setting.
tm_start function	Starts timer operation. Operation setting is performed with tm_init function.
tm_stop function	Stops timer operation.
tm_checkOvf function	Checks the timer overflow flag status.
tm_clearOvf function	Executes timer overflow flag clear operation.
tm_checkFmFunc function	Checks whether the 16-bit frequency measuring mode is enabled or disabled.

3.12.3. List of Constants

The following tables list the constants used in the timer module.

Table 3-43 Constants for Arguments

Constant name	Defined value	Description
TM_M16_8BIT	0	Specifies 8-bit mode as the operating mode.
TM_M16_16BIT	1	Specifies 16-bit mode as the operating mode.
TM_CS_LSCLK	0	Specifies LSCLK as the operating clock.
TM_CS_HTBCLK	1	Specifies HTBCLK as the operating clock.
TM_CS_EXTCLK	3	Specifies an external clock as the operating clock.

Table 3-44 Constants for Return Values

Constant name	Defined value	Description
TM_R_OK	0	Processing succeeded.
TM_R_ERR_CHNO	-1	The channel No. is outside the range.
TM_R_ERR_M16	-2	The operating mode setting is outside the range.
TM_R_ERR_CS	-3	The operating clock setting is outside the range.
TM_R_ERR_CNT	-4	The overflow interval is outside the range.
TM_R_OVF	1	Overflow has occurred.
TM_R_NOT_OVF	0	Overflow has not occurred yet.
TM_R_FM_ENA	1	16-bit frequency measuring mode is enabled.
TM_R_FM_DIS	0	16-bit frequency measuring mode is disabled.

3.12.4. API Details

This section describes details of the timer module.

3.12.4.1. tm_init Function

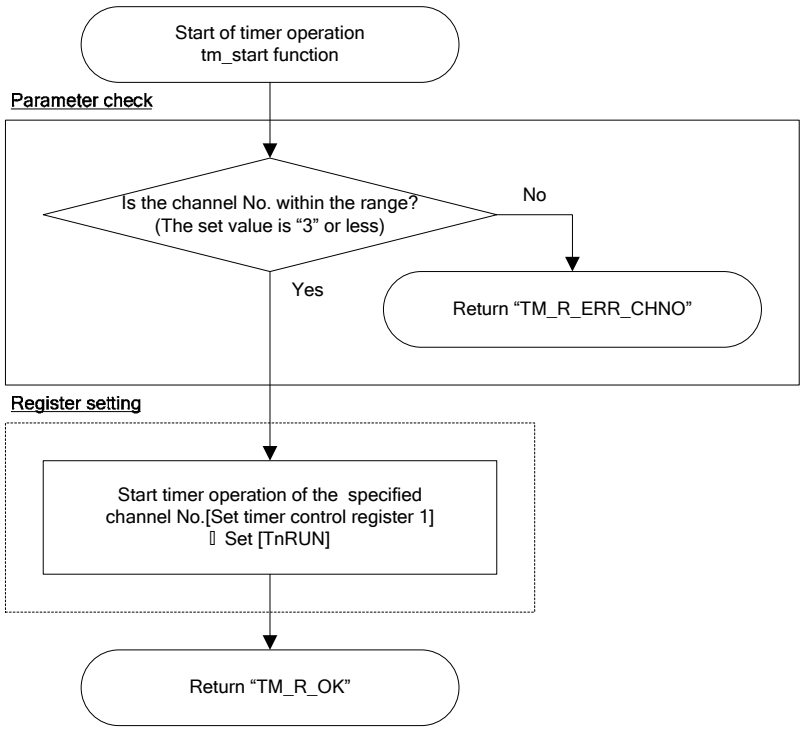
This function sets timer's operating clock, selects 8- or 16-bit mode, and specifies the overflow interval.

Function name:	int tm_init(unsigned char chNo, unsigned char m16, unsigned char cs, unsigned short cnt,)
Arguments:	unsigned char chNo … Timer channel No. (0 to 3) unsigned char m16 … Operating mode 8-bit mode: TM_M16_8BIT(=0) 16-bit mode: TM_M16_16BIT(=1) 16-bit frequency measuring mode: TM_M16_FM(=3) unsigned char cs … Operating clock LSCLK: TM_CS_LSCLK(=0) HTBCLK: TM_CS_HTBCLK(=1) External clock: TM_CS_EXTCLK(=3) unsigned short cnt … Overflow interval (specified with count)
Return values:	int Initialization succeeded: TM_R_OK(=0) The channel No. is outside the range: TM_R_ERR_CHNO(= -1) The operating mode setting is outside the range: TM_R_ERR_M16(= -2) The operating clock setting is outside the range: TM_R_ERR_CS(= -3) The overflow interval is outside the range: TM_R_ERR_CNT(= -4)
Processing:	See next page.


Timer module initialization
tm_init function
Parameter check
Is the operating clock setting within the range?
(The set value is "0", "1", "3")
No
Yes
Return "TM_R_ERR_CS"
Is the operating mode "8-bit mode"?
No
Yes
Is the operating mode "16-bit mode"?
No
Yes
Is the operating mode "16-bit frequency measuring mode"?
No
Yes
Obtain information as to whether the frequency measurement function is enabled or disabled
[Timer 2 control register 0]
[Set timer control register 1]
• See [T2FMA4-7]
Is the frequency measurement function enabled?
([T2FMA4-7] value is 0xA)
No
Yes
Return "TM_R_ERR_M16"
Is the channel No. within the range?
(The set value is "3" or less)
No
Yes
Return "TM_R_ERR_CHNO"
Is the channel No. within the range?
(The set value is "0" or "2")
No
Yes
Return "TM_R_ERR_CHNO"
Is the channel No. within the range?
(The set value is "2")
No
Yes
Return "TM_R_ERR_CHNO"
Is the "overflow interval" within the range?
(between 00h and FFh)
No
Yes
Return "TM_R_ERR_CNT"
Is the operating clock setting within the range?
(The set value is "1")
No
Yes
Return "TM_R_ERR_CS"
Register setting
Is the operating mode "8-bit mode"?
No
Yes
Stop the timer operation on the specified channel No.
[Set the timer control register 1]
• Clear [TnRUN]
Clear the timer counter on the specified channel No.
[Set the timer counter register]
• Set "0" in [TnC0-7]
Set the overflow count value of the specified channel No.
[Set the timer data register]
• Set "overflow interval" in [TnD0-7]
Set the operating mode and operating clock on the specified channel No.
[Set timer control register 0]
• Set "0" in [TnM16]
• Set "operating clock" in [TnCS0-1]
Stop the timer operation on the [specified channel No.] and [specified channel No.+1]
[Set the timer control register 1]
• Clear [TnRUN] and [T(n+1)RUN]
Clear the timer counter registers on the [specified channel No.] and [specified channel No.+1]
[Set the timer counter register]
• Set "0" in [TnC0-7] and [T(n+1)C0-7]
Store the calculated setting values in the timer data registers on the [specified channel No.] and [specified channel No.+1]
[Set the timer data register]
• Set "overflow interval" in [TnD0-7] and [T(n+1)D0-7]
Set the operating mode and operating clock on the specified channel
[Set timer control register 0]
• Set "operating mode" in [TnM16]
• Set "operating clock" in[TnCS0-1]
Return "TM_R_OK"

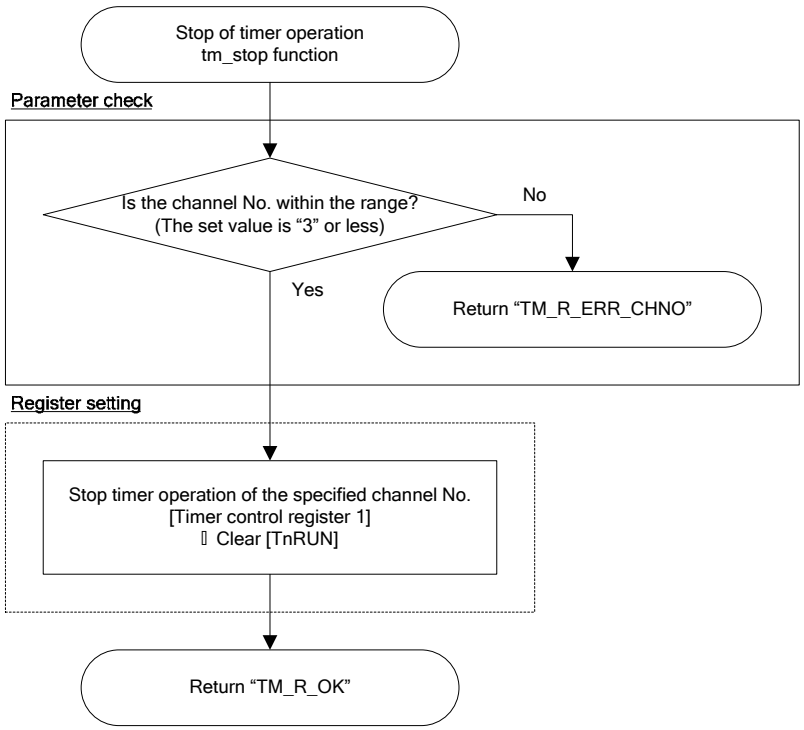
3.12.4.2. tm_start Function

This function starts timer operation. Set the overflow interval using the tm_init function.

Function name:	int tm_start(unsigned char chNo)
Arguments:	unsigned char chNo … Timer channel No. (0 to 3)
Return values:	int Start processing succeeded: TM_R_OK(=0) The channel No. is outside the range: TM_R_ERR_CHNO(=-1)
Processing:	

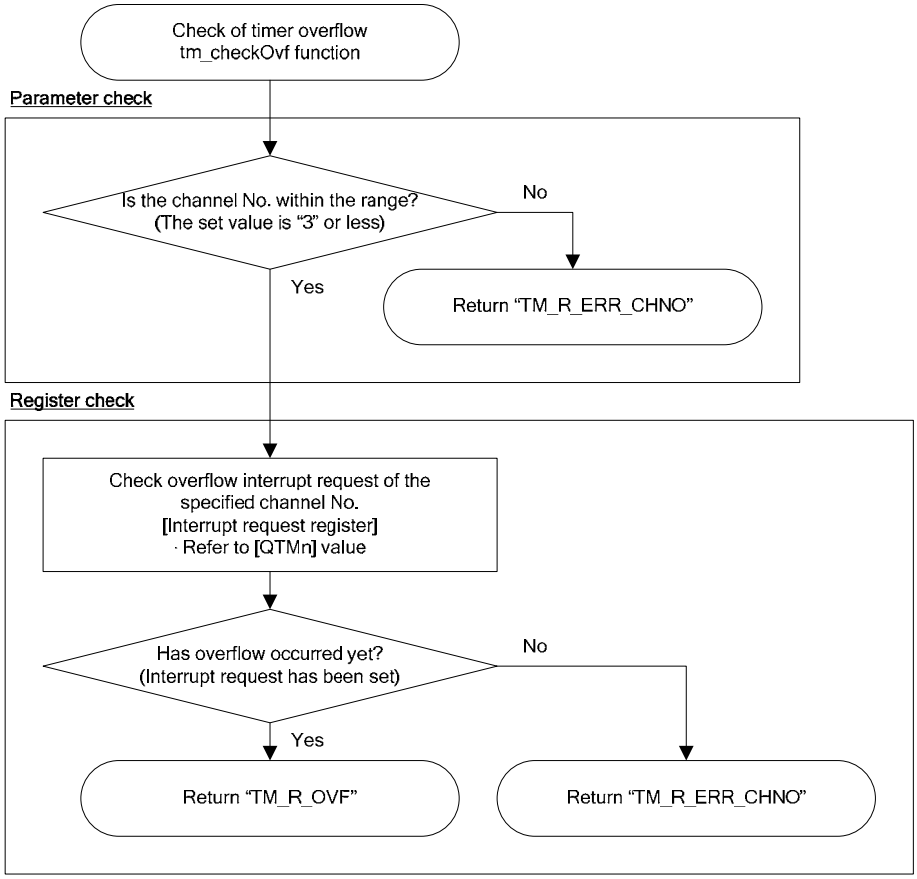
3.12.4.3. tm_stop Function

This function stops timer operation.

Function name:	int tm_stop(unsigned char chNo)
Arguments:	unsigned char chNo … Timer channel No. (0 to 3)
Return values:	int Stop processing succeeded: TM_R_OK(=0) The channel No. is outside the range: TM_R_ERR_CHNO(=-1)
Processing:	

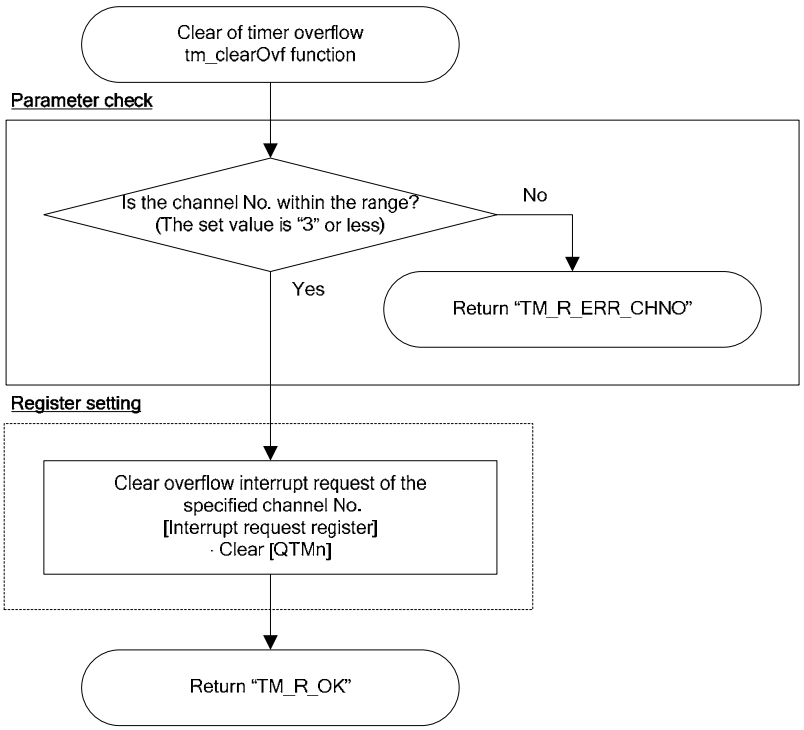
3.12.4.4. tm_checkOvf Function

This function checks the status of the timer overflow flag (QTMn of the interrupt request register). However, since the MCU clears the timer overflow flag during the use of interrupts, this function always returns "No overflow has occurred yet: TM_R_NOT_OVF(=0)".

Function name:	int tm_checkOvf(unsigned char chNo)
Arguments:	unsigned char chNo … Timer channel No. (0 to 3)
Return values:	int Overflow has occurred: TM_R_OVF(=1) No overflow has occurred yet: TM_R_NOT_OVF(=0) The channel No. is outside the range: TM_R_ERR_CHNO(=-1)
Processing:	

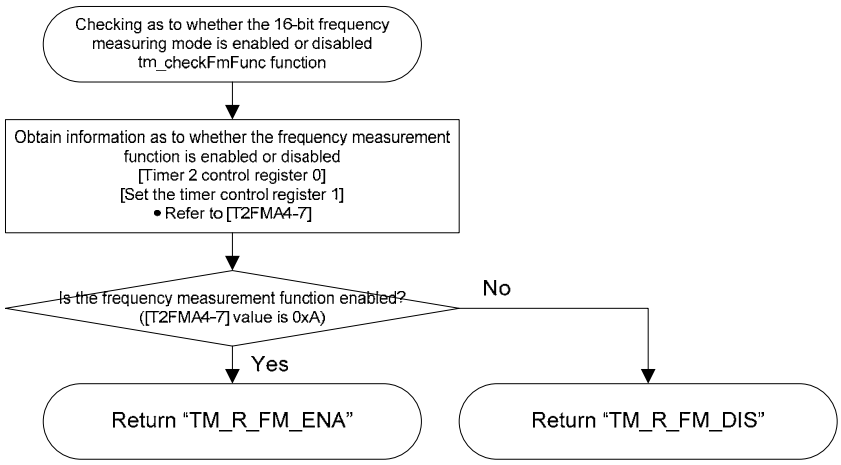
3.12.4.5. tm_clearOvf Function

This function performs timer overflow flag clear processing (QTMn of the interrupt request register).

Function name:	int tm_clearOvf(unsigned char chNo)
Arguments:	unsigned char chNo … Timer channel No.(0 to 3)
Return values:	int Clear processing succeeded: TM_R_OK(=0) The channel No. is outside the range: TM_R_ERR_CHNO(=-1)
Processing:	

3.12.4.6. tm_checkFmFunction

This function checks whether the 16-bit frequency measuring mode is enabled or disabled.

Function name:	int tm_checkFmFunc(void)
Arguments:	None
Return values:	int 16-bit frequency measuring mode is enabled: TM_R_FM_ENA(=1) 16-bit frequency measuring mode is disabled: TM_R_FM_DIS(=-0)
Processing:	

3.13. Clock Control Module

3.13.1. Overview of Functions

The clock control module controls clocks of the MCU.
Clock control is achieved by functions that perform operations such as system clock selection and acquisition, high-speed clock divide ratio selection, selection of the operating mode of high-speed clock, start/stop of the oscillation of the high-speed clock oscillation circuit, and start/stop of the low-speed 2x clock.

3.13.2. List of Functions

The clock control module APIs and system definition functions are listed below.

Table 3-45 Clock Control Module APIs

Function name	Description
clk_setSysclk function	Selects low-speed clock or high-speed clock as the system clock; selects the divide ratio of high-speed clock; selects the operating mode of the high-speed clock generation circuit; sets the clock frequency to be used.
clk_getSysclk function	Acquires the system clock settings (LSCLK/HSCLK selection, system clock frequency).
clk_setHsclk function	Selects the divide ratio of high-speed clock; selects the operating mode of the high-speed clock generation circuit; sets the clock frequency to be used.
clk_enaHsclk function	Starts oscillation of the high-speed clock oscillation circuit.
clk_disHsclk function	Stops oscillation of the high-speed clock oscillation circuit.
clk_getHsclk function	Acquires the high-speed clock frequency.
clk_enaLsclk2 function	Starts the operation of the low-speed 2x clock.
clk_disLsclk2 function	Stops the operation of the low-speed 2x clock.

3.13.3. List of System Definition Functions

The following table lists the system definition functions of the clock control module. When using a system definition function, it is necessary to create internal processing according to the system used.

Table 3-46 System Definition Functions of Clock Control Module

Function name	Description
clk_wait500us function	Waits for 500 μs.

3.13.4. List of Constants

The following tables list the constants used in the clock control module.

Table 3-47 Constants for Arguments

Constant name	Defined value	Description
CLK_SYSCLK_LSCLK	0	Selects low-speed clock as the system clock.
CLK_SYSCLK_HSCLK	1	Selects high-speed clock as the system clock.
CLK_SYSC_OSCLK	0	When high-speed clock selected: Selects OSCLK.
CLK_SYSC_OSCLK_DIV2	1	When high-speed clock selected: Selects 1/2OSCLK
CLK_SYSC_OSCLK_DIV4	2	When high-speed clock selected: Selects 1/4OSCLK
CLK_SYSC_OSCLK_DIV8	3	When high-speed clock selected: Selects 1/8OSCLK
CLK_OSCM_RC	0	When high-speed clock selected: Selects RC oscillation mode.
CLK_OSCM_CRYSTAL	1	When high-speed clock selected: Selects crystal/ceramic oscillation mode.
CLK_OSCM_PLL	2	When high-speed clock selected: Selects internal PLL oscillation mode.
CLK_OSCM_EXTCLK	3	When high-speed clock selected: Selects external clock input mode.

Table 3-48 Constants for Return Values

Constant name	Defined value	Description
CLK_R_OK	0	Processing succeeded.
CLK_R_ERR_SCLK	-1	The selected system clock is outside the range.
CLK_R_ERR_SYSC	-2	The selected divide ratio is outside the range.
CLK_R_ERR_OSCM	-3	The selected high-speed clock is outside the range.
CLK_R_ERR_ENOSC	-4	High-speed oscillation is operating.

3.13.5. List of Variables

The following table lists the variables used in the clock control module.

Table 3-49 List of Variables

Variable name	Initial value	Description
Static unsigned short _gsHsclk	4096	Variable for storing the frequency when the crystal/ceramic oscillation mode or external clock input mode is selected. Since the input clock in the above modes is externally connected, it is necessary to retain the specified value inside the module.

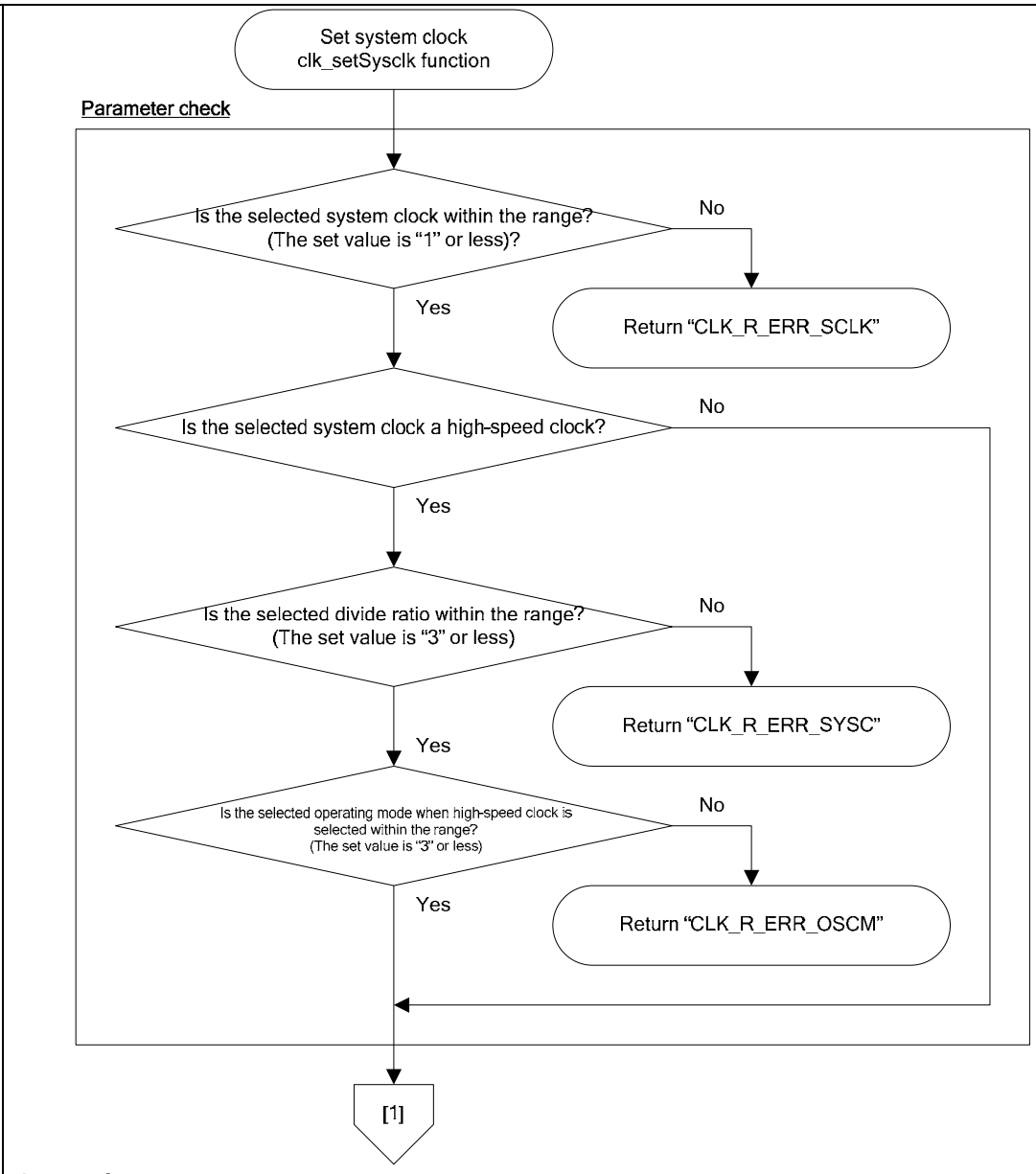
3.13.6. Details of APIs

This section describes details of the clock control module.

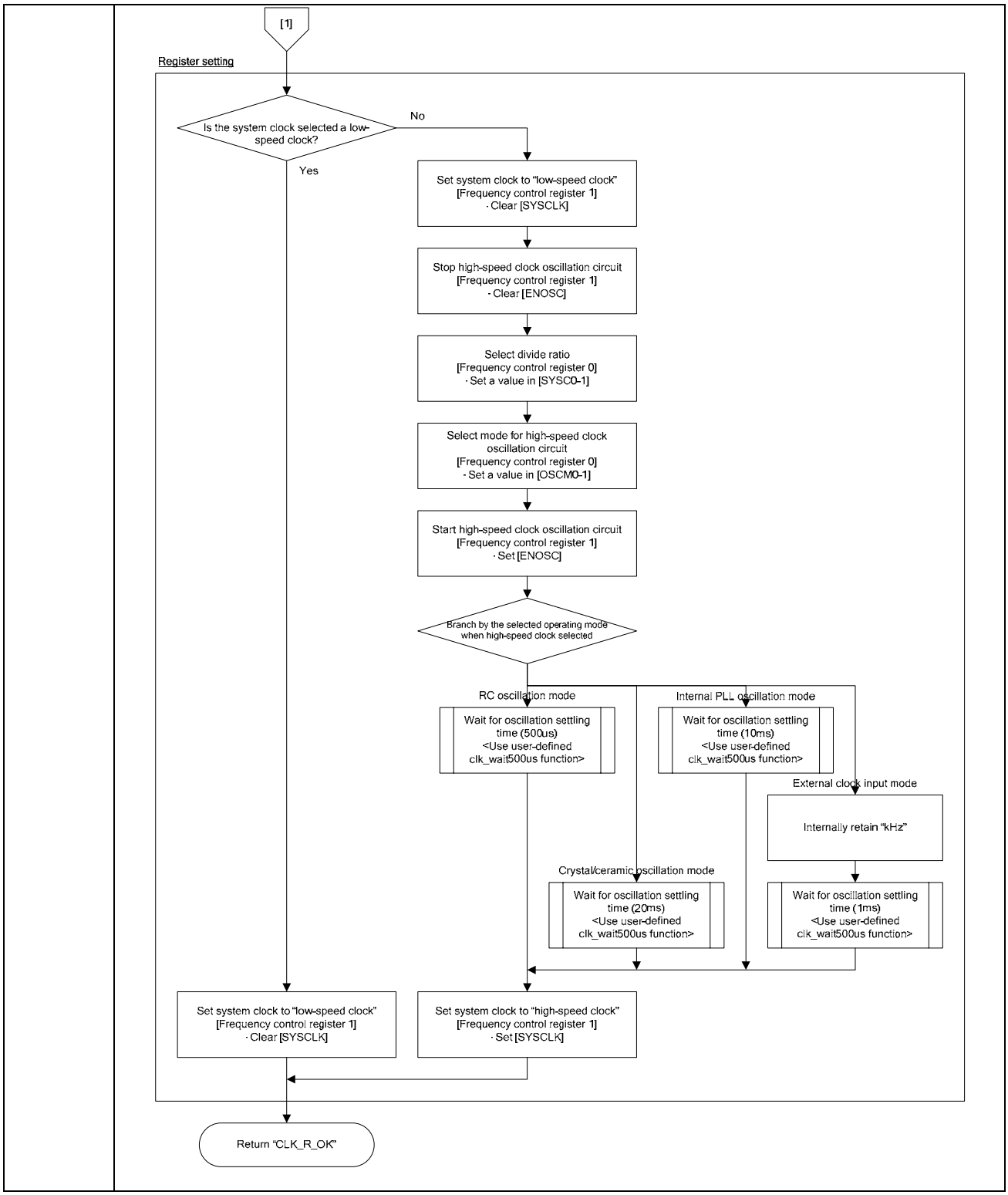
3.13.6.1. clk_setSysclk Function

This function selects low-speed clock or high-speed clock as the system clock; selects the divide ratio of high-speed clock; selects the operating mode of the high-speed clock oscillation circuit; sets the clock frequency to be used.

Function name:	int clk_setSysclk(unsigned char sysclk, unsigned char sysc, unsigned char oscm, unsigned short kHz)
Arguments:	unsigned char sysclk … Select system clock Low-speed clock: CLK_SYSCLK_LSCLK(=0) High-speed clock: CLK_SYSCLK_HSCLK(=1) unsigned char sysc … Select the divide ratio of high-speed clock [When low-speed clock is selected] No reference [When high-speed clock is selected] OSCLK: CLK_SYSC_OSCLK(=0) 1/2OSCLK: CLK_SYSC_OSCLK_DIV2(=1) 1/4OSCLK: CLK_SYSC_OSCLK_DIV4(=2) 1/8OSCLK: CLK_SYSC_OSCLK_DIV8(=3) unsigned char oscm … Selects the mode of the high-speed clock generation circuit. [When low-speed clock is selected] No reference [When high-speed clock is selected] RC oscillation mode: CLK_OSCM_RC(=0) Crystal/ceramic oscillation mode: CLK_OSCM_CRYSTAL(=1) Internal PLL oscillation mode: CLK_OSCM_PLL(=2) External clock input mode: CLK_OSCM_EXTCLK(=3) unsigned short kHz … Input frequency (kHz) [When low-speed clock is selected] No reference [When high-speed clock is selected] Referenced only when external clock input mode is selected.
Return values:	int Initialization succeeded: CLK_R_OK(=0) The selected system clock is outside the range: CLK_R_ERR_SCLK(=-1) The selected divide ratio is outside the range: CLK_R_ERR_SYSC(=-2) The selected high-speed clock is outside the range: CLK_R_ERR_OSCM(=-3)
Processing:	See next page.



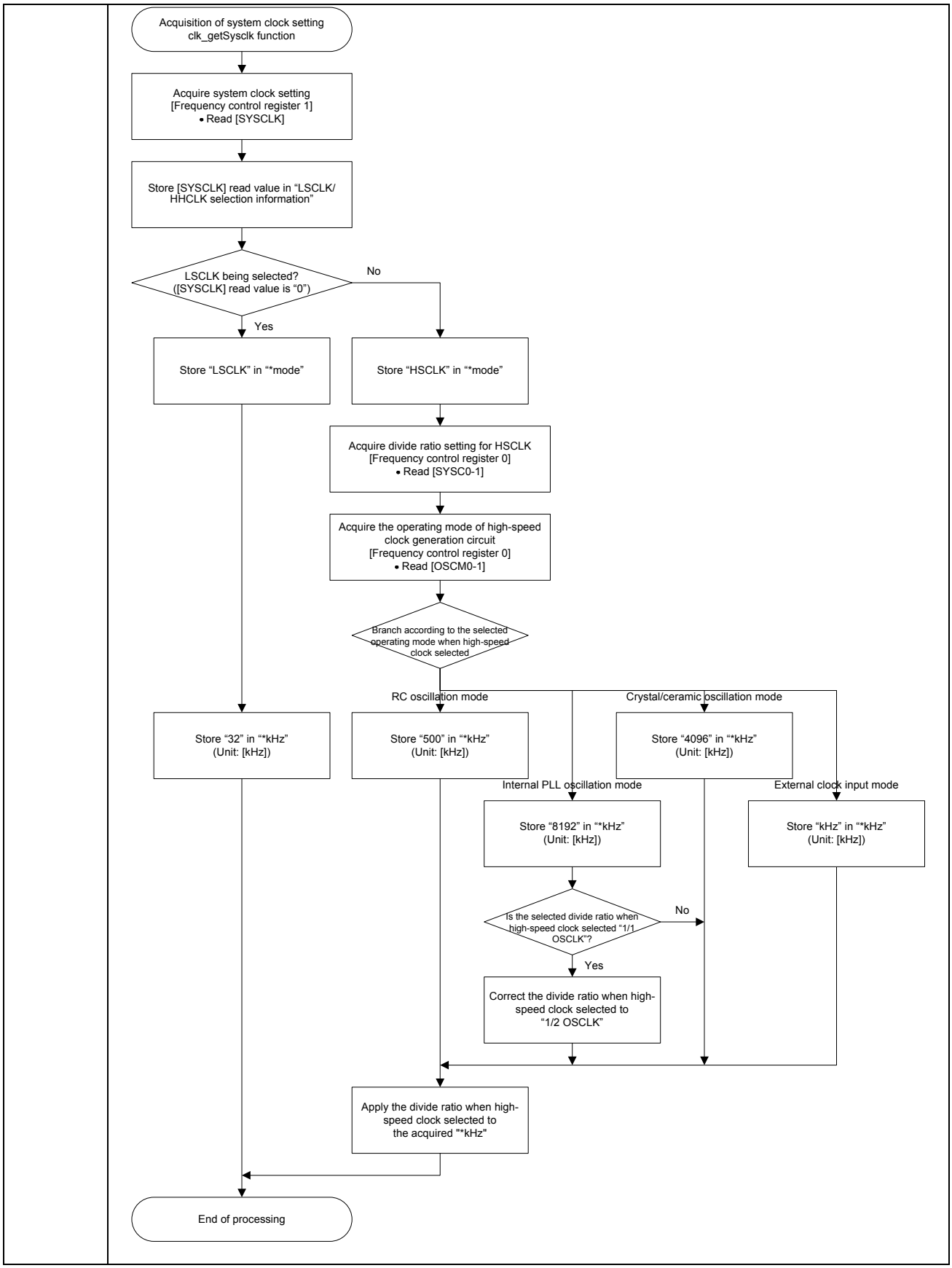
Continued on next page


[1]
Register setting
Is the system clock selected a low-speed clock?
No
Yes
Set system clock to "low-speed clock"
[Frequency control register 1]
· Clear [SYSCLK]
Stop high-speed clock oscillation circuit
[Frequency control register 1]
· Clear [ENOSC]
Select divide ratio
[Frequency control register 0]
· Set a value in [SYSC0-1]
Select mode for high-speed clock oscillation circuit
[Frequency control register 0]
· Set a value in [OSCM0-1]
Start high-speed clock oscillation circuit
[Frequency control register 1]
· Set [ENOSC]
Branch by the selected operating mode when high-speed clock selected
RC oscillation mode
Internal PLL oscillation mode
External clock input mode
Crystal/ceramic oscillation mode
Wait for oscillation settling time (500us)
<Use user-defined clk_wait500us function>
Wait for oscillation settling time (10ms)
<Use user-defined clk_wait500us function>
Internally retain "kHz"
Wait for oscillation settling time (20ms)
<Use user-defined clk_wait500us function>
Wait for oscillation settling time (1ms)
<Use user-defined clk_wait500us function>
Set system clock to "low-speed clock"
[Frequency control register 1]
· Clear [SYSCLK]
Set system clock to "high-speed clock"
[Frequency control register 1]
· Set [SYSCLK]
Return "CLK_R_OK"

3.13.6.2. clk_getSysclk Function

This function acquires the system clock settings (LSCLK/HSCLK selection, system clock frequency).

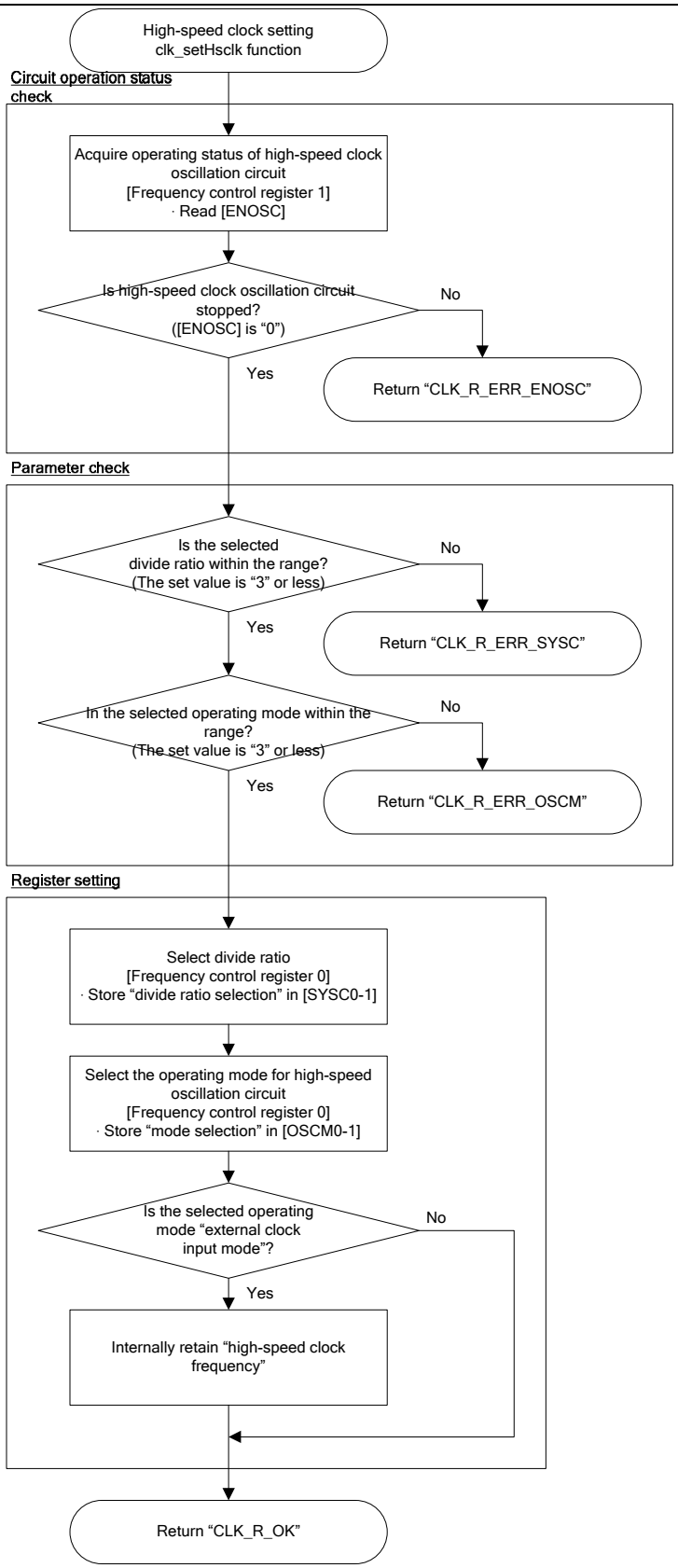
Function name:	void clk_getSysclk(unsigned char *mode, unsigned short *kHz)
Arguments:	unsigned char *mode … Pointer to the area that stores LSCLK/HSCLK selection information Low-speed clock: CLK_SYSCLK_LSCLK(=0) High-speed clock: CLK_SYSCLK_HSCLK(=1) unsigned short * kHz … Area that stores the frequency (stores a value in 1-kHz units) - The value should be 32.768 kHz during operation with a “low-speed clock”; however, since “1-kHz units” is specified for this argument, “32” is stored. For this reason, if “low-speed clock” is returned to the argument “mode”, it is recommended to use a value of “32.768 kHz” without using the value stored in this argument.
Return values:	None
Processing:	See next page.


Acquisition of system clock setting
clk_getSysclk function
Acquire system clock setting
[Frequency control register 1]
• Read [SYSCLK]
Store [SYSCLK] read value in "LSCLK/
HHCLK selection information"
LSCLK being selected?
([SYSCLK] read value is "0")
No
Yes
Store "LSCLK" in "*mode"
Store "HSCLK" in "*mode"
Acquire divide ratio setting for HSCLK
[Frequency control register 0]
• Read [SYSC0-1]
Acquire the operating mode of high-speed
clock generation circuit
[Frequency control register 0]
• Read [OSCM0-1]
Branch according to the selected
operating mode when high-speed
clock selected
RC oscillation mode
Crystal/ceramic oscillation mode
Internal PLL oscillation mode
External clock input mode
Store "32" in "*kHz"
(Unit: [kHz])
Store "500" in "*kHz"
(Unit: [kHz])
Store "4096" in "*kHz"
(Unit: [kHz])
Store "8192" in "*kHz"
(Unit: [kHz])
Store "kHz" in "*kHz"
(Unit: [kHz])
Is the selected divide ratio when
high-speed clock selected "1/1
OSCLK"?
No
Yes
Correct the divide ratio when high-
speed clock selected to
"1/2 OSCLK"
Apply the divide ratio when high-
speed clock selected to
the acquired "*kHz"
End of processing

3.13.6.3. clk_setHsclk Function

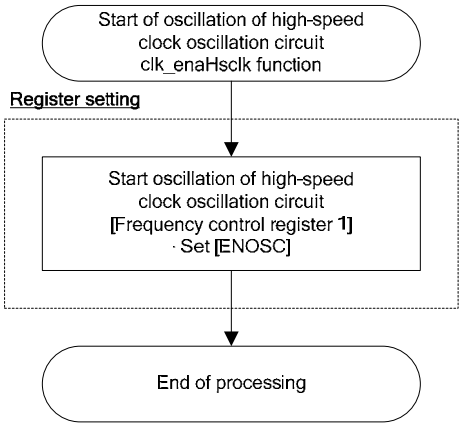
This function selects the divide ratio of high-speed clock; selects the operating mode of the high-speed clock generation circuit; sets the clock frequency to be used.

Function name:	int clk_setHsclk(unsigned char sysc, unsigned char oscm, unsigned short kHz)
Arguments:	unsigned char sysc …Select the divide ratio of high-speed clock OSCLK: CLK_SYSC_OSCLK(=0) 1/2OSCLK: CLK_SYSC_OSCLK_DIV2(=1) 1/4OSCLK: CLK_SYSC_OSCLK_DIV4(=2) 1/8OSCLK: CLK_SYSC_OSCLK_DIV8(=3) unsigned char oscm … Select the operating mode of the high-speed clock generation circuit RC oscillation mode: CLK_OSCM_RC(=0) Crystal/ceramic oscillation mode: CLK_OSCM_CRYSTAL(=1) Internal PLL oscillation mode: CLK_OSCM_PLL(=2) External clock input mode: CLK_OSCM_EXTCLK(=3) unsigned short kHz … Input frequency (kHz). Referenced only when external clock input mode is selected.
Return values:	int Setting succeeded: CLK_R_OK(=0) The selected divide ratio is outside the range: CLK_R_ERR_SYSC(=-2) The selected high-speed mode is outside the range: CLK_R_ERR_OSCM(=-3) High-speed oscillation is operating: CLK_R_ERR_ENOSC(=-4)
Processing:	See next page.


High-speed clock setting
clk_setHsclk function
Circuit operation status check
Acquire operating status of high-speed clock oscillation circuit
[Frequency control register 1]
· Read [ENOSC]
Is high-speed clock oscillation circuit stopped?
([ENOSC] is "0")
No
Yes
Return "CLK_R_ERR_ENOSC"
Parameter check
Is the selected divide ratio within the range?
(The set value is "3" or less)
No
Yes
Return "CLK_R_ERR_SYSC"
In the selected operating mode within the range?
(The set value is "3" or less)
No
Yes
Return "CLK_R_ERR_OSCM"
Register setting
Select divide ratio
[Frequency control register 0]
· Store "divide ratio selection" in [SYSC0-1]
Select the operating mode for high-speed oscillation circuit
[Frequency control register 0]
· Store "mode selection" in [OSCM0-1]
Is the selected operating mode "external clock input mode"?
No
Yes
Internally retain "high-speed clock frequency"
Return "CLK_R_OK"

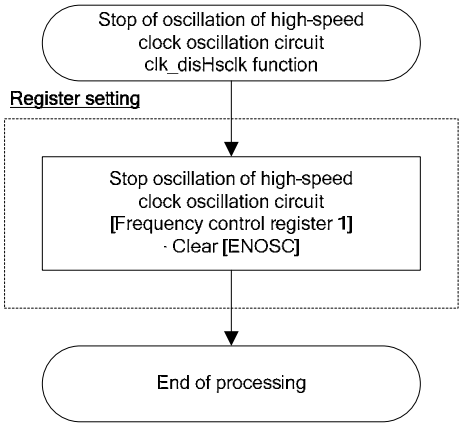
3.13.6.4. clk_enaHsclk Function

This function starts the operation of the high-speed clock oscillation circuit.

Function name:	void clk_enaHsclk(void)
Arguments:	None
Return values:	None
Processing:	

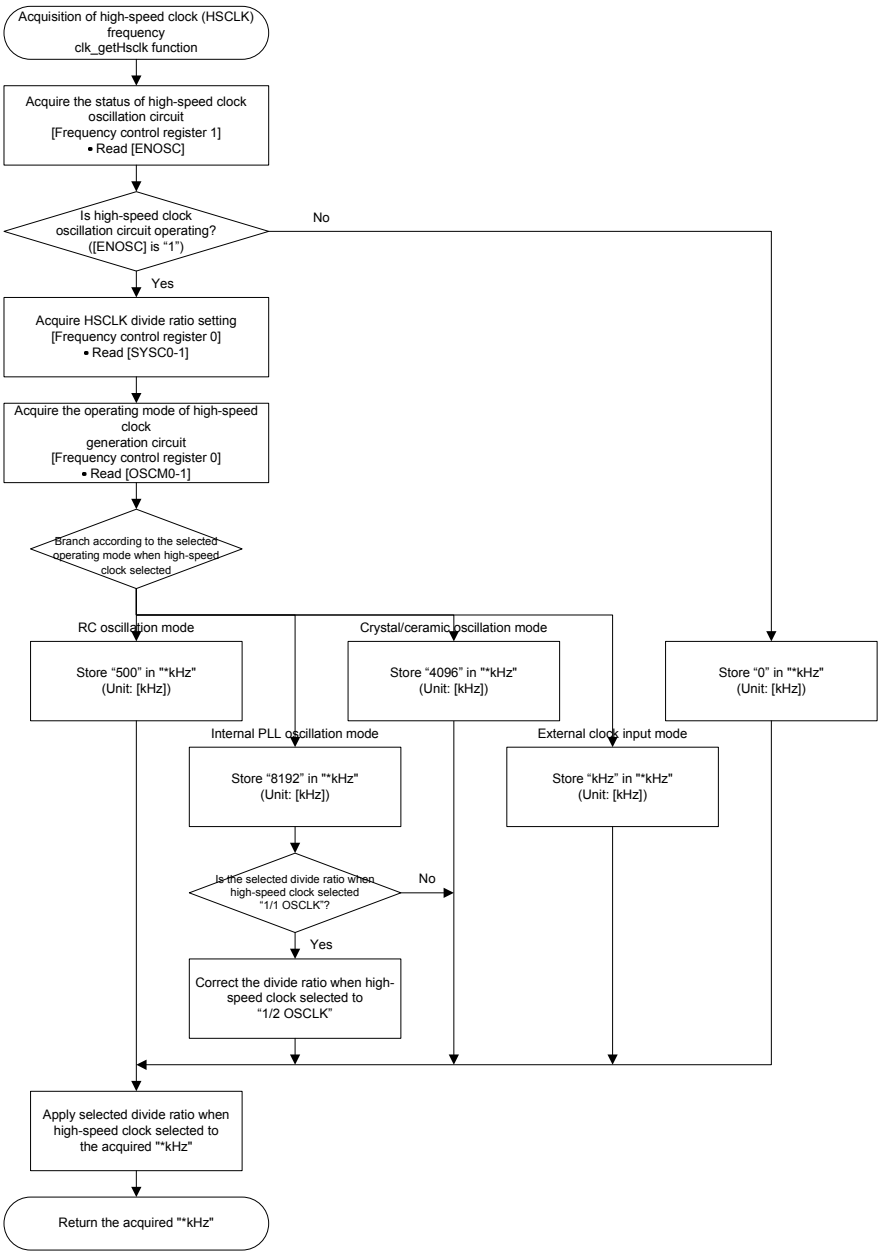
3.13.6.5. clk_disHsclk Function

This function stops the operation of the high-speed clock oscillation circuit.

Function name:	void clk_disHsclk(void)
Arguments:	None
Return values:	None
Processing:	

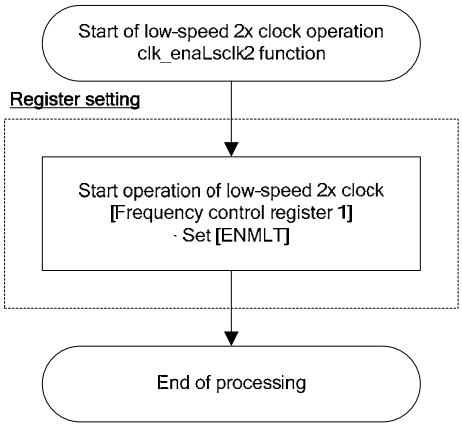
3.13.6.6. clk_getHsclk Function

This function acquires the high-speed clock frequency.

Function name:	unsigned long clk_getHsclk(void)
Arguments:	None
Return values:	unsigned short Frequency of HSCLK (kHz units)
Processing:	

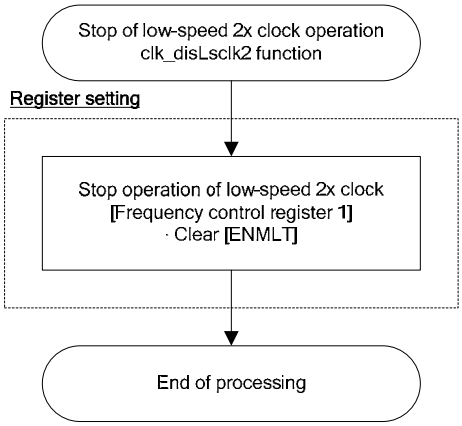
3.13.6.7. clk_enaLsclk2 Function

This function starts the operation of the low-speed 2x clock.

Function name:	void clk_enaLsclk2(void)
Arguments:	None
Return values:	None
Processing:	

3.13.6.8. clk_disLsclk2 Function

This function stops the operation of the low-speed 2x clock.

Function name:	void clk_disLsclk2(void)
Arguments:	None
Return values:	None
Processing:	

3.13.7. System Definition Function

3.13.7.1. clk_wait500us Function

This function waits for 500 μs. Create the function according to the system used.

Function name:	void clk_wait500us(void)
Arguments:	None
Return values:	None
Processing:	• Waits for 500 μs within the function.

3.14. Time Base-Counter Control Module

3.14.1. Overview of Functions

The time-base counter module controls the high-speed side time-base counter of the MCU.
Control of the time-base counter is achieved by setting and acquiring the divide ratio of the high-speed side time-base counter.

3.14.2. List of APIs

The following table lists the time-base counter control module APIs.

Table 3-50 Time-Base Counter Control Module APIs

Function name	Description
tb_setHtbdiv function	Sets the divide ratio of the time-base counter on the high-speed side.
tb_getHtbdiv function	Acquires the divide ratio of the time-base counter on the high-speed side.

3.14.3. List of Constants

The following tables list the constants used in the time-base counter control module

Table 3-51 Constants for Arguments

Constant name	Defined value	Description
TB_HTD_1_16	0	Divide ratio of HTB: 1/16
TB_HTD_1_15	1	Divide ratio of HTB: 1/15
TB_HTD_1_14	2	Divide ratio of HTB: 1/14
TB_HTD_1_13	3	Divide ratio of HTB: 1/13
TB_HTD_1_12	4	Divide ratio of HTB: 1/12
TB_HTD_1_11	5	Divide ratio of HTB: 1/11
TB_HTD_1_10	6	Divide ratio of HTB: 1/10
TB_HTD_1_9	7	Divide ratio of HTB: 1/9
TB_HTD_1_8	8	Divide ratio of HTB: 1/8
TB_HTD_1_7	9	Divide ratio of HTB: 1/7
TB_HTD_1_6	10	Divide ratio of HTB: 1/6
TB_HTD_1_5	11	Divide ratio of HTB: 1/5
TB_HTD_1_4	12	Divide ratio of HTB: 1/4
TB_HTD_1_3	13	Divide ratio of HTB: 1/3
TB_HTD_1_2	14	Divide ratio of HTB: 1/2
TB_HTD_1_1	15	Divide ratio of HTB: 1/1

Table 3-52 Constants for Return Values

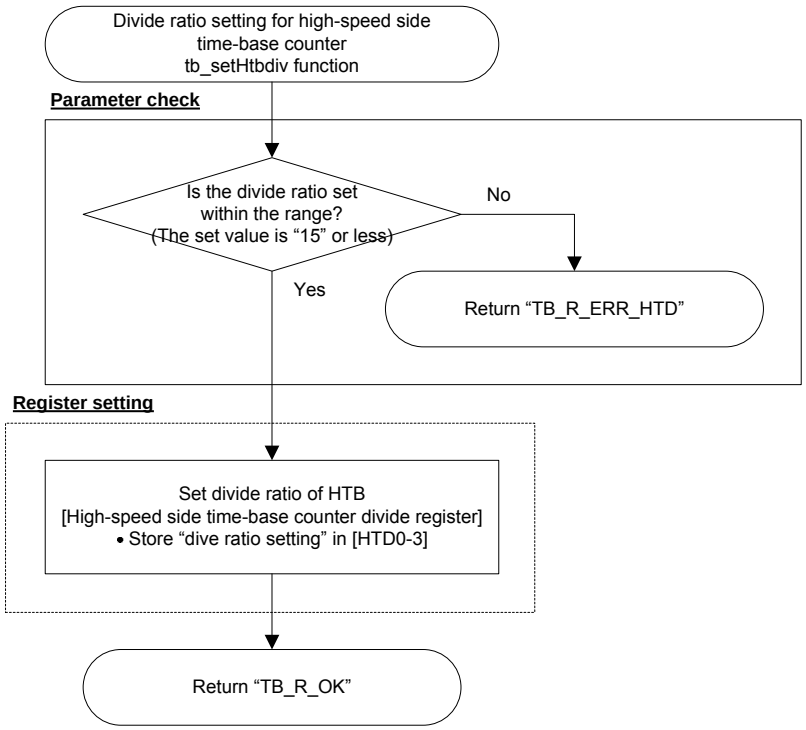
Constant name	Defined value	Description
TB_R_OK	0	Processing succeeded.
TB_R_ERR_HTD	-1	The divide ratio setting is outside the range.

3.14.4. Details of APIs

This section describes details of the time-base counter module.

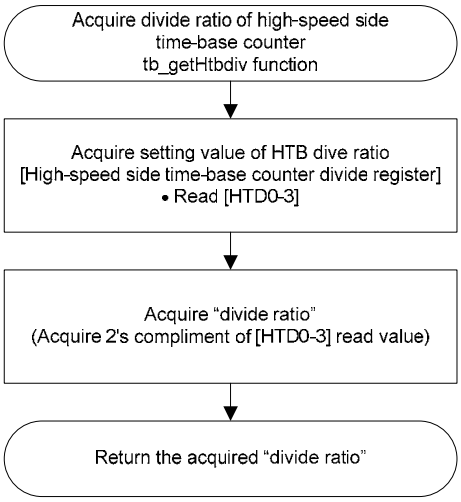
3.14.4.1. tb_setHtbdiv Function

This function sets the divide ratio of the time-base counter on the high-speed side.

Function name:	int tb_setHtbdiv(unsigned char htd)
Arguments:	unsigned char htb … Divide ratio setting Frequency divided by 16 (HTBCLK=256kHz): TB_HTD_1_16(=0) … (*For details see Table 3-51) 1/1 divide (HTBCLK=4096kHz): TB_HTD_1_1(=15)
Return values:	int Setting succeeded: TB_R_OK(=0) The divide ratio setting is outside the range: TB_R_ERR_HTD(=-1)
Processing:	

3.14.4.2. tb_getHtbdiv Function

This function acquires the divide ratio of the high-speed side time-base counter.

Function name:	unsigned char tb_getHtbdiv(void)
Arguments:	None
Return values:	unsigned char Divide ratio (1 to 16)
Processing:	

3.15. 1kHz Timer Control Module

3.15.1. Overview of Functions

The 1kHz timer control module controls the 1kHz timer of the MCU.

3.15.2. List of APIs

The following table lists the 1kHz timer control module APIs.

Table 3-53 List of APIs

Function name	Description
t1k_init function	Initializes the 1kHz timer interrupt.
t1K_start function	Starts timer operation.
t1k_stop function	Stops timer operation.
t1k_getT1KCR function	Reads the T1KCR register.
t1k_clrT1KCR function	Clears the contents of the T1KCR register.
t1k_checkOvf function	Checks the status of the timer overflow flag.

3.15.3. List of Constants

The following table lists the constants used in the kHz timer control module.

Table 3-54 List of Constants for APIs

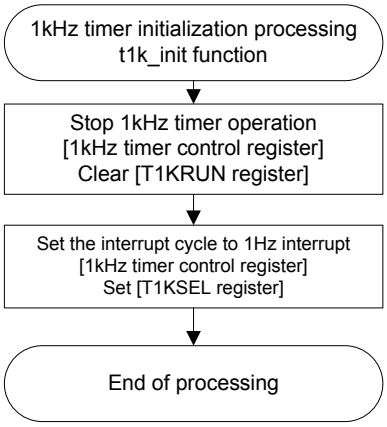
Constant name	Defined value	Description
T1KSEL_10HZ	0	Selects the interrupt cycle of the 1 kHz timer: Selects 10Hz interrupt.
T1KSEL_1HZ	1	Selects the interrupt cycle of the 1 kHz timer: Selects 1Hz interrupt.

3.15.4. Details of APIs

This section describes details of the 1kHz timer control module APIs.

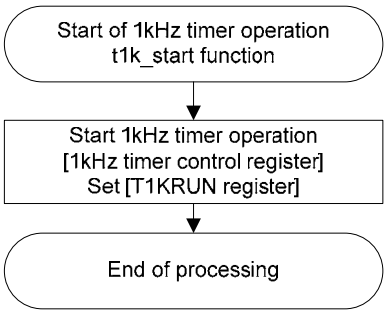
3.15.4.1. t1k_init Function

This function sets the 1Hz/10Hz interrupts, stops the timer, and clears T1KCR.

Function name:	void t1k_init(unsigned char set_T1KSET_Hz)
Arguments:	unsigend char set_T1KSET_Hz Timer interrupt setting value 10Hz : T1KSEL_10HZ(=0) 1Hz : T1KSEL_1HZ(=1)
Return values:	None
Processing:	

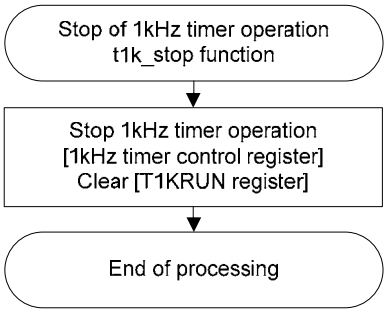
3.15.4.2. t1k_start Function

This function starts the 1kHz timer.

Function name:	void t1k_start(void)
Arguments:	None
Return values:	None
Processing:	

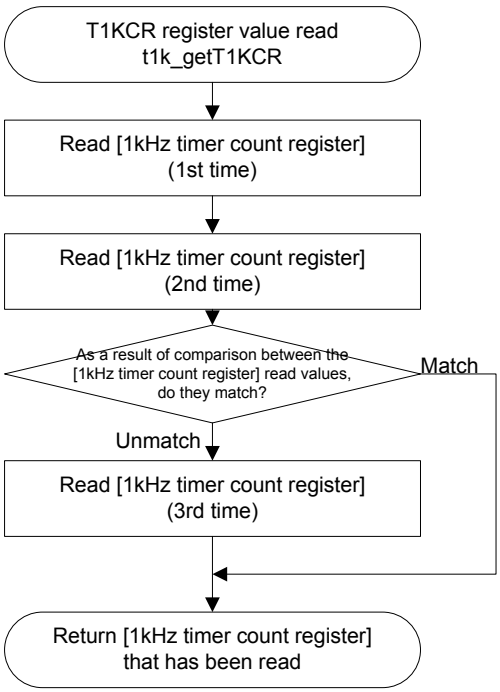
3.15.4.3. t1k_stop Function

This function stops the 1kHz timer.

Function name:	void t1k_stop(void)
Arguments:	None
Return values:	None
Processing:	

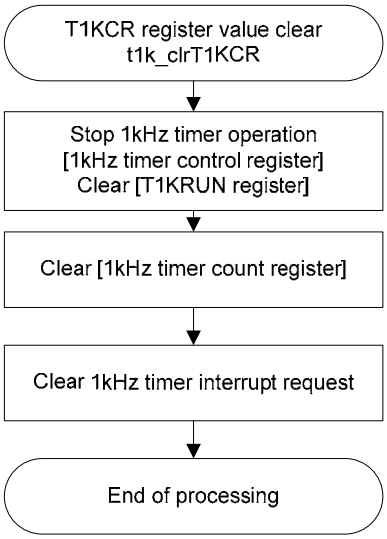
3.15.4.4. t1k_getT1KCR Function

This function reads the value of the T1KCR register.

Function name:	unsigned short t1k_getT1KCR(void)
Arguments:	None
Return values:	unsigned short T1KCR register value
Processing:	

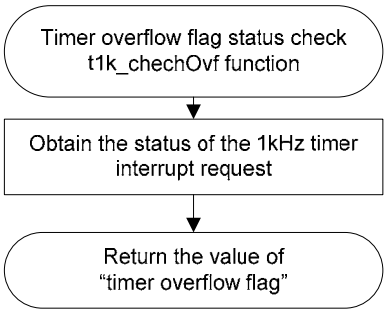
3.15.4.5. t1k_clrT1KCR Function

This function stops the 1kHz timer and clears the T1KCR register value.

Function name:	void t1k_clrT1KCR(void)
Arguments:	None
Return values:	None
Processing:	

3.15.4.6. t1km_checkOvf Function

This function checks the status of the timer overflow flag.

Function name:	unsigned char t1k_chechOvf(void)
Arguments:	None
Return values:	Status of the timer overflow flag 1: Overflow occurred 0: No overflow occurred
Processing:	

3.16. Stopwatch Module

3.16.1. Overview of Functions

The stopwatch module provides a stopwatch function by using the 1kHz timer module.

3.16.2. List of APIs

The following table lists the stopwatch module APIs.

Table 3-55 Stopwatch Module APIs

Function name	Description
chrono_init function	Initializes the stopwatch.
chrono_start function	Starts the stopwatch.
chrono_stop function	Stops the stopwatch.
chrono_getHz100 function	Reads the value of a tenth and a hundredth of a second.
chrono_getHz1000 function	Reads the value of a tenth, a hundredth, and a thousandth of a second.
chrono_getTime function	Reads the value of minute, second, a tenth, a hundredth, and a thousandth of a second.
chrono_checkOvf function	Checks the stopwatch's 60-minute overflow flag.
chrono_clrOvf function	Clears the stopwatch's 60-minute overflow flag.
chrono_int32Hz function	Performs 32Hz timer interrupt processing for screen update.
chrono_int1kHz_Spilt function	Performs 1kHz timer interrupt processing during SPLIT.
chrono_int1kHz_Run function	Performs 1kHz timer interrupt processing during RUN.
chrono _Stop_S1 function	Performs S1 key event processing while the stopwatch status is STOP.
chrono _Stop_S2 function	Performs S2 key event processing while the stopwatch status is STOP.
chrono _RUN_S1 function	Performs S1 key event processing while the stopwatch status is RUN or SPLIT.
chrono _RUN_S2 function	Performs S2 key event processing while the stopwatch status is RUN or SPLIT.
chrono_Int_S1 function	Performs S1 key event interrupt processing.
chrono_Int_S2 function	Performs S2 key event interrupt processing.
chrono_get_ProcSts function	Obtains the stopwatch status.
chrono_get_DspReq function	Obtains display update request.
chrono_clr_DspReq function	Clears display update request.
chrono_get_DspTim function	Obtains the display time.

3.16.3. List of Constants

The following table lists the constants used in the stopwatch module.

Table 3-56 Stopwatch Module Constants

Constant name	Defined value	Description
CHR_STS_STOP	0	Stopwatch status: STOP state
CHR_STS_RUN	1	Stopwatch status: RUN state
CHR_STS_SPLIT	2	Stopwatch status: SPLIT state

3.16.4. Structure

This section describes the structures used in the stopwatch module. Set each parameter value using binary-coded decimal (BCD) numbers.

■ Watch setting parameters

```
typedef struct {
    unsigned char    1ms;    // 1/1000 data (0x00 to 0x99)
    unsigned char    ms;     // 1/10 and 1/100 data (0x00 to 0x99)
    unsigned char    sec;    // Second data (0x00 to 0x59)
    unsigned char    min;    // Minute data (0x00 to 0x59)
} t1K_Time;
```

3.16.5. List of Variables

The following table lists the variables used in the stopwatch module.

Table 3-57 List of Variables

Variable name	Initial value	Description
t1K_Time _chrono_Tim	1ms :0 ms :0 sec :0 min:0	Clock time (internal time)
t1K_Time _Dsp_Tim	1ms :0 ms :0 sec :0 min:0	Clock time for display
unsigned char_Ovf_Flg	0	Stopwatch overflow flag
unsigned char_chrono_proc_Sts	CHR_STS_STOP	Stopwatch status
static const vfv _chrono_tbls_S1_Event[3](*1)	0	S1 key event processing function table
static const vfv _chrono_tbls_S2_Event[3](*1)	0	S2 key event processing function table

*1: The interface specification of the functions to be called upon generation of the S1 or S2 key event is as follows:
typedef void(*vfv)(void);
Also, Table 3-58 lists the functions to be executed upon generation of the S1 or S2 key event.

Table 3-58 S1 and S2 Key Event Processing Function Table

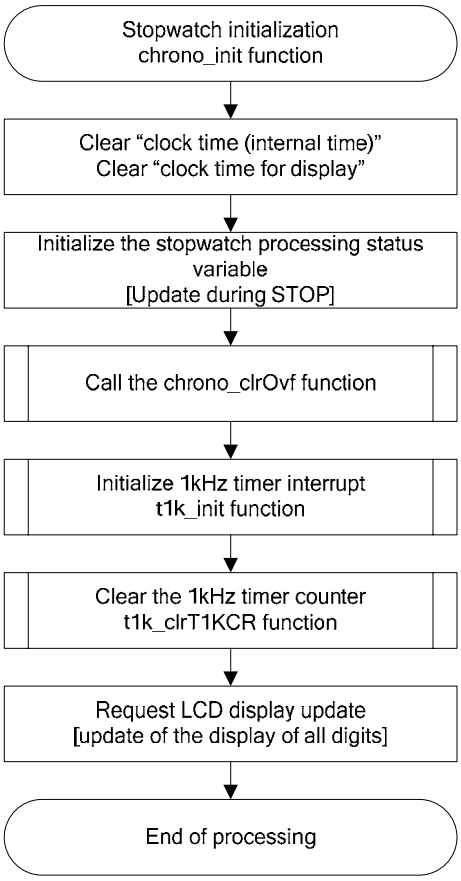
Index(status)	S1 key event table _chrono_tbls_S1_Event	S2 key event table _chrono_tbls_S2_Event
0 (STOP)	chrono_Stop_S1	chrono_Stop_S2
1 (RUN)	chrono_Run_S1	chrono_Run_S2
2 (SPLIT)	chrono_Run_S1	chrono_Run_S2

3.16.6. Details of APIs

This section describes details of the stopwatch module APIs.

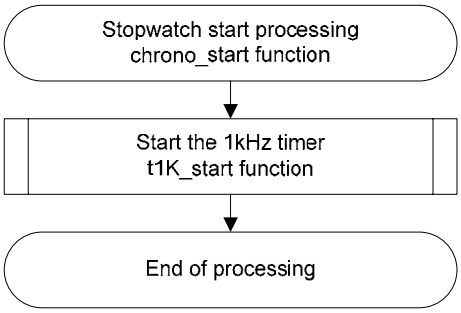
3.16.6.1. chrono_init Function

This function initializes the stopwatch.

Function name:	void chrono_init(void)
Arguments:	None
Return values:	None
Processing:	

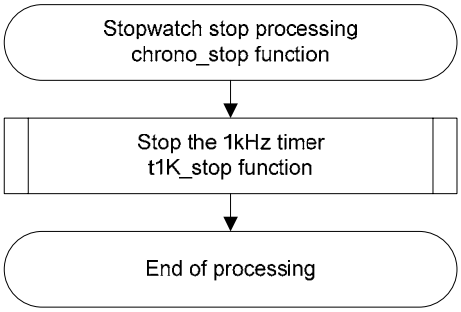
3.16.6.2. chrono_start Function

This function starts the stopwatch.

Function name:	void chrono_start(void)
Arguments:	None
Return values:	None
Processing:	

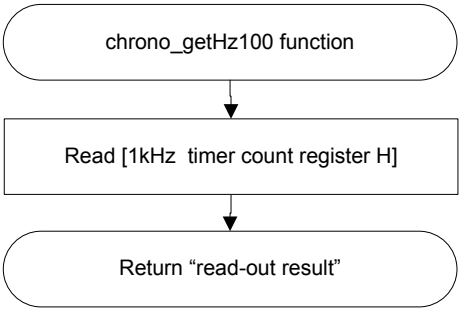
3.16.6.3. chrono_stop Function

This function stops the stopwatch.

Function name:	void chrono_stop(void)
Arguments:	None
Return values:	None
Processing:	

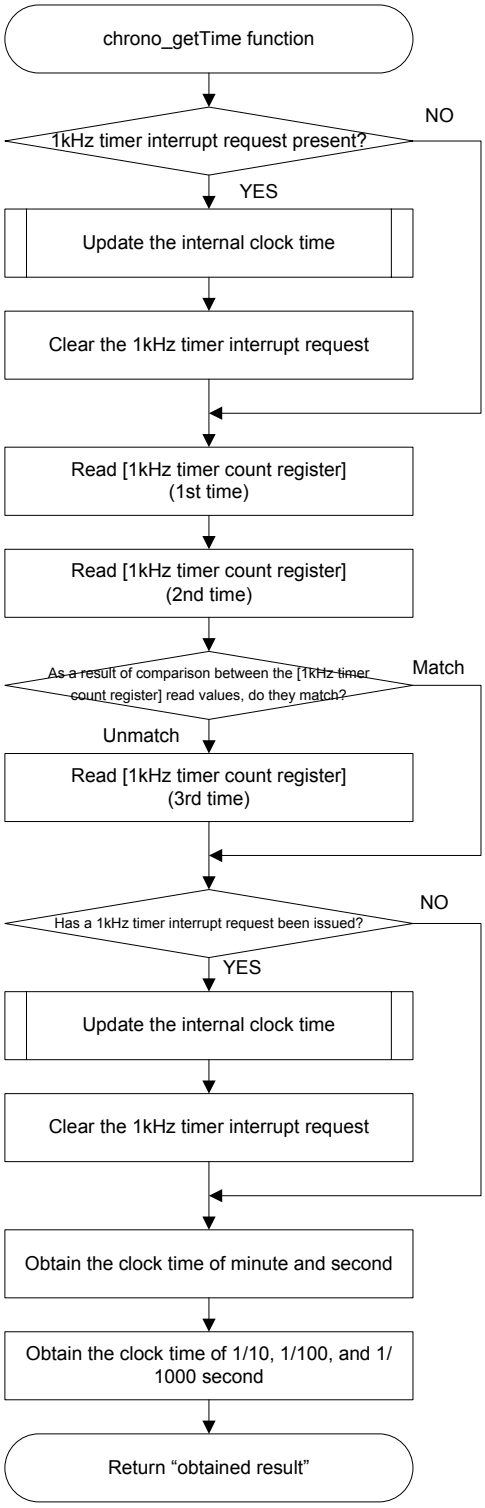
3.16.6.4. chrono_getHz100 Function

This function reads the value of a tenth and a hundredth of a second.

Function name:	unsigned char chrono_getHz100(void)
Arguments:	None
Return values:	Data of a tenth and a hundredth of a second
Processing:	

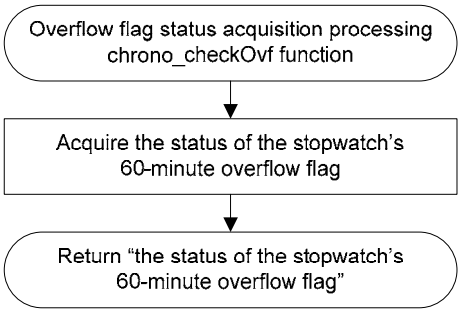
3.16.6.5. chrono_getTime Function

This function reads the value of minute, second, a tenth, a hundredth, and a thousandths of a second.

Function name:	void chrono_getTime(void)
Arguments:	t1K_Time
Return values:	None
Processing:	

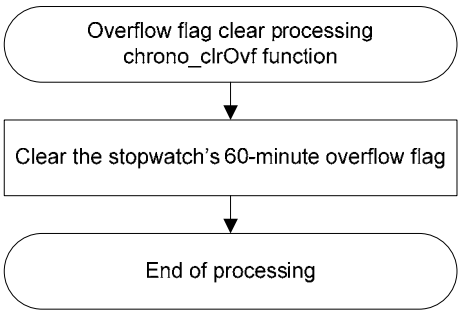
3.16.6.6. chrono_checkOvf Function

This function obtains the status of the stopwatch's 60-minute overflow flag.

Function name:	void chrono_checkOvf(void)
Arguments:	None
Return values:	Status of the stopwatch's 60-minute overflow flag
Processing:	

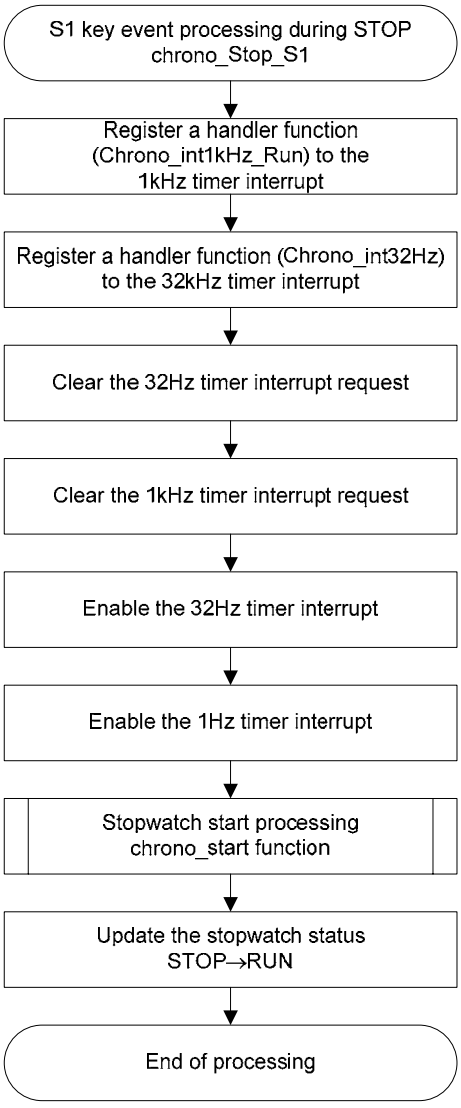
3.16.6.7. chrono_clrOvf Function

This function clears the status of the stopwatch's 60-minute overflow flag.

Function name:	unsigned char chrono_clrovf(void)
Arguments:	None
Return values:	None
Processing:	

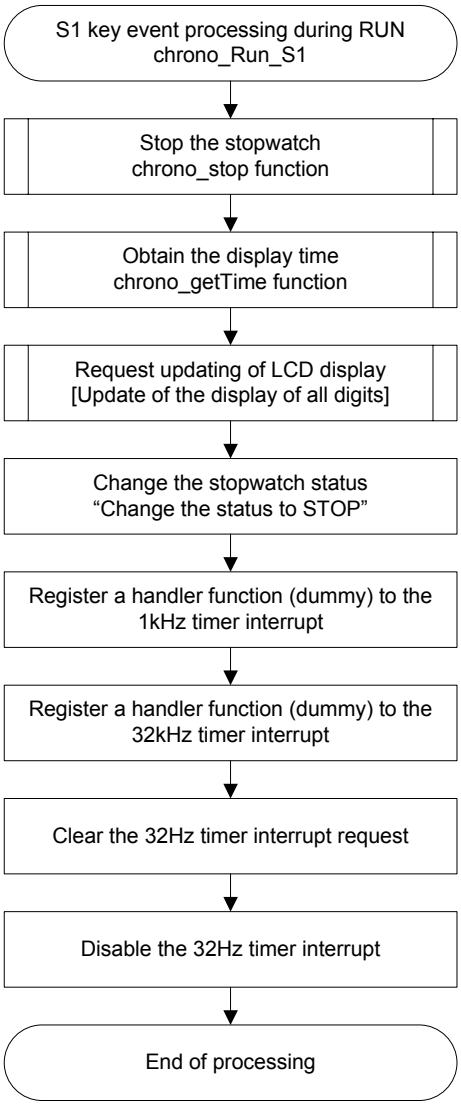
3.16.6.8. chrono_Stop_S1 Function

This function performs S1 key event processing while the stopwatch status is STOP.

Function name:	void chrono_Stop_S1 (void)
Arguments:	None
Return values:	None
Processing:	

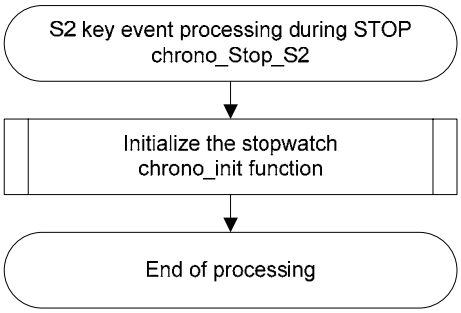
3.16.6.9. chrono_Run_S1 Function

This function performs S1 key event processing while the stopwatch status is RUN or SPLIT.

Function name:	void chrono_Run_S1(void)
Arguments:	None
Return values:	None
Processing:	

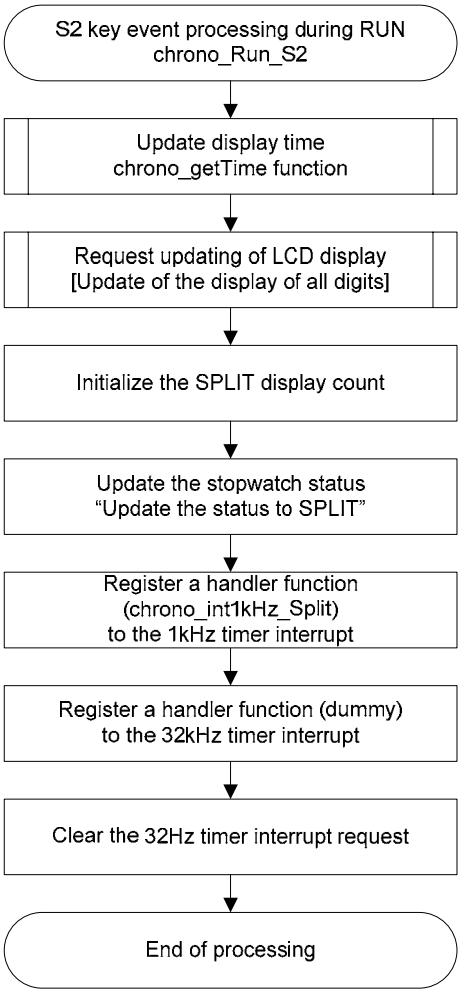
3.16.6.10. chrono_Stop_S2 Function

This function performs S2 key event processing while the stopwatch status is STOP.

Function name:	void chrono_Stop_S2(void)
Arguments:	None
Return values:	None
Processing:	

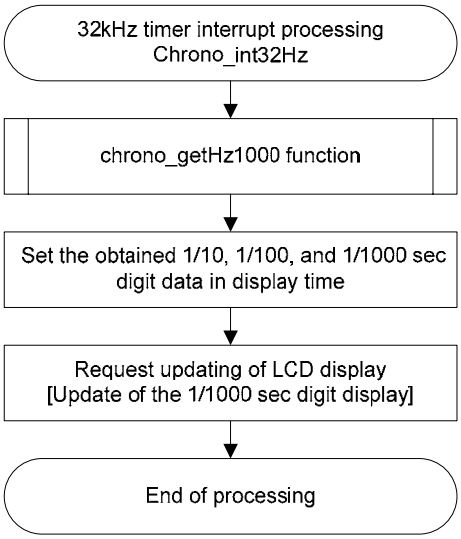
3.16.6.11. chrono_Run_S2 Function

This function performs S2 key event processing while the stopwatch status is RUN or SPLIT.

Function name:	void chrono_Run_S2 (void)
Arguments:	None
Return values:	None
Processing:	

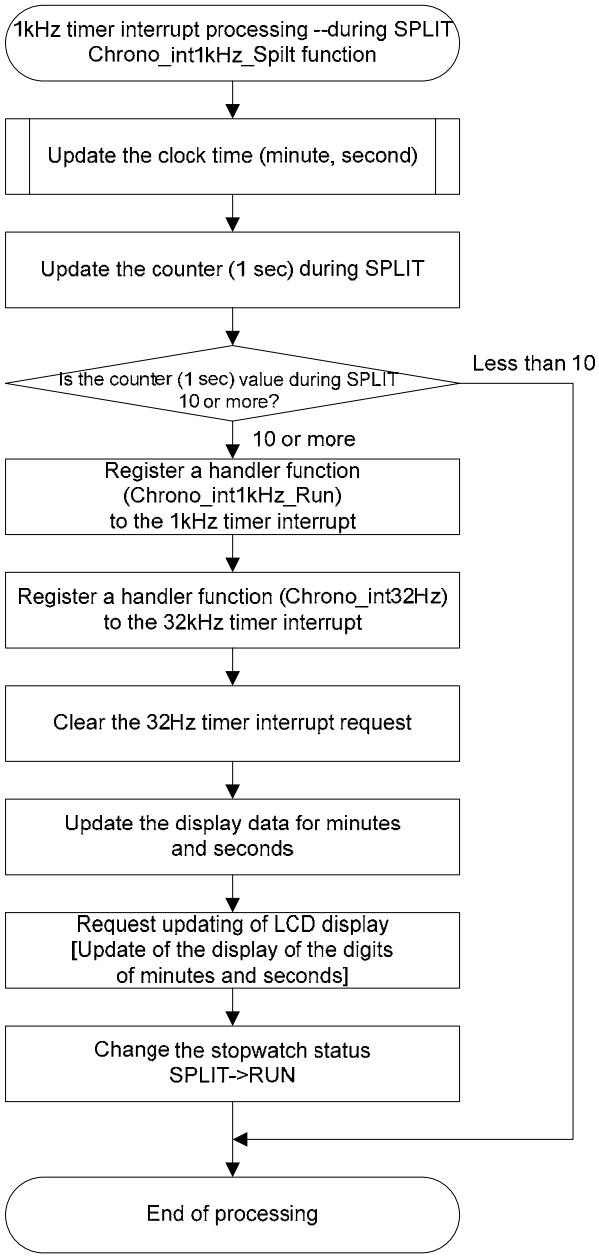
3.16.6.12. chrono_int32Hz Function

This function performs 32kHz timer interrupt processing.

Function name:	void chrono_int32Hz(void)
Arguments:	None
Return values:	None
Processing:	

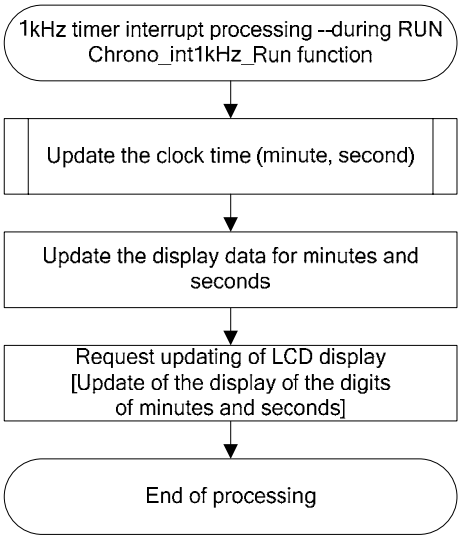
3.16.6.13. chrono_int1kHz_Split Function

This function performs 1kHz timer interrupt processing while the stopwatch status is SPLIT.

Function name:	void chrono_int1kHz_Split (void)
Arguments:	None
Return values:	None
Processing:	

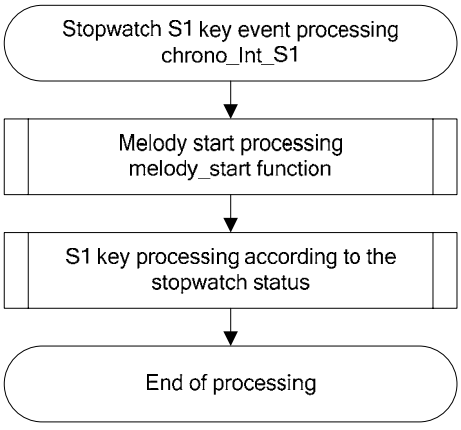
3.16.6.14. chrono_int1kHz_Run Function

This function performs 1kHz timer interrupt processing while the stopwatch status is RUN.

Function name:	void chrono_int1kHz_Run(void)
Arguments:	None
Return values:	None
Processing:	

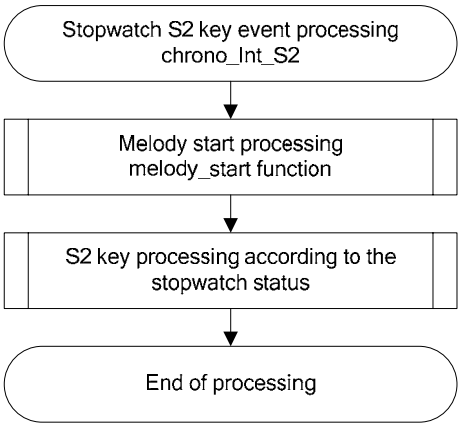
3.16.6.15. chrono_Int_S1 Function

This function performs S1 key event interrupt processing.

Function name:	void chrono_Int_S1 (void)
Arguments:	None
Return values:	None
Processing:	

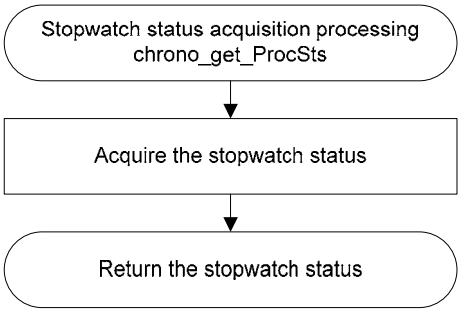
3.16.6.16. chrono_Int_S2 Function

This function performs S2 key event interrupt processing.

Function name:	void chrono_Int_S2 (void)
Arguments:	None
Return values:	None
Processing:	

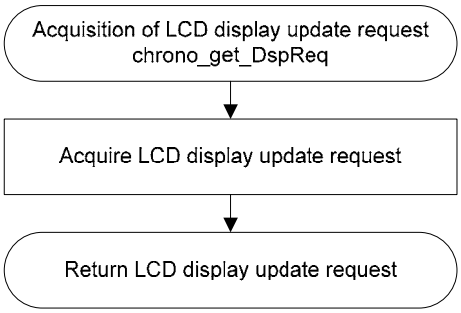
3.16.6.17. chrono_get_ProcSts Function

This function obtains the stopwatch status.

Function name:	void chrono_get_ProcSts (void)
Arguments:	None
Return values:	None
Processing:	

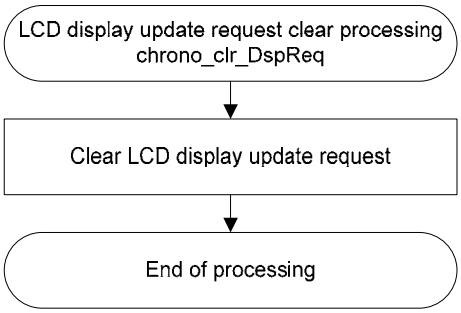
3.16.6.18. chrono_get_DspReq Function

This function acquires a display update request.

Function name:	void chrono_get_DspReq (void)
Arguments:	None
Return values:	None
Processing:	

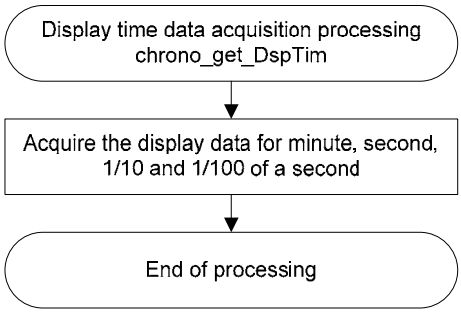
3.16.6.19. chrono_clr_DspReq Function

This function clears display update request.

Function name:	void chrono_clr_DspReq (unsigned char Req)
Arguments:	Req : Display update No. 1: Request for updating of the minute and second display 2: Request for updating of the 1/10, 1/100, and 1/1000 second display
Return values:	None
Processing:	

3.16.6.20. chrono_get_DspTim Function

This function acquires the display time.

Function name:	void chrono_get_DspTim (t1K_Time *pTim)
Arguments:	pTim: Display time storage destination address
Return values:	None
Processing:	

3.17. BLD Module

3.17.1. Overview of Functions

The BLD module controls the battery level detector (BLD) of the MCU and obtains the battery level.

3.17.2. List of APIs

The following table lists the BLD module APIs.

Table 3-59 BLD Module APIs

Function name	Description
bld_start_levelCheck function	Obtains battery level detection.
bld_getLevel function	Obtains the battery level (17 levels in all).
bld_check function	Checks whether the battery voltage is lower or higher than the threshold voltage.
bld_on function	Turns on the BLD.
bld_off function	Turns off the BLD.

3.17.3. List of Constants

The following table lists the constants used in the BLD module.

Table 3-60 BLD Module Constants

Constant name	Value	Description
BLD_CHECK_STOP	0	Indicates the state where the battery level detection by the BLD is stopped.
BLD_CHECK_FIRST	1	Indicates the state where the battery level detection has been started.
BLD_CHECK_TBL_MAX	7	Indicates the state where battery level detection has been completed.
BLD_R_HIGH	0	Indicates that the battery voltage is higher than the threshold voltage.
BLD_R_LOW	1	Indicates that the battery voltage is lower than the threshold voltage.

3.17.4. List of Variables

The following table lists the variables used in the BLD module.

Table 3-61 BLD Module Variables

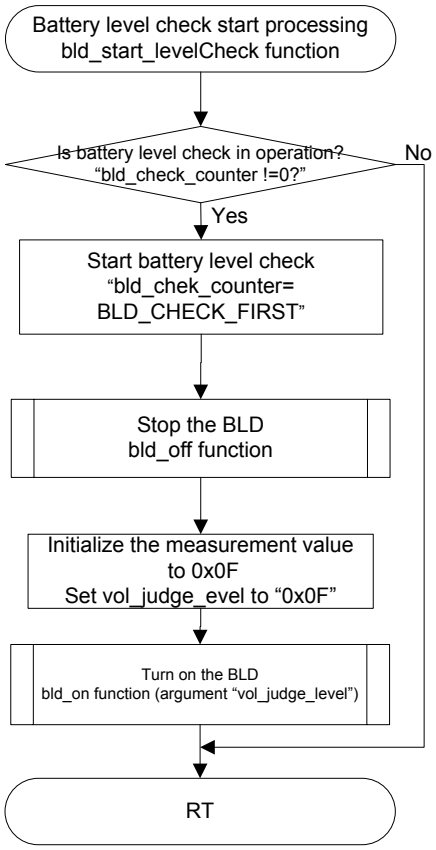
Variable name	Initial value	Description
vol_level	0	Determined measurement value of battery level, which is determined by the BLD module
vol_judge_level	0	Measurement value during execution of the BLD module
bld_check_counter	0	Indicates the status of the BLD module while it is being processed.

3.17.5. Details of APIs

This section describes details of the BLD module APIs.

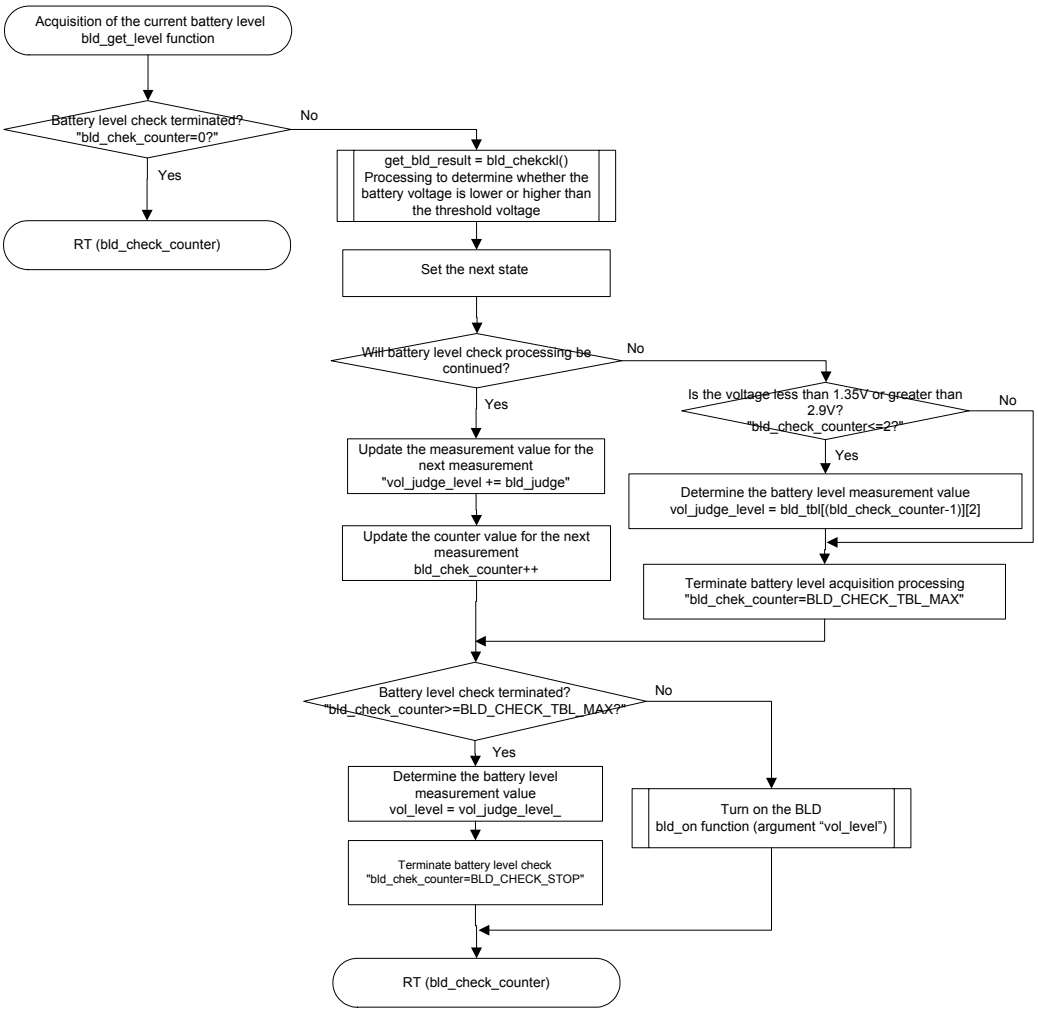
3.17.5.1. bld_start_levelCheck Function

This function starts battery level detection.

Function name:	void bld_start_levelCheck(void)
Arguments:	None
Return values:	None
Processing:	

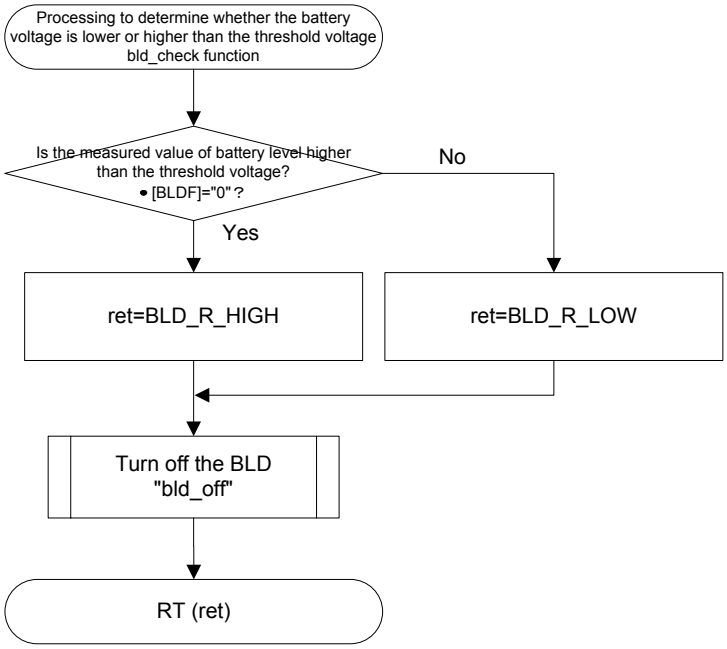
3.17.5.2. bld_getLevel Function

This function acquires the battery level. Call this function at 1-ms intervals until the return value becomes “0” (be sure to call the bld_check_start function before using this function).

Function name:	unsigned char bld_getLevel(void)
Arguments:	None
Return values:	unsigned char
Processing:	

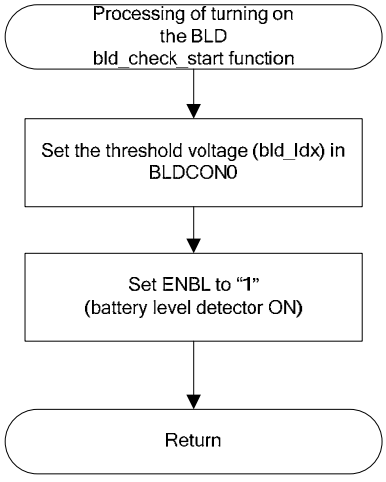
3.17.5.3. bld_check Function

This function checks whether the battery voltage is lower or higher than the threshold voltage.

Function name:	unsigned char bld_check (void)
Arguments:	None
Return values:	unsigned char Judgment result as to whether the battery voltage is lower or higher than the threshold voltage Higher than the threshold voltage: BLD_R_HIGH Lower than the threshold voltage: BLD_R_LOW
Processing:	

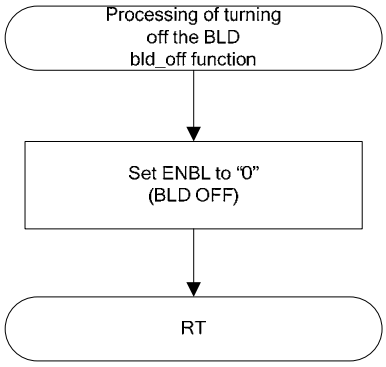
3.17.5.4. bld_on Function

This function turns on the BLD.

Function name:	Void bld_on (unsigned char bld_Idx)
Arguments:	unsigned char bld_Idx 0 to 15: “vol_level”
Return values:	None
Processing:	

3.17.5.5. bld_off Function

This function turns off the BLD.

Function name:	void bld_off (void)
Arguments:	None
Return values:	None
Processing:	

4. Revision History

Revision	Date	Page		Description
		Previous Edition	Current Edition	
1	June. 26, 2009	–	–	First edition
2	January. 27, 2010	–	37	uart_PortClear and uart_PortSet function are added.
		42	43	Setting value in UART0 baud rate register is corrected.
		52	53	Port setting of P41 and P40 is corrected to Nch open-drain output.
		109	110	Interrupts are disabled in key_getEvent function.
3	April. 16, 2010	40	40	Section title for adjustBaudrate_startCount function is added.