V850 Microcontrollers

R01AN0193EJ0100
Rev.1.00
Sep 30, 2010

V850ES/Jx3-L
Real-Time Counter (RTC Backup Mode)

Introduction

This application note is about a sample program that can be used to shift the V850ES/Jx3-L microcontroller to RTC backup mode so that the real-time counter used for the watch feature can continue operating if the power supply voltage (V_{DD}) stops. The application note describes how the sample program works, how the program should be used, and how to set and use the real-time counter.

Target Devices

μPD70F3792

μPD70F3793

μPD70F3794

μPD70F3795

μPD70F3796

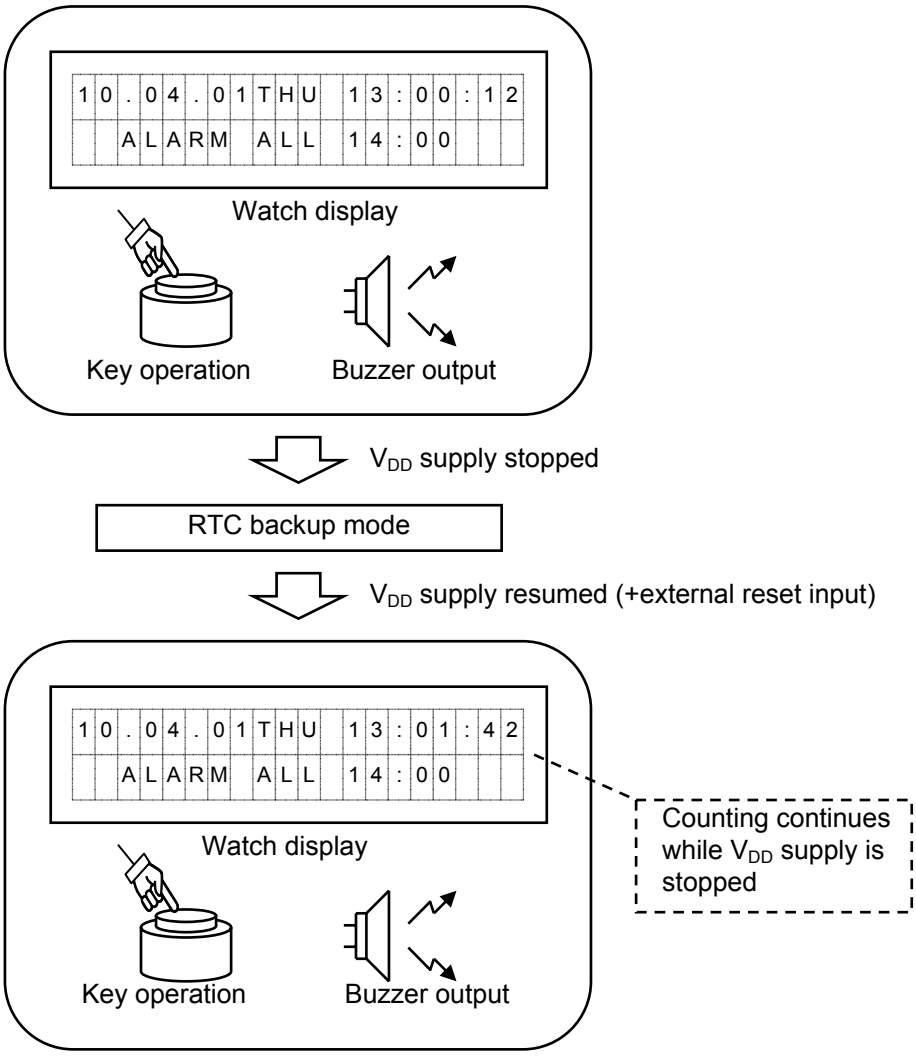
Contents

1. OVERVIEW

This document describes the real-time counter of the V850ES/JG3-L (μPD70F3792, μPD70F3793, μPD70F3794, μPD70F3795, and μPD70F3796), including the watch feature, alarm feature, watch error correction feature, and RTC backup mode.

By using the watch feature, alarm feature, and RTC backup mode as specified in the sample program, you can achieve a watch-with-alarm that continues operating even if the supply voltage (V_{DD}) stops.

Operational overview



Remarks 1. In the RTC backup mode, the subclock and real-time counter use the RTC backup power supply (RV_{DD}).

2. In this sample program, it is assumed that an electric double-layer capacitor (0.1 F) is used so that the power supply to the RTC backup power supply (RV_{DD}) can continue for a certain period if the supply voltage (V_{DD}) stops. For details about the circuits, see 2.1 Circuit Diagram.

1.1 Overview of Sample Program

This sample program is used to specify the initial settings for the real-time counter (fixed-cycle interrupt, alarm interrupt, interval interrupt), RTC backup mode, and key interrupt features. Details are provided below.

(1) Main initial settings

The main initial settings are as follows.

<Specification using option byte>

- Specify the oscillation stabilization time after a reset

<Settings for initialization after a reset>

- Specify the initial settings for RTC backup mode
  - Set the subclock oscillation mode to normal oscillation
  - Enable the RTC backup mode
- Set the system wait control register to 1 wait
- Set the on-chip debug mode register to normal operation mode
- Specify stopping the internal oscillator
- Stop watchdog timer 2
- Specify the I/O port settings
  - Specify P50 to P53 (KR0 to KR3) as key input
  - Specify P70 as buzzer control
  - Specify P90 to P96 as LCD module control
  - Set the unused ports
- Specify the low-voltage detector settings[Note 1]
  - Set the low-voltage detection level to 2.8 V
  - Enable low-voltage detection
  - Wait until the supply voltage becomes 2.8 V or higher[Note 2]
- Specify PLL mode (5 MHz x 4 = 20 MHz operation)
- Set the CPU clock oscillation stabilization time after the standby mode is released to about 1.6 ms
- Specify the real-time counter settings[Note 3]
  - Set the count start time to 13:00:00 on April 1, 2010 (Thursday)
  - Set the alarm time to 14:00 everyday
  - Set the fixed-cycle interrupt operation to a cycle of 0.5 seconds
  - Enable the alarm interrupt
  - Set the interval operation to about 3.9 ms
  - Start the count operation
- Specify the initial settings for the LCD module
- Enable the key interrupt (KR0 to KR3) operation
- Specify the interrupt settings
  - Enable the INTKR interrupt
  - Enable the INTRTC0 interrupt

- Enable the INTRTC1 interrupt
- Enable the INTLVI interrupt
- Enable acknowledgment of maskable interrupt request signals

Notes: 1. For details about the low-voltage detector, see the V850ES/Jx3-L user's manual.
2. This wait state is for satisfying the operating condition of a supply voltage of 2.7 V or higher.
3. After a reset, this setting is omitted if the real-time counter has already started operating.

(2) Processing after specifying initial settings

After specifying the initial settings, the system shifts to stop mode. An interrupt is then generated and program execution starts according to the generated interrupt.

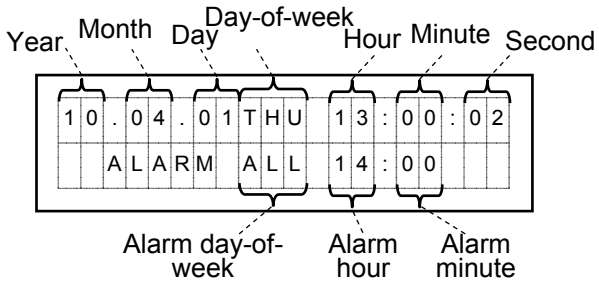
- If the 0.5-second-cycle INTRTC0 interrupt is generated, processing such as updating the watch display is executed.
- If the INTRTC1 interrupt is triggered by the occurrence of an alarm, buzzer output processing is executed.
- If the INTKR interrupt is triggered by the occurrence of a key input, key sampling starts using the 3.9-millisecond-cycle INTRTC2 interrupt. If sampling results in the detection of a valid key input, processing to refresh the watch display and the real-time counter count value is executed.
- If the INTLVI interrupt is triggered by the detection of low voltage, the system prepares to enter the RTC backup mode and then shifts to stop mode. If supply of the power supply voltage (V_{DD}) then stops, the system shifts to the RTC backup mode. However, if the INTLVI interrupt is triggered by the power supply voltage being restored from the low voltage state after the system has shifted to stop mode, the system cancels the preparation to enter the RTC backup mode and then returns to the normal operation mode.

1.2 Watch Display

This sample program specifies the use of the LCD module (character display, 20 characters x 2 rows) as a watch display.

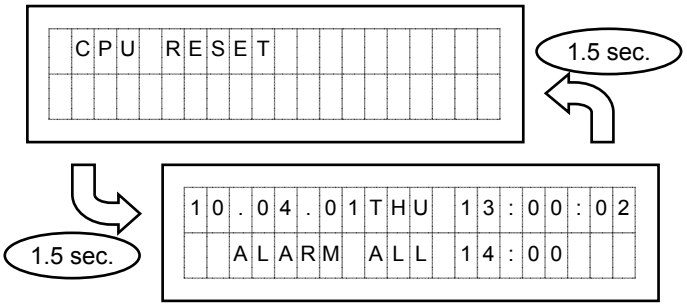
(1) Display

The day (year/month/day/day-of-week), time (hour/minute/second), and alarm setting (day-of-week/hour/minute) are displayed.



Remarks 1. When the current time and day values match the alarm settings, a buzzer sound is output for 1 second.

2. After a reset, an indication that the system has been restored from the reset state alternates with the watch display every 1.5 seconds. Inputting any key sets the display to the watch display.



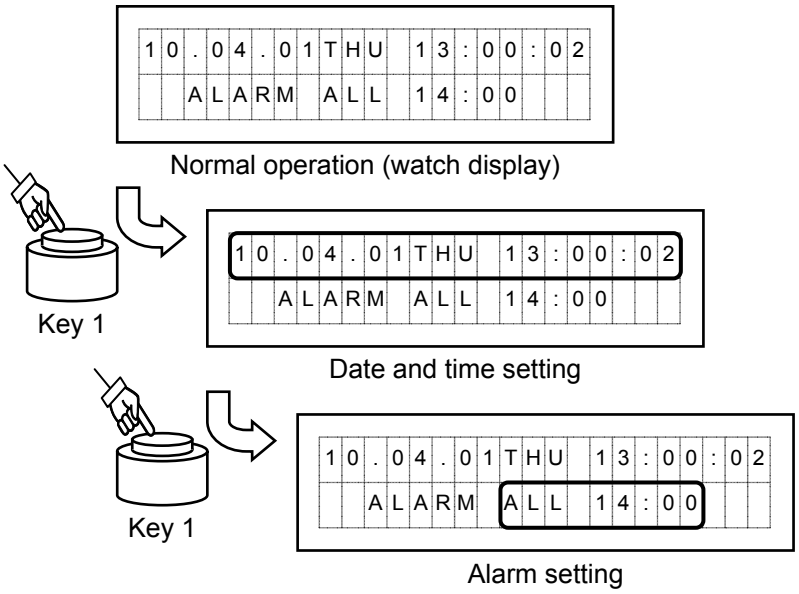
(2) Key operation

Day, time, and alarm settings are changed by key operations (keys 1 to 4).

<1> Key 1 operation

Key 1 is used to specify the mode as follows (setting key).

Normal operation (watch display) → date and time setting → alarm setting → normal operation (watch display) → …



Remark The settings being specified are ringed in black.

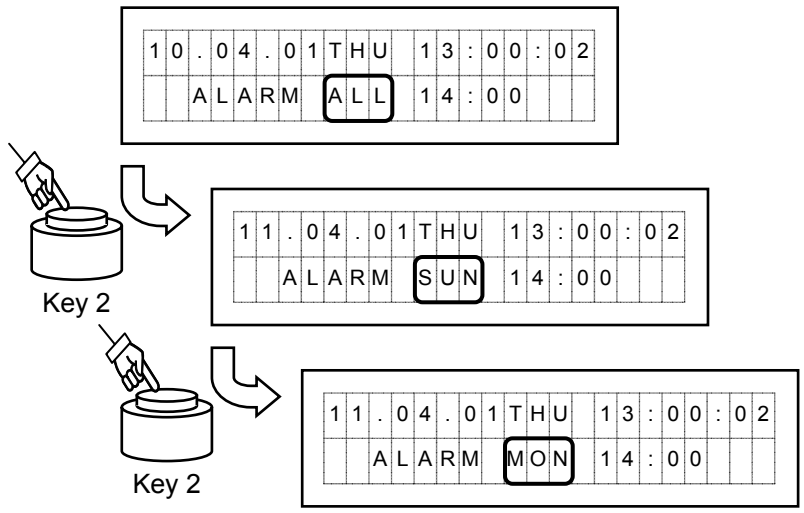
<2> Key 2 operation

Key 2 is used to increment the value (up key).

<Specifying the alarm day-of-week setting>

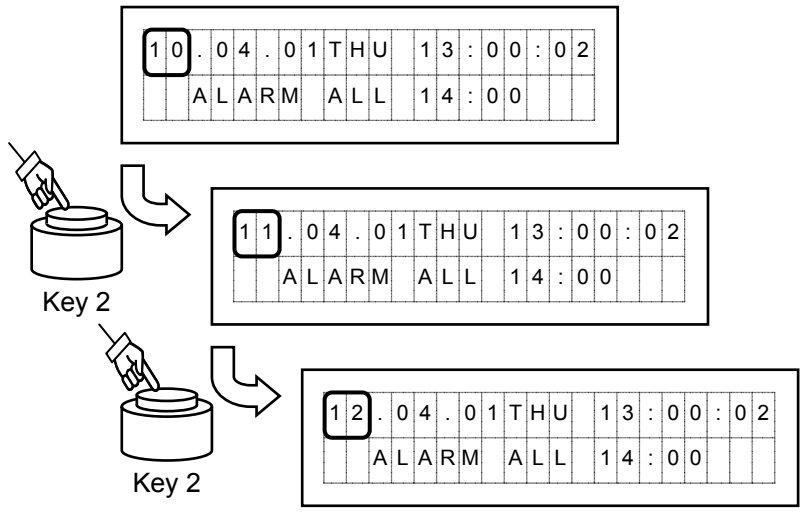
The value being set is switched as follows:

ALL[Note] → SUN → MON → TUE → WED → THU → FRI → SAT → ALL[Note] →…



<Specifying other settings>

The value being set is incremented.



Note "ALL" indicates "everyday".

Caution **The day-of-week must be set manually in accordance with the actual year, month, and day.**

Remark The settings being specified are ringed in black.

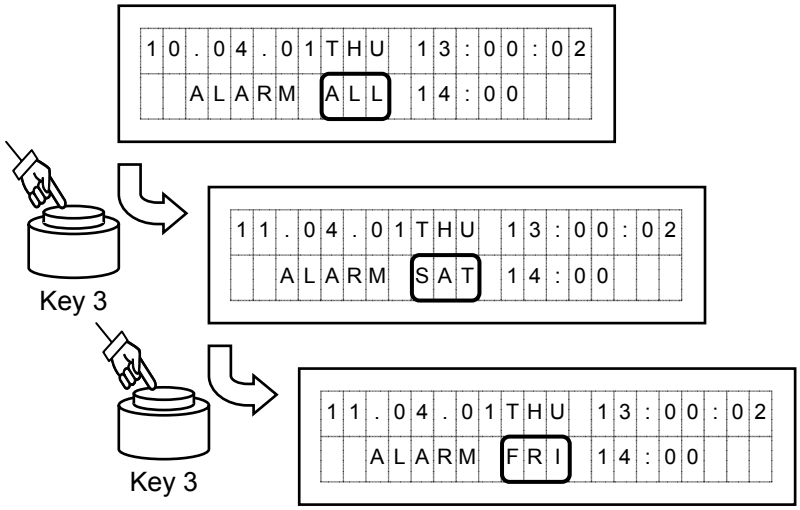
<3> Key 3 operation

Key 3 is used to decrement the value (down key).

<Specifying the alarm day-of-week setting>

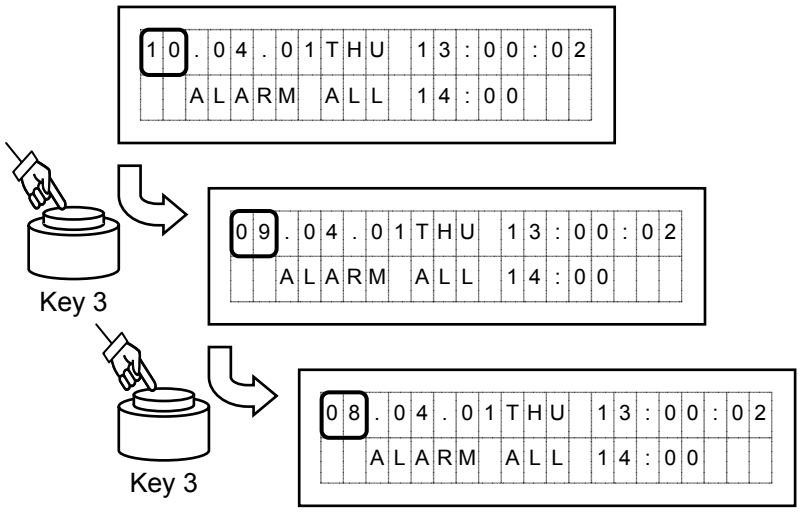
The value being set is switched as follows:

ALL[Note] → SAT → FRI → THU → WED → TUE → MON → SUN → ALL[Note] →…



<Specifying other settings>

The value being set is decremented.



Note "ALL" indicates "everyday".

Caution **The day-of-week must be set manually in accordance with actual the year, month, and day.**

Remark The settings being specified are ringed in black.

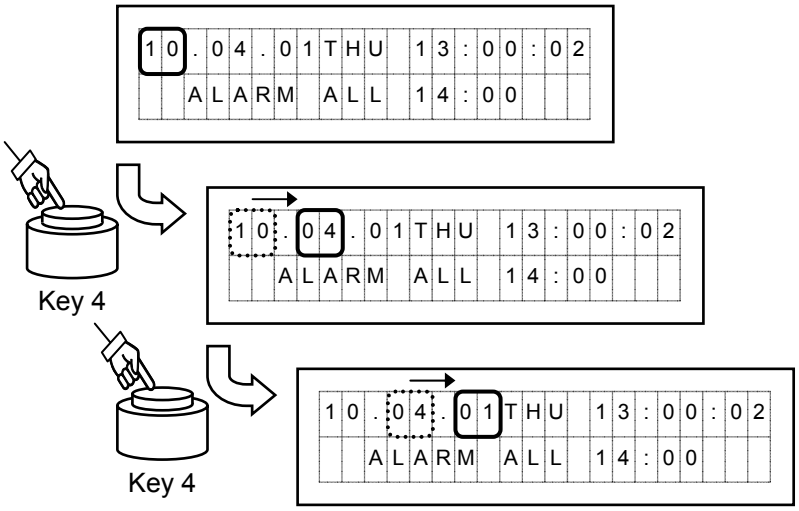
<4> Key 4 operation

Key 4 is used to switch items.

<Specifying the day and time setting>

The items being set are switched as follows:

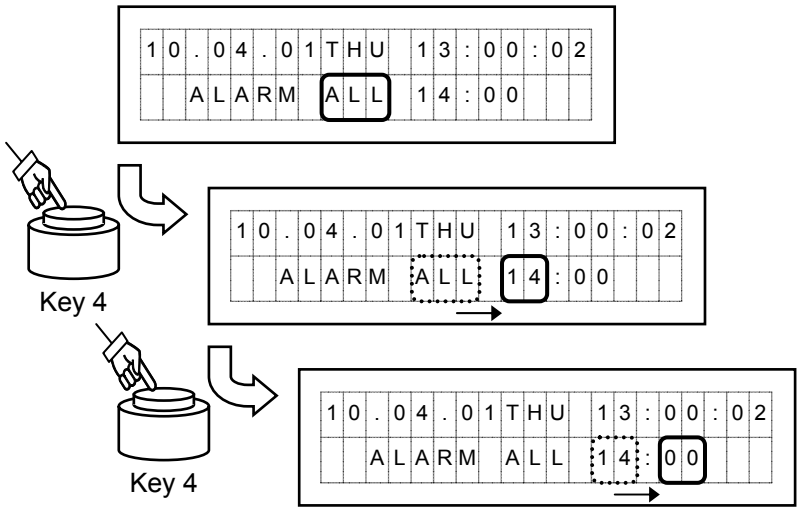
Year → month → day → day-of-week → hour → minute → second → year →…



<Specifying the alarm setting>

The items being set are switched as follows:

Day-of-week → hour → minute → day-of-week →…



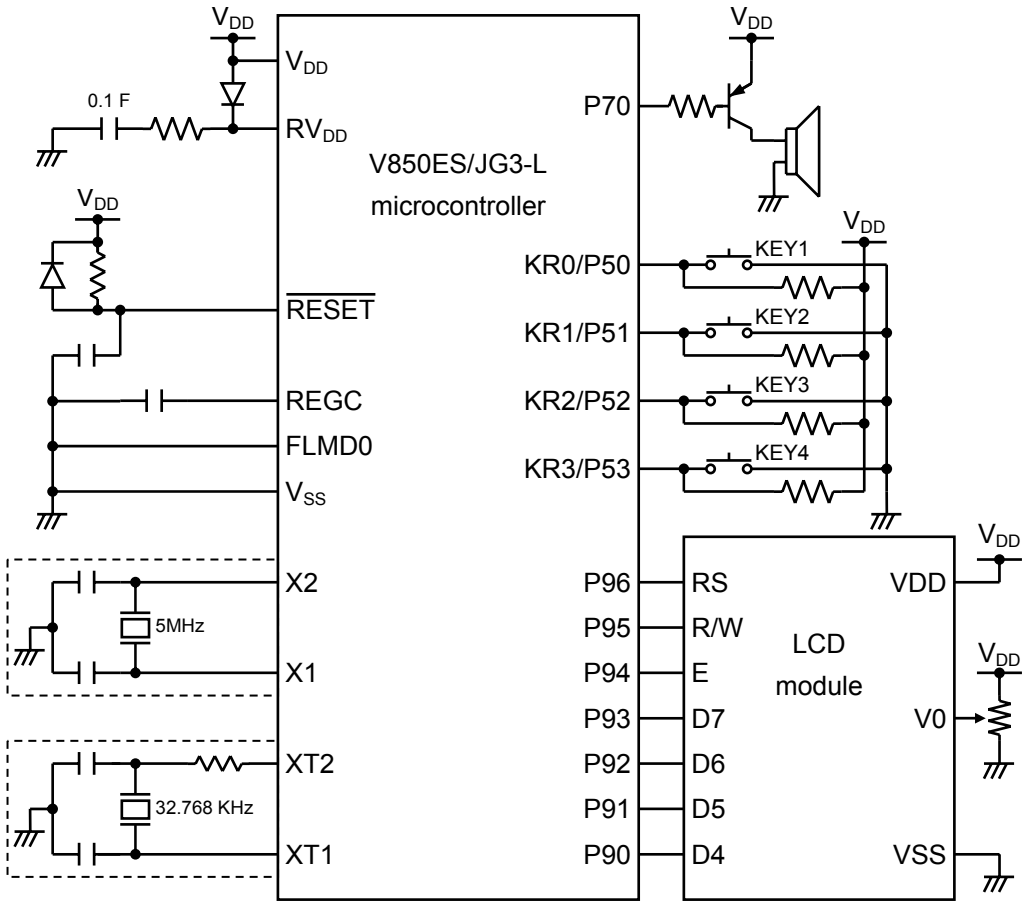
Remark The settings being specified are ringed in black.

2. CIRCUIT DIAGRAM

This chapter shows the circuit diagram and describes the devices other than the microcontroller used in this sample program.

2.1 Circuit Diagram

The circuit diagram is shown below.



Cautions 1. In normal operation mode, use V_{DD} in the range of 2.8 V < V_{DD} ≤ 3.6 V.

2. Directly connect the EVDD, AVREF0, and AVREF1 pins to V_{DD}.

3. Directly connect the EVSS and AVSS pins to GND.

4. Connect the REGC to GND via a capacitor (recommended value: 4.7 μF).

5. Connect the FLMD0 pin to GND in normal operation mode.

6. Leave all the unused ports open so that they are handled as output ports.

7. When using the main clock oscillator and/or subclock oscillator, wire as follows in the area enclosed by the broken lines in the above figure to avoid an adverse effect from wiring capacitance.

- **Keep the wiring length as short as possible.**
- **Do not cross the wiring with the other signal lines.**
- **Do not route the wiring near a signal line through which a high fluctuating current flows.**
- **Always make the ground point of the oscillator capacitor the same potential as V_{SS}.**
- **Do not ground the capacitor to a ground pattern through which a high current flows.**
- **Do not fetch signals from the oscillator.**

8. Connect an electric double-layer capacitor (0.1 F) to the RV_{DD} pin so that supply of power to the RTC backup power supply (RV_{DD}) can continue for a period if the supply voltage (V_{DD}) is stopped.

Remark For the resonator selection and oscillator constant, customers are requested to either evaluate the oscillation themselves or apply to the resonator manufacturer for evaluation.

2.2 Devices Used Other Than Microcontroller

The following shows devices other than the microcontroller used in this sample program.

(1) LCD module

An LCD module (character display, 20 characters x 2 rows) is used as the display device.

(2) Push keys (KEY1, KEY2, KEY3, KEY4)

Push keys are used for key operation.

(3) Electronic buzzer

An electronic buzzer is used to output the alarm sound.

3. SOFTWARE

This chapter describes the organization of the compressed files that are downloaded, the on-chip microcontroller peripherals used, and the initial settings and operation of the sample program, which are also shown in a flow chart.

3.1 File Organization

The compressed files that are downloaded are organized as follows:

File Name (Tree Structure)	Description	Compressed (*.zip) File	
			
conf — crtE.s	Startup routine file[Note 1]	–	√
— AppNote_RTC_BUM.dir	Link directive file[Note 2]	√	√
— AppNote_RTC_BUM.prj	Project file for integrated development environment PM+	–	√
— AppNote_RTC_BUM.prw	Workspace file for integrated development environment PM+	–	√
src — main.c	C source file containing code for microcontroller's hardware initialization processing and main processing.	√	√
— lcd.c	C source file containing code for LCD module (NHD-0220FZ-FSW-GBW-P-3V3) control processing.	√	√
— minicube2.s	Source file used to reserve area for MINICUBE2	√	√
— opt_b.s	Source file used to specify option byte settings	√	√

Notes 1. This is the startup file copied if Copy Sample (C) is selected for Specify Startup File when creating a new workspace. (If the default installation path is selected, the file that is copied will be C:\Program Files\NEC Electronics Tools\CA850*version-used*\lib850\r32\crtE.s.)

2. If Copy Sample (C) is selected for Specify Link Directive File when creating a new workspace and the Memory Usage: Internal Memory Only (I) option is checked, MINICUBE2 segments will be automatically added to this link directive file. (If the default installation path is selected, the reference file will be C:\Program Files\NEC Electronics Tools\PM+*version-used*\bin\w_data\V850_i.dat.)

Remark  : Includes only source files.

 : Includes source files used by PM+ integrated development environment.

3.2 On-Chip Peripherals Used

This sample program uses the following on-chip microcontroller peripherals:

<Peripheral I/O circuits>

- Real-time counter: Used to trigger fixed-cycle interrupts, alarm interrupts, and interval interrupts. Also operates in RTC backup mode.
- Low-voltage detector: Used to detect a low voltage and to check whether V_{DD} is 2.7 V or higher.

<Pins>

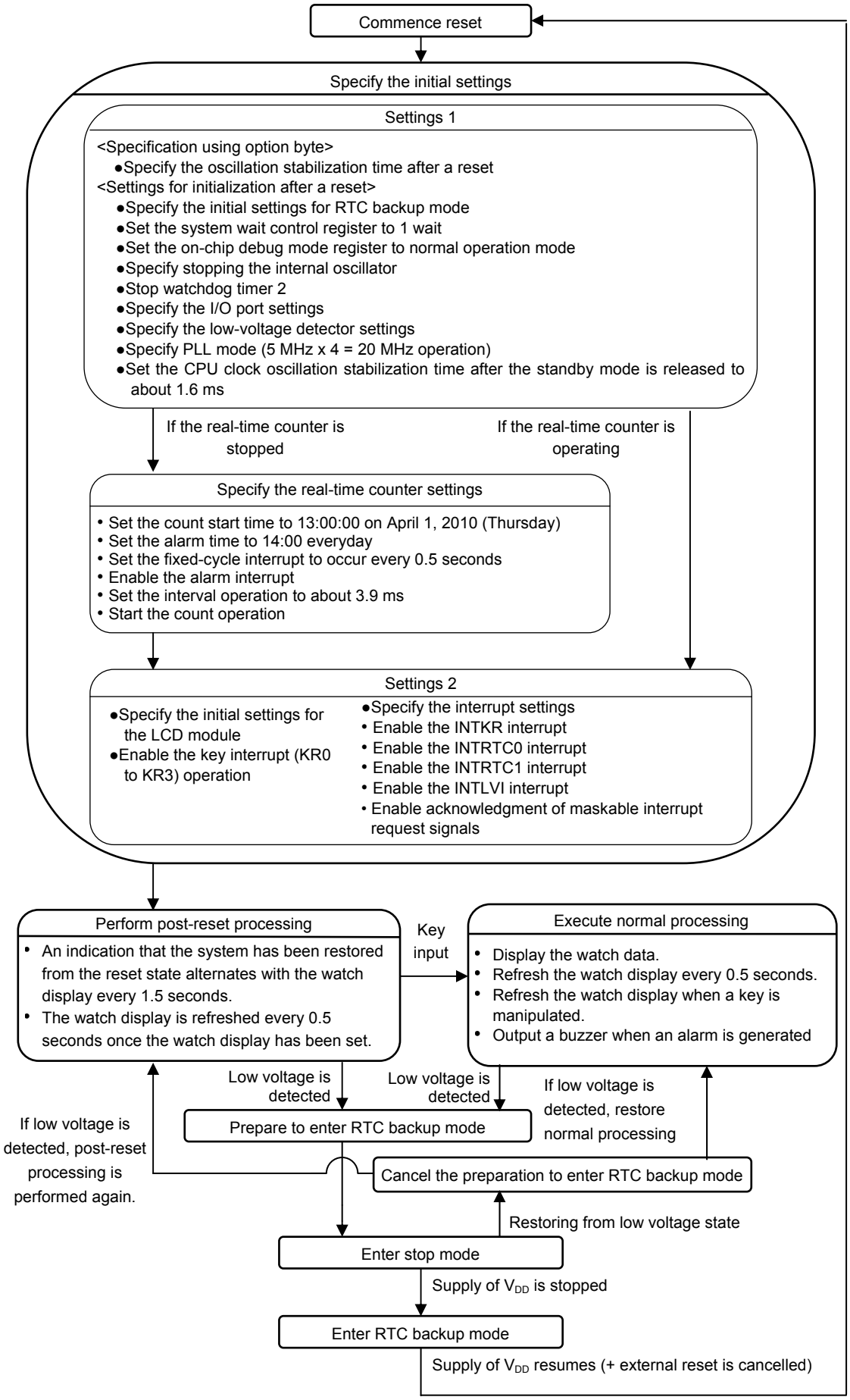
- KR0 to KR3: Used for key input.
- P70: Used for buzzer control.
- P90 to P96: Used to control the LCD module.

3.3 Initial Settings and Operation Overview

This sample program is used to specify initial settings such as selecting the clock frequency, stopping watchdog timer 2, the I/O port and external interrupt pin settings, the low-voltage detector settings, the real-time counter settings, the RTC backup mode settings, and the LCD module settings. Note that if the real-time counter is already operating when specification of the initial settings starts, specifying the real-time counter settings is not required.

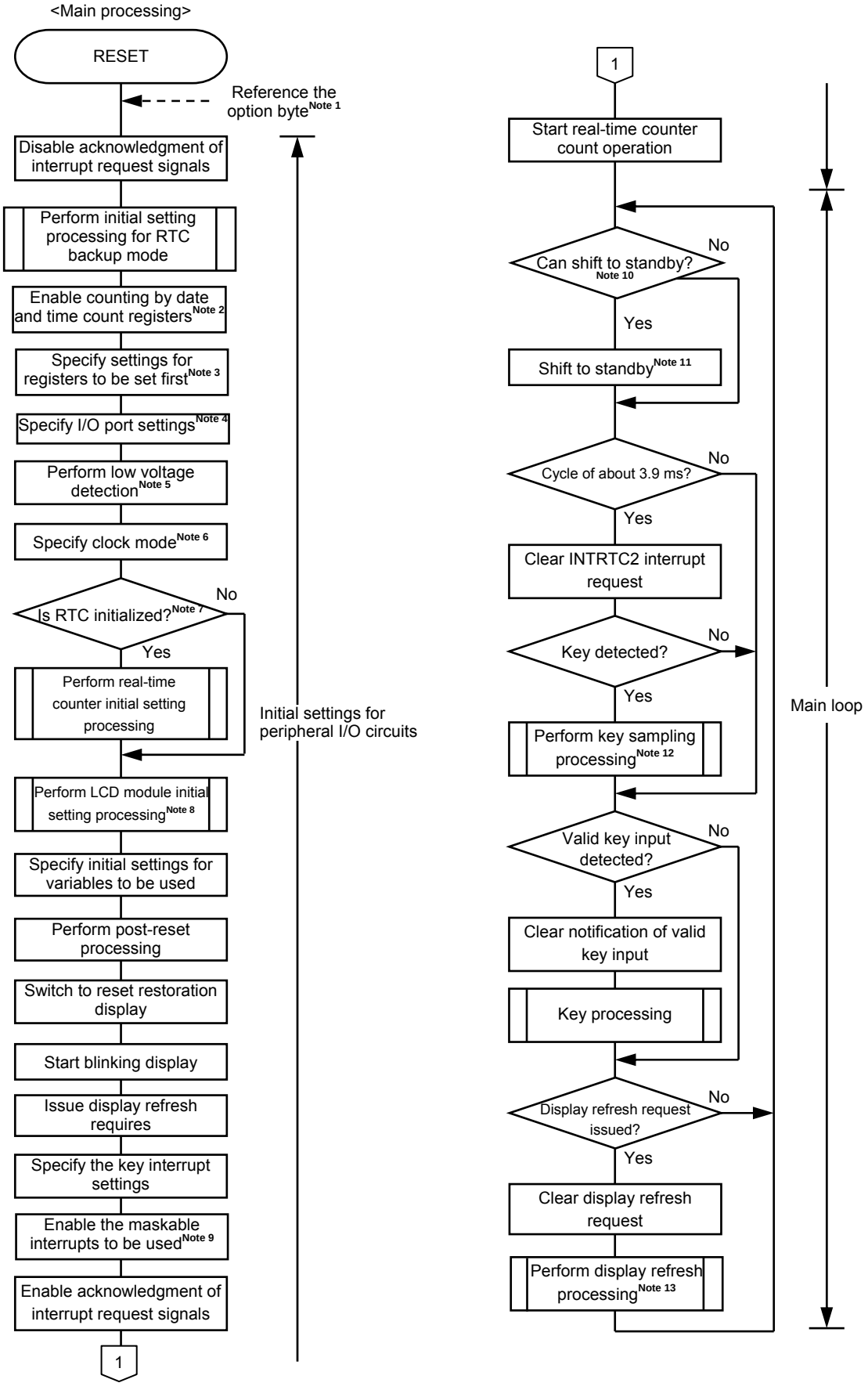
When specification of the initial settings is complete, an interrupt is generated to start program execution. The processing that is executed differs depending on the generated interrupt. The interrupts generated to start program execution include the 0.5-second-cycle INTRTC0 interrupt, the alarm-triggered INTRTC1 interrupt, the 3.9-millisecond interval interrupt, and the key-input-triggered INTKR interrupt. The processing executed by the generation of these interrupts includes key manipulation, LCD module control, and buzzer control. Another interrupt, the INTLVI interrupt, which is triggered by the detection of a low voltage or by the restoration of the voltage to a specific level, is also used to prepare the system to enter the RTC backup mode or to cancel the preparation to enter the RTC backup mode.

For details about the processing flow, see the status transition diagram and flow charts below.


Commence reset
Specify the initial settings
Settings 1
<Specification using option byte>
●Specify the oscillation stabilization time after a reset
<Settings for initialization after a reset>
●Specify the initial settings for RTC backup mode
●Set the system wait control register to 1 wait
●Set the on-chip debug mode register to normal operation mode
●Specify stopping the internal oscillator
●Stop watchdog timer 2
●Specify the I/O port settings
●Specify the low-voltage detector settings
●Specify PLL mode (5 MHz x 4 = 20 MHz operation)
●Set the CPU clock oscillation stabilization time after the standby mode is released to about 1.6 ms
If the real-time counter is stopped
If the real-time counter is operating
Specify the real-time counter settings
• Set the count start time to 13:00:00 on April 1, 2010 (Thursday)
• Set the alarm time to 14:00 everyday
• Set the fixed-cycle interrupt to occur every 0.5 seconds
• Enable the alarm interrupt
• Set the interval operation to about 3.9 ms
• Start the count operation
Settings 2
●Specify the initial settings for the LCD module
●Enable the key interrupt (KR0 to KR3) operation
●Specify the interrupt settings
• Enable the INTKR interrupt
• Enable the INTRTC0 interrupt
• Enable the INTRTC1 interrupt
• Enable the INTLVI interrupt
• Enable acknowledgment of maskable interrupt request signals
Perform post-reset processing
• An indication that the system has been restored from the reset state alternates with the watch display every 1.5 seconds.
• The watch display is refreshed every 0.5 seconds once the watch display has been set.
Key input
Execute normal processing
• Display the watch data.
• Refresh the watch display every 0.5 seconds.
• Refresh the watch display when a key is manipulated.
• Output a buzzer when an alarm is generated
Low voltage is detected
Low voltage is detected
If low voltage is detected, restore normal processing
If low voltage is detected, post-reset processing is performed again.
Prepare to enter RTC backup mode
Cancel the preparation to enter RTC backup mode
Restoring from low voltage state
Enter stop mode
Supply of V_{DD} is stopped
Enter RTC backup mode
Supply of V_{DD} resumes (+ external reset is cancelled)

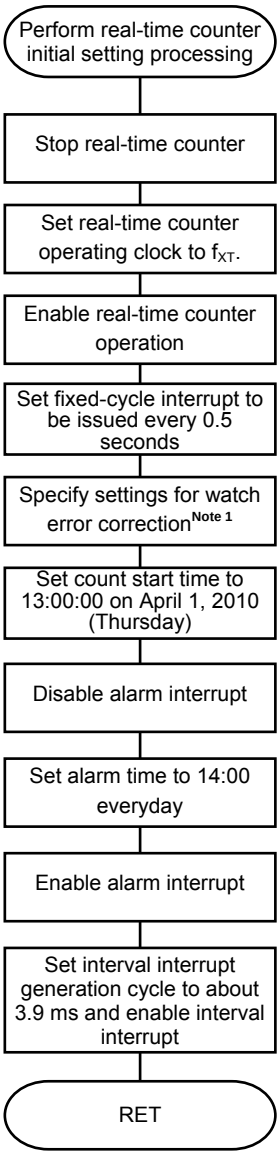
3.4 Flow Charts

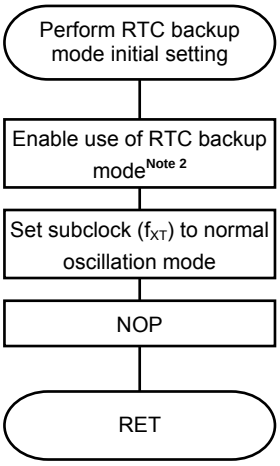
The processing performed by this sample program is illustrated in the following flow charts.



Notes

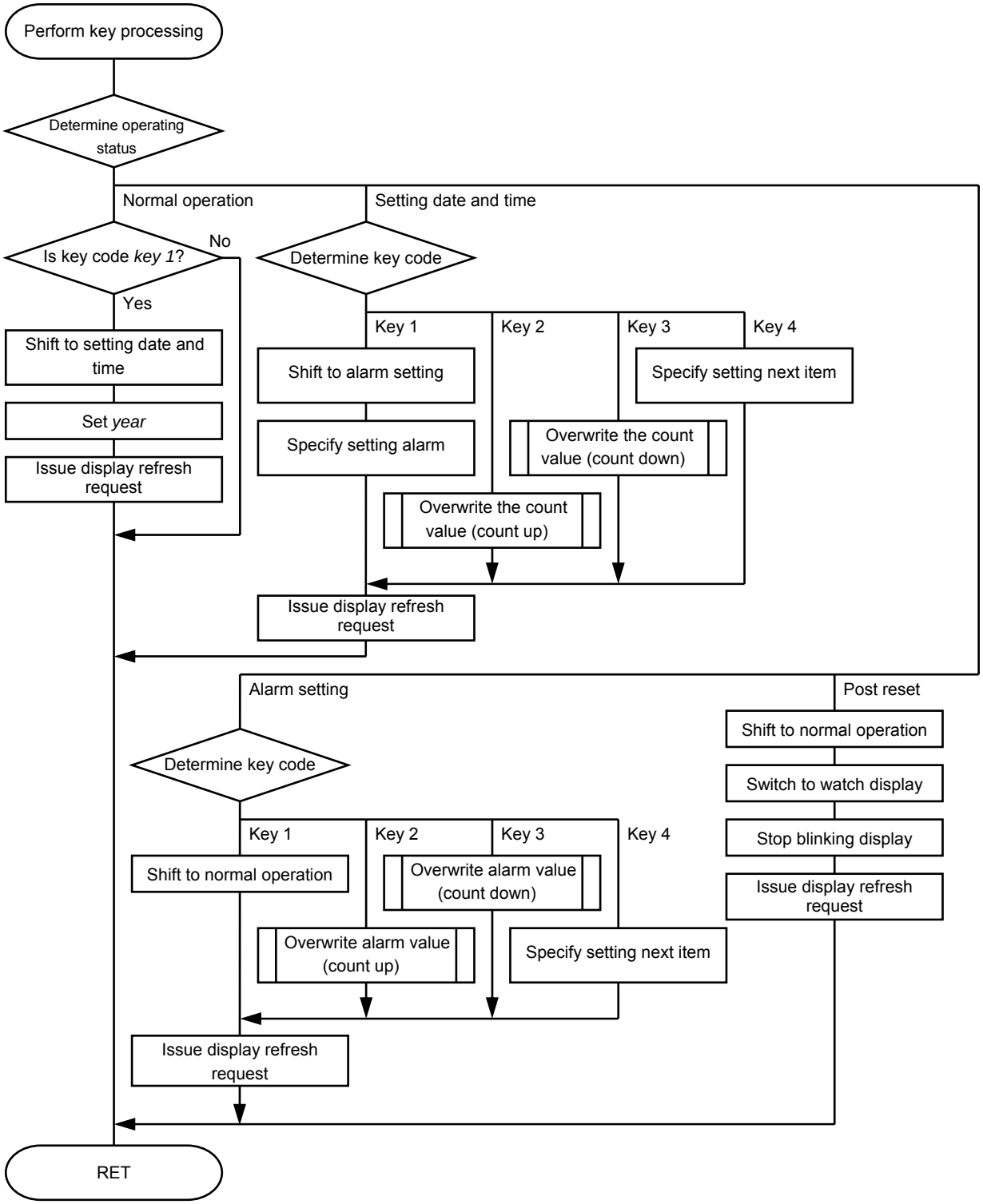
1. Referencing the option byte is performed automatically by the microcontroller after the reset is cancelled. In this sample program, the oscillation stabilization time after reset is cancelled is set to 6.554 ms.
2. If the real-time counter is already operating after reset is cancelled, the date and time registers might have stopped counting due to the processing that was executed before the reset signal was input. The count operation of the date and time registers is therefore enabled here.
3. Specify the settings of the VSWC, OCDM, RCM, and WDTM2 registers. These registers should be set as follows:
   - Set the system wait control register (VSWC) to 1 wait.
   - Set the on-chip debug mode register (OCDM) to normal operation mode.
   - Stop internal oscillation (by using the RCM register).
   - Stop operation of watchdog timer 2 (by using the WDTM2 register).
4. Set P50 to P53 (KR0 to KR3) to key input, P70 to buzzer control, and P90 to P96 to LCD module control. Set all unused ports to low level.
5. Enable operation of the low voltage detector and wait until the power supply voltage reaches 2.8 V.
6. Set the clock mode to PLL mode (5 MHz × 4 = 20 MHz). Also set the CPU clock oscillation stabilization time after the system exits standby to about 1.6 ms.
7. Determine whether the real-time counter is in the initial state. If it is already operating, the initial settings are omitted.
8. This processing specifies the initial settings for the LCD module used.
9. Enable the INTRTC0, INTRTC1, INTKR, and INTLVI interrupts.
10. If no key was detected or no display refresh processing is to be performed, it is possible to shift to standby.
11. The system exits standby if the INTRTC0, INTRTC1, INTKR, or INTLVI interrupt is generated.
12. In this processing, the key input pins are sampled (three times, at an interval of about 3.9 ms). The sampling processing returns the key code (key 1, key 2, key 3, or key 4) and a notification that a valid key input has been detected. If a key is off, the key detection notification for that key is cleared.
13. In this processing, new display data is applied to the LCD module.

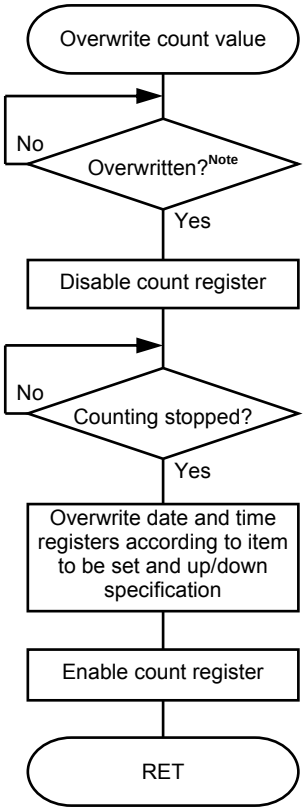


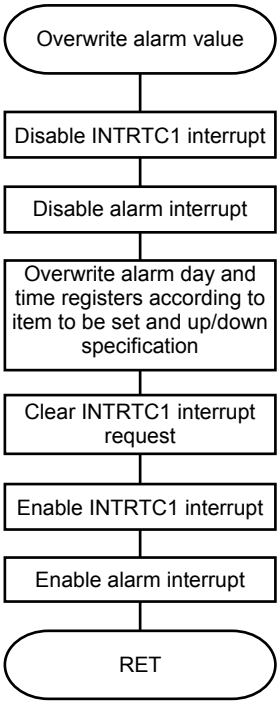


Notes 1. The watch error correction feature is disabled in this sample program.

2. If the system enters the state in which it is preparing to enter RTC backup mode after reset is canceled, it exits this state here.

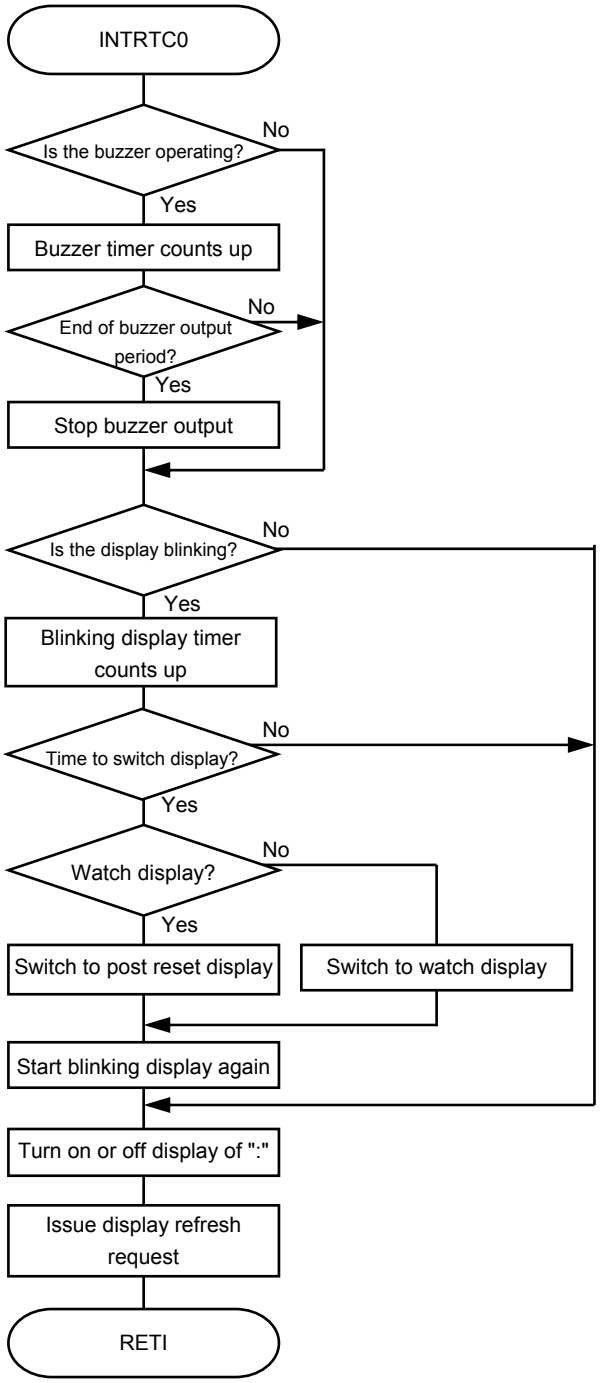

Perform key processing
Determine operating status
Normal operation
Setting date and time
Is key code key 1?
No
Yes
Determine key code
Key 1
Key 2
Key 3
Key 4
Shift to setting date and time
Set year
Issue display refresh request
Shift to alarm setting
Specify setting alarm
Overwrite the count value (count down)
Specify setting next item
Overwrite the count value (count up)
Issue display refresh request
Alarm setting
Post reset
Shift to normal operation
Switch to watch display
Stop blinking display
Issue display refresh request
Determine key code
Key 1
Key 2
Key 3
Key 4
Shift to normal operation
Overwrite alarm value (count down)
Overwrite alarm value (count up)
Specify setting next item
Issue display refresh request
RET



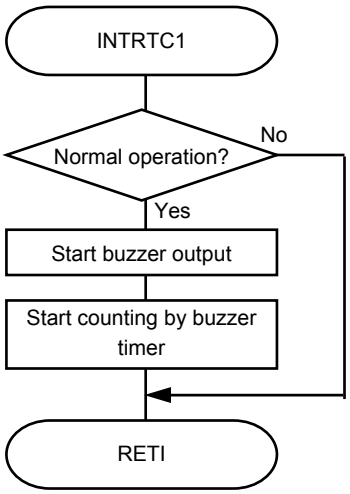


Note Check that the previous processing to write the count register is complete.

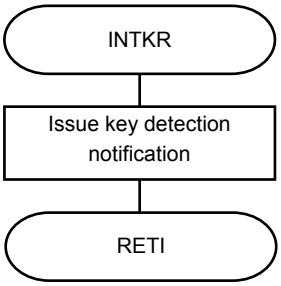
<Fixed-cycle interrupt processing>

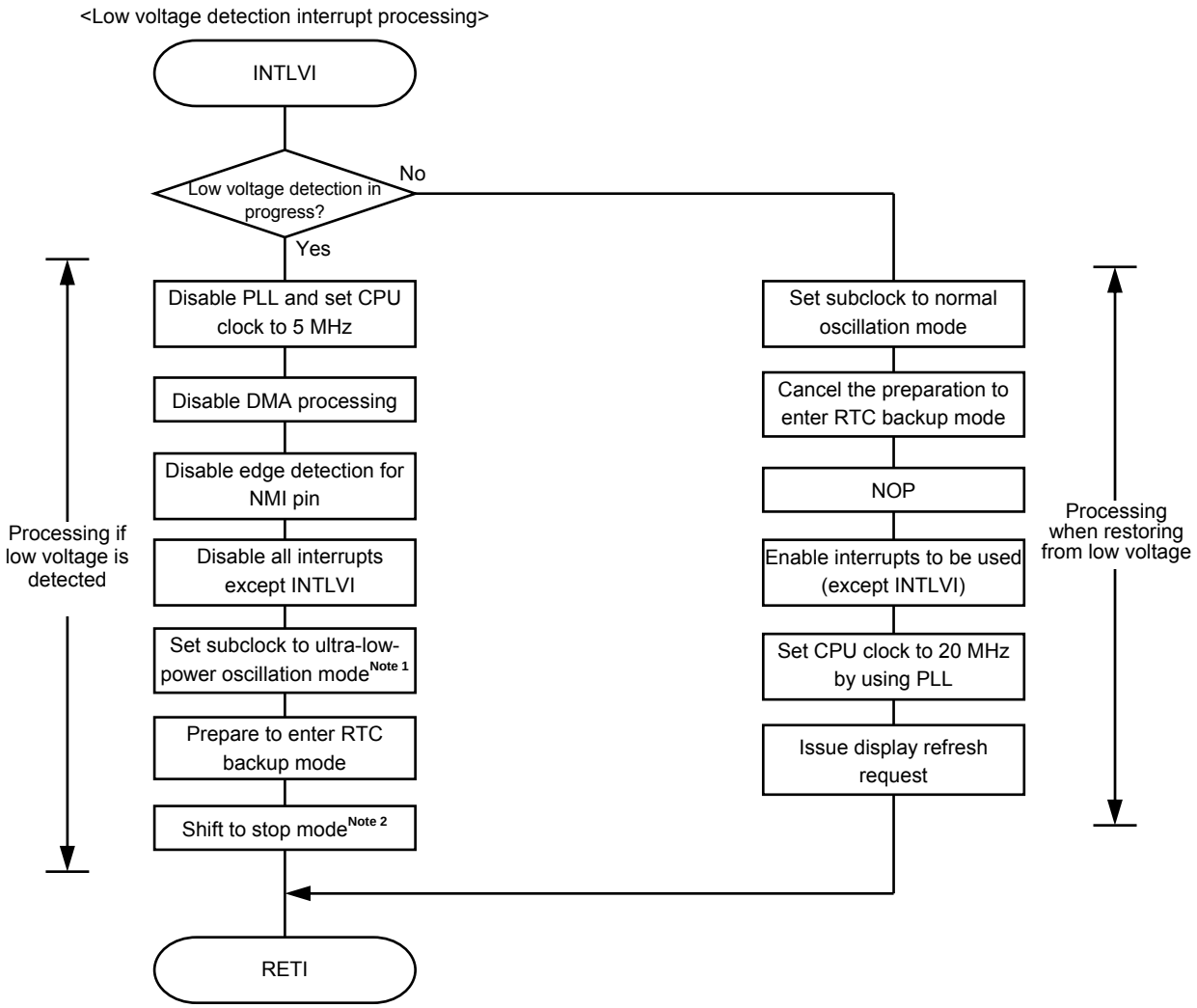


<Alarm interrupt processing>



<Key interrupt processing>





Notes 1. This setting must be specified to use the subclock's ultra-low-power feature, but it is not necessarily required to enter RTC backup mode.

2. If supply of the power supply voltage (V_{DD}) stops in stop mode, the system shifts to RTC backup mode. On the other hand, if the voltage rises again from the low voltage state to the normal operating voltage level, the processing in <1> to <4> below is executed.

<1> The INTLVI interrupt is generated and the system exits stop mode. Note that at this time, acknowledgment of the INTLVI interrupt is held pending.

<2> The RETI instruction is issued to restore the system from low voltage interrupt processing.

<3> After the system is restored, the pending INTLVI interrupt is acknowledged and low voltage interrupt processing is executed again.

<4> The processing branches to the processing when restoring from low voltage shown in the figure above, and the system cancels the preparation to enter RTC backup mode.

4. SPECIFYING SETTINGS

This chapter describes the settings to be specified for the real-time counter, RTC backup mode, and watch error correction feature.

For details about the initial settings of the low voltage detector, see the application note entitled V850ES/Jx3-L Sample Program (Low-Voltage Detector (LVI)) Reset Generation When Low Voltage Detected.

For details about other initial settings, see the application note entitled V850ES/Jx3-L Sample Program (Initial Settings)) LED Lighting Switch Control.

For details about registers, see the V850ES/Jx3-L user's manual.

For the extended C language code, see the following user's manual:

- CA850 C Compiler Package for C Language

4.1 Setting the Real-Time Counter

In this sample program, the real-time counter is controlled by using the following registers:

- Real-time counter control register 0 (RC1CC0)
- Real-time counter control register 1 (RC1CC1)
- Real-time counter control register 2 (RC1CC2)
- Real-time counter control register 3 (RC1CC3)
- Second count register (RC1SEC)
- Minute count register (RC1MIN)
- Hour count register (RC1HOUR)
- Day count register (RC1DAY)
- Day-of-week count register (RC1WEEK)
- Month count register (RC1MONTH)
- Year count register (RC1YEAR)
- Watch error correction register (RC1SUBU)
- Alarm minute setting register (RC1ALM)
- Alarm hour setting register (RC1ALH)
- Alarm day-of-week setting register (RC1ALW)

(1) Real-time counter control register 0 (RC1CC0)

The RC1CC0 register selects the real-time counter input clock.

This register can be read or written in 8-bit or 1-bit units.

Figure 4-1-1. Format of Real-Time Counter Control Register 0 (RC1CC0)

Real-time counter control register 0 (RC1CC0)

Address: FFFFFADDH

7	6	5	4	3	2	1	0
RC1PWR	RC1CKS[Note]	0	0	0	0	0	0

RC1PWR	Real-time counter operation control
0	Stop real-time counter operation.
1	Enable real-time counter operation.

RC1CKS[Note]	Operation clock selection
0	Select f_{XT} as operating clock.
1	Select f_{BRG} as operating clock.

Note If f_{XT} is selected as the operating clock, be sure to set the RC1CKS bit to 0 while the system is preparing to enter RTC backup mode (while RTCBUMCTL0.RBMSET is set to 1).

Cautions 1. Follow the description in *11.4.8 Initializing real-time counter* in the user's manual when stopping the real-time counter while it is operating (by changing the RC1PWR bit from 1 to 0).

2. The RC1CKS bit can be rewritten only when the real-time counter is stopped (RC1PWR = 0). Furthermore, rewriting the RC1CKS bit at the same time as changing the RC1PWR bit from 0 to 1 is prohibited.

3. Be sure to set bits 0 to 5 to "0".

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. The parts written in red in the above figure are the values set by using this sample program.

(2) Real-time counter control register 1 (RC1CC1)

The RC1CC1 register is an 8-bit register that starts or stops the real-time counter, controls the RTCCL and RTC1HZ pins, selects the 12-hour or 24-hour system, and sets the fixed-cycle interrupt.

This register can be read or written in 8-bit or 1-bit units.

Figure 4-1-2. Format of Real-Time Counter Control Register 1 (RC1CC1)

Real-time counter control register 1 (RC1CC1)

Address: FFFFFADEH

7	6	5	4	3	2	1	0
RTCE	0	CLOE1	CLOE0	AMPM	CT2	CT1	CT0

RTCE	Control of operation of each counter
0	Stop counter operation.
1	Enable counter operation.

CLOE1	RTC1HZ pin output control
0	Disable RTC1HZ pin output (1 Hz)
1	Enable RTC1HZ pin output (1 Hz)

CLOE0	RTCCL pin output control
0	Disable RTCCL pin output (32.768 kHz)
1	Enable RTCCL pin output (32.768 kHz)

AMPM	12-hour system/24-hour system selection
0	12-hour system (a.m. and p.m. are displayed.)
1	24-hour system

CT2	CT1	CT0	Fixed-cycle interrupt (INTRTC0) selection
0	0	0	Do not use fixed-cycle interrupts
0	**0**	**1**	Once every 0.5 seconds (synchronous with second count-up)
0	1	0	Once a second (simultaneous with second count-up)
0	1	1	Once a minute (every minute at 00 seconds)
1	0	0	Once a hour (every hour at 00 minutes 00 seconds)
1	0	0	Once a day (every day at 00 hours 00 minutes 00 seconds)
1	1	x	Once a month (one day every month at 00 hours 00 minutes 00 seconds a.m.)

Cautions 1. Writing 0 to the RTCE bit while the RTCE bit is 1 is prohibited. Clear the RTCE bit by clearing the RC1PWR bit according to *11.4.8 Initializing real-time counter* in the user's manual.

2. The RTC1HZ output operates as follows when the CLOE1 bit setting is changed:

- **When changed from 0 to 1: RTC1HZ outputs a 1 Hz pulse within two clocks (2 × 32.768 kHz).**
- **When changed from 1 to 0: RTC1HZ output is stopped (fixed to low level) within two clocks (2 × 32.768 kHz).**

3. Specify the RC1HOUR register setting again if the AMPM bit is rewritten.

4. Be sure to set bit 6 to "0".

Remarks 1. In RTC backup mode, mask the fixed-cycle interrupt (by setting the RTC0MK bit to 1) and stop output from the RTCCL and RTC1HZ pins.

2. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

3. The parts written in red in the above figure are the values set by using this sample program.

(3) Real-time counter control register 2 (RC1CC2)

The RC1CC2 register is an 8-bit register that controls the alarm interrupt and counter waits.

This register can be read or written in 8-bit or 1-bit units.

Figure 4-1-3. Format of Real-Time Counter Control Register 2 (RC1CC2)

Real-time counter control register 2 (RC1CC2)

Address: FFFFFADFH

7	6	5	4	3	2	1	0
WALE	0	0	0	0	0	RWST	RWAIT

WALE	Alarm interrupt (INTRTC1) operation control
0	Do not generate interrupt upon alarm match.
1	Generate interrupt upon alarm match.

RWST	Real-time counter wait state
0	Counter operating
1	Counting up of date and time counters stopped (Reading and writing of counter values enabled)

RWAIT	Real-time counter wait control
0	Sets counter operation.
1	Stops count operation of date and time counters. (Counter value read/write mode)

Cautions 1. Confirm that the RWST bit is set to 1 when reading or writing each counter value.

2. The RWST bit does not become 0 while each counter is being written, even if the RWAIT bit is set to 0. It becomes 0 when writing to each counter is completed.

3. Be sure to set bits 2 to 6 to "0".

Remarks 1. In RTC backup mode, mask the alarm interrupt (by setting the RTC1MK bit to 1).

2. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

3. The parts written in red in the above figure are the values set by using this sample program.

(4) Real-time counter control register 3 (RC1CC3)

The RC1CC3 register is an 8-bit register that controls the interval interrupt and RTCDIV pin.

This register can be read or written in 8-bit or 1-bit units.

Figure 4-1-4. Format of Real-time Counter Control Register 3 (RC1CC3)

Real-time counter control register 3 (RC1CC3)

Address: FFFFFAE0H

7	6	5	4	3	2	1	0
RINTE	CLOE2	CKDIV	0	0	ICT2	ICT1	ICT0

RINTE	Interval interrupt (INTRTC2) control
0	Do not generate interval interrupt.
1	Generate interval interrupt.

CLOE2	RTCDIV pin output control
0	Disable RTCDIV pin output.
1	Enable RTCDIV pin output.

CKDIV	RTCDIV pin output frequency selection
0	Output 512 Hz (1.95 ms) from RTCDIV pin.
1	Output 16.384 kHz (0.061 ms) from RTCDIV pin.

ICT2	ICT1	ICT0	Interval interrupt (INTRTC2) selection
0	0	0	$2^6/f_{XT}$ (1.953125 ms)
0	**0**	**1**	$2^7/f_{XT}$ (3.90625 ms)
0	1	0	$2^8/f_{XT}$ (7.8125 ms)
0	1	1	$2^9/f_{XT}$ (15.625 ms)
1	0	0	$2^{10}/f_{XT}$ (31.25 ms)
1	0	0	$2^{11}/f_{XT}$ (62.5 ms)
1	1	x	$2^{12}/f_{XT}$ (125 ms)

Cautions 1. The RTCDIV output operates as follows when the CLOE2 bit setting is changed:

- **When changed from 0 to 1: A pulse set by the CKDIV bit is output within two clocks (2 × 32.768 kHz).**
- **When changed from 1 to 0: Output of the RTCDIV output is stopped within two clocks (fixed to low level, 2 × 32.768 kHz).**

2. Be sure to set bits 3 and 4 to "0".

Remarks 1. In RTC backup mode, disable the interval interrupt and RTCDIV pin output.

2. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

3. The parts written in red in the above figure are the values set by using this sample program.

(5) Second count register (RC1SEC)

The RC1SEC register is an 8-bit register that takes a value of 0 to 59 (decimal) and indicates the count value of the seconds. It counts up when the sub-counter overflows.

When data is written to this register, it is written to a buffer and then to the counter up to 2 clocks (2 × 32.768 kHz) later. Set a decimal value of 00 to 59 to this register in BCD code. If a value outside this range is set, the register value returns to the normal value after one period.

This register can be read or written in 8-bit units.

Note The sub-count register (RC1SUBC) is a 16-bit register that holds a value of between 0000H and 7FFFH and counts up every second based on a 32.768 kHz clock. For details of this register, see the V850ES/Jx3-L user's manual.

Caution Setting the RC1SEC register to values other than 00 to 59 is prohibited.

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 00H in this sample program.

Figure 4-1-5. Format of Second Count Register (RC1SEC)

Second count register (RC1SEC)

Address: FFFFFAD2H

7	6	5	4	3	2	1	0
0							

(6) Minute count register (RC1MIN)

The RC1MIN register is an 8-bit register that takes a value of 0 to 59 (decimal) and indicates the count value of the minutes. It counts up when the second counter overflows.

When data is written to this register, it is written to a buffer and then to the counter up to 2 clocks (2 × 32.768 kHz) later. Set a decimal value of 00 to 59 to this register in BCD code.

This register can be read or written 8-bit units.

Caution Setting a value other than 00 to 59 to the RC1MIN register is prohibited.

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 00H in this sample program.

Figure 4-1-6. Format of Minute Count Register (RC1MIN)

Minute count register (RC1MIN)

Address: FFFFFAD3H

7	6	5	4	3	2	1	0
0							

(7) Hour count register (RC1HOUR)

The RC1HOUR register is an 8-bit register that takes a value of 0 to 23 or 1 to 12 (decimal) and indicates the count value of the hour. It counts up when the minute counter overflows.

When data is written to this register, it is written to a buffer and then to the counter up to 2 clocks (2 × 32.768 kHz) later. Set a decimal value of 00 to 23, 01 to 12, or 21 to 32 to this register in BCD code. If a value outside this range is set, the register value returns to the normal value after 1 period.

This register can be read or written 8-bit units.

Cautions 1. Bit 5 of the RC1HOUR register indicates a.m. (0) or p.m. (1) if AMPM = 0 (if the 12-hour system is selected).

2. Setting a value other than 01 to 12, 21 to 32 (AMPM bit= 0), or 00 to 23 (AMPM bit = 1) to the RC1HOUR register is prohibited.

Remarks 1. This register is reset to 12H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 13H (AMPM bit = 1) in this sample program.

Figure 4-1-7. Format of Hour Count Register (RC1HOUR)

Hour count register (RC1HOUR)

Address: FFFFFAD4H

7	6	5	4	3	2	1	0
0	0						

Table 4-1 shows the relationship between the AMPM bit value, the RC1HOUR register value, and the time.

Table 4-1. Time Digit Display

12-Hour Display (AMPM Bit = 0)		24-Hour Display (AMPM Bit = 1)	
Time	RC1HOUR Register Value	Time	RC1HOUR Register Value
0:00 a.m.	12H	0:00	00H
1:00 a.m.	01H	1:00	01H
2:00 a.m.	02H	2:00	02H
3:00 a.m.	03H	3:00	03H
4:00 a.m.	04H	4:00	04H
5:00 a.m.	05H	5:00	05H
6:00 a.m.	06H	6:00	06H
7:00 a.m.	07H	7:00	07H
8:00 a.m.	08H	8:00	08H
9:00 a.m.	09H	9:00	09H
10:00 a.m.	10H	10:00	10H
11:00 a.m.	11H	11:00	11H
0:00 p.m.	32H	12:00	12H
1:00 p.m.	21H	13:00	13H
2:00 p.m.	22H	14:00	14H
3 :00 p.m.	23H	15:00	15H
4:00 p.m.	24H	16:00	16H
5:00 p.m.	25H	17:00	17H
6:00 p.m.	26H	18:00	18H
7:00 p.m.	27H	19:00	19H
8:00 p.m.	28H	20:00	20H
9:00 p.m.	29H	21:00	21H
10:00 p.m.	30H	22:00	22H
11:00 p.m.	31H	23:00	23H

The RC1HOUR register value is displayed in 12-hour format if the AMPM bit is 0 and in 24-hour format if the AMPM bit is 1.

In 12-hour format, a.m. or p.m. is indicated by the fifth bit of RC1HOUR, with 0 indicating before noon (a.m.) and 1 indicating noon or afternoon (p.m.).

(8) Day count register (RC1DAY)

The RC1DAY register is an 8-bit register that takes a value of 1 to 31 (decimal) and indicates the count value of the day. This register can be read or written in 8-bit units.

This register counts up when the hour counter overflows.

This counter counts as follows:

- 01 to 31 (January, March, May, July, August, October, December)
- 01 to 30 (April, June, September, November)
- 01 to 29 (February in a leap year)
- 01 to 28 (February in a normal year)

When data is written to this register, it is written to a buffer and then to the counter up to 2 clocks (32.768 kHz) later.

Set a decimal value of 00 to 31 to this register in BCD code.

Caution Setting a value other than 01 to 31 to the RC1DAY register is prohibited. Setting a value outside the above-mentioned count range, such as "February 30" is also prohibited.

Remarks 1. This register is reset to 01H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 01H in this sample program.

Figure 4-1-8. Format of Day Count Register (RC1DAY)

Day count register (RC1DAY)

Address: FFFFFAD6H

7	6	5	4	3	2	1	0
0	0						

(9) Day-of-week count register (RC1WEEK)

The RC1WEEK register is an 8-bit register that takes a value of 0 to 6 (decimal) and indicates the count value of the day of the week.

It counts up in synchronization with the day counter.

When data is written to this register, it is written to a buffer and then to the counter up to 2 clocks (2 × 32.768 kHz) later. Set a decimal value of 00 to 06 to this register in BCD code. If a value outside this range is set, the register value returns to the normal value after 1 period.

This register can be read or written in 8-bit units.

Figure 4-1-9. Format of Day-of-Week Count Register (WEEK)

Day-of-week count register (RC1WEEK)

Address: FFFFFAD5H

7	6	5	4	3	2	1	0
0	0	0	0	0			

Cautions 1. Setting a value other than 00 to 06 to the RC1WEEK register is prohibited.

2. Values corresponding to the month count register and day count register are not automatically stored in the day-of-week register. Be sure to set as follows after applying RV_{DD}:

Day of Week	RC1WEEK
Sunday	00H
Monday	01H
Tuesday	02H
Wednesday	03H
Thursday	04H
Friday	05H
Saturday	06H

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 04H in this sample program.

(10) Month count register (RC1MONTH)

The RC1MONTH register is an 8-bit register that takes a value of 1 to 12 (decimal) and indicates the count value of the month. It counts up when the day counter overflows.

When data is written to this register, it is written to a buffer and then to the counter up to 2 clocks (2 × 32.768 kHz) later. Set a decimal value of 01 to 12 to this register in BCD code.

This register can be read or written in 8-bit units.

Caution Setting a value other than 01 to 12 to the RC1MONTH register is prohibited.

Remarks 1. This register is reset to 01H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 04H in this sample program.

Figure 4-1-10. Format of Month Count Register (RC1MONTH)

Month count register (RC1MONTH)

Address: FFFFFAD7H

7	6	5	4	3	2	1	0
0	0	0					

(11)Year count register (RC1YEAR)

The RC1YEAR register is an 8-bit register that takes a value of 0 to 99 (decimal) and indicates the count value of the year. It counts up when the month counter overflows.

The values 00, 04, 08, …, 92, and 96 indicate a leap year.

When data is written to this register, it is written to a buffer and then to the counter up to 2 clocks (2 × 32.768 kHz) later. Set a decimal value of 00 to 99 to this register in BCD code.

This register can be read or written in 8-bit units.

Caution Setting a value other than 00 to 99 to the RC1YEAR register is prohibited.

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 10H in this sample program.

Figure 4-1-11. Format of Year Count Register (RC1YEAR)

Year count register (RC1YEAR)

Address: FFFFFAD8H

7	6	5	4	3	2	1	0

(12) Alarm minute setting register (RC1ALM)

The RC1ALM register is an 8-bit register that is used to set the minutes of the alarm.

This register can be read or written in 8-bit units.

Cautions 1. Set a decimal value of 00 to 59 to this register in BCD code. If a value outside the range is set, the alarm is not detected.

2. See the V850ES/Jx3-L user's manual for details about how to change the alarm minute setting register (RC1ALM) while the real-time counter is operating.

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 00H in this sample program.

Figure 4-1-12. Format of Alarm Minute Setting Register (RC1ALM)

Alarm minute setting register (RC1ALM)

Address: FFFFFADAH

7	6	5	4	3	2	1	0
0							

(13) Alarm hour setting register (RC1ALH)

The RC1ALH register is an 8-bit register that is used to set the hour of the alarm.

This register can be read or written in 8-bit units.

Cautions 1. Set a decimal value of 00 to 23, 01 to 12, or 21 to 32 to this register in BCD code. If a value outside the range is set, the alarm is not detected.

2. Bit 5 of the RC1ALH register indicates a.m. (0) or p.m. (1) if the AMPM bit = 0 (12-hour system) is selected.

3. See the V850ES/Jx3-L user's manual for details about how to change the alarm hour setting register (RC1ALH) while the real-time counter is operating.

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 14H in this sample program.

Figure 4-1-13. Format of Alarm Hour Setting Register (RC1ALH)

Alarm hour setting register (RC1ALH)

Address: FFFFFADBH

7	6	5	4	3	2	1	0
0	0						

(14) Alarm day-of-week setting register (RC1ALW)

The RC1ALW register is an 8-bit register that is used to set the day-of-week of the alarm.

This register can be read or written in 8-bit units.

Figure 4-1-14. Format of Alarm Day-of-Week Setting Register (RC1ALW)

Alarm day-of-week setting register (RC1ALW)

Address: FFFFFADCH

7	6	5	4	3	2	1	0
0	RC1ALW6	RC1ALW5	RC1ALW4	RC1ALW3	RC1ALW2	RC1ALW1	RC1ALW0
	Saturday	Friday	Thursday	Wednesday	Tuesday	Monday	Sunday

RC1ALW6	Alarm interrupt day-of-week bit 6
0	Do not generate alarm interrupt if RC1WEEK = 06H (Saturday).
1	Generate an alarm interrupt if the time specified by using the RC1ALM and RC1ALH registers is reached while RC1WEEK is set to 06H (Saturday).

RC1ALW5	Alarm interrupt day-of-week bit 5
0	Do not generate alarm interrupt if RC1WEEK = 05H (Friday).
1	Generate an alarm interrupt if the time specified by using the RC1ALM and RC1ALH registers is reached while RC1WEEK is set to 05H (Friday).

RC1ALW4	Alarm interrupt day-of-week bit 4
0	Do not generate alarm interrupt if RC1WEEK = 04H (Thursday).
1	Generate an alarm interrupt if the time specified by using the RC1ALM and RC1ALH registers is reached while RC1WEEK is set to 04H (Thursday).

RC1ALW3	Alarm interrupt day-of-week bit 3
0	Do not generate alarm interrupt if RC1WEEK = 03H (Wednesday).
1	Generate an alarm interrupt if the time specified by using the RC1ALM and RC1ALH registers is reached while RC1WEEK is set to 03H (Wednesday).

RC1ALW2	Alarm interrupt day-of-week bit 2
0	Do not generate alarm interrupt if RC1WEEK = 02H (Tuesday).
1	Generate an alarm interrupt if the time specified by using the RC1ALM and RC1ALH registers is reached while RC1WEEK is set to 02H (Tuesday).

RC1ALW1	Alarm interrupt day-of-week bit 1
0	Do not generate alarm interrupt if RC1WEEK = 01H (Monday).
1	Generate an alarm interrupt if the time specified by using the RC1ALM and RC1ALH registers is reached while RC1WEEK is set to 01H (Monday).

RC1ALW0	Alarm interrupt day-of-week bit 0
0	Do not generate alarm interrupt if RC1WEEK = 00H (Sunday).
1	Generate an alarm interrupt if the time specified by using the RC1ALM and RC1ALH registers is reached while RC1WEEK is set to 00H (Sunday).

Cautions 1. Be sure to set bit 7 to "0".

2. See the V850ES/Jx3-L user's manual for details about how to change the alarm day-of-week setting register (RC1ALW) while the real-time counter is operating.

Remarks 1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

2. This register is set to an initial value of 7FH (Friday) in this sample program.

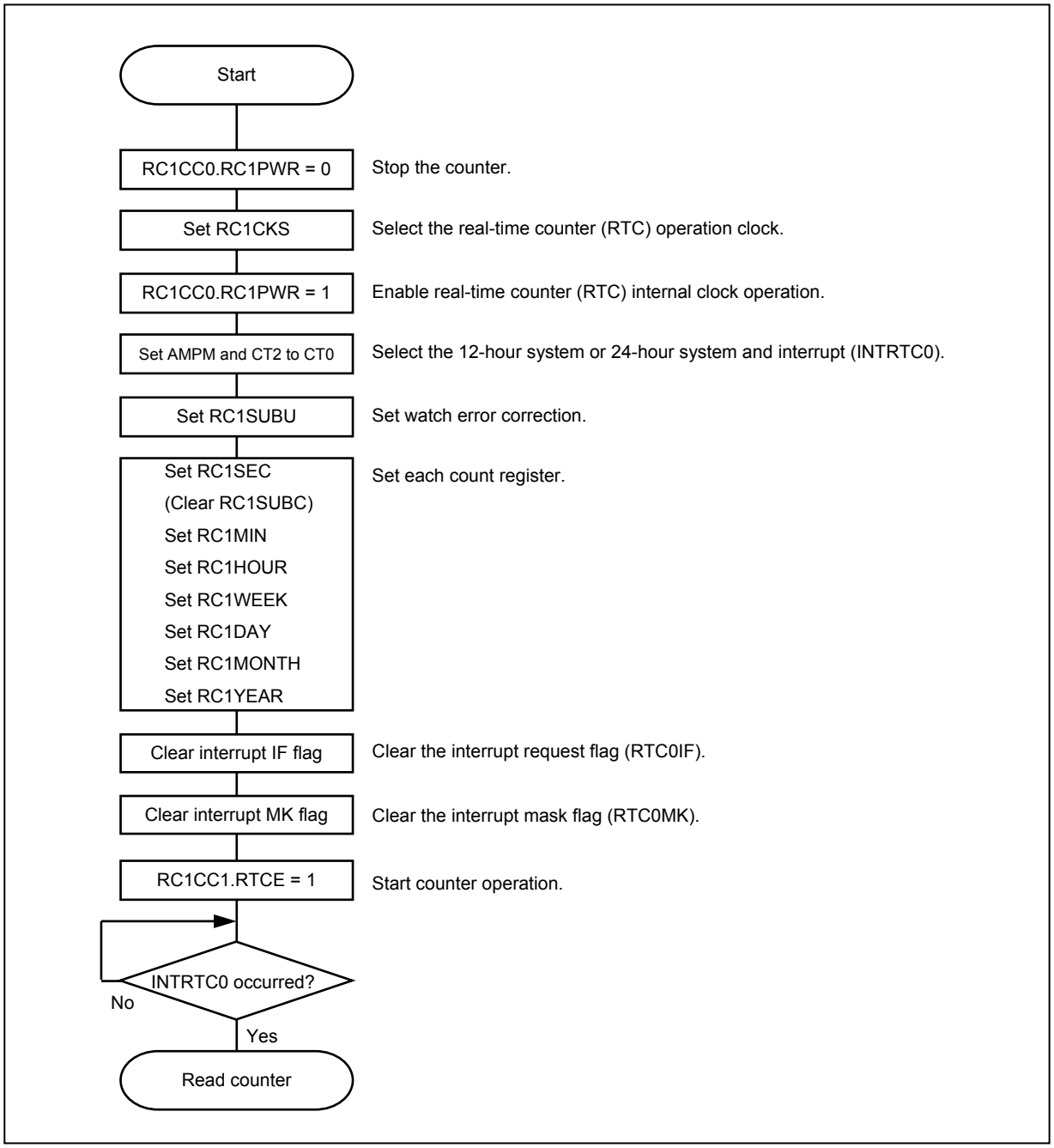
(15) Setting the registers

Set the registers used by the real-time counter as described below.

<1> Initial settings

Specify the initial settings for the watch feature and fixed-cycle interrupt as follows:

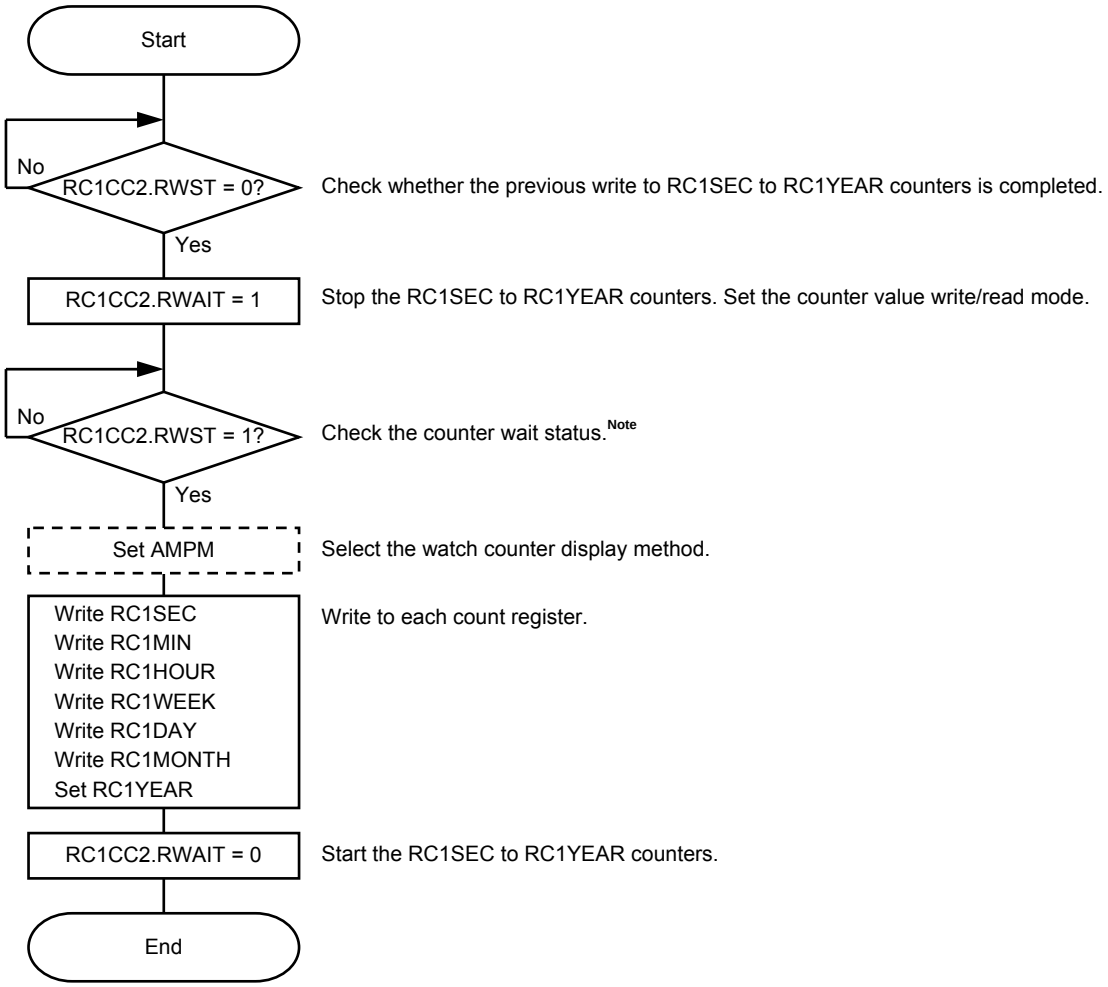
Figure 4-1-15-1. Initial Setting Procedure



<2> Rewriting counters during real-time counter operation

Set as follows when rewriting the counters (RC1SEC, RC1MIN, RC1HOUR, RC1WEEK, RC1DAY, RC1MONTH, RC1YEAR) during real-time counter operation (RC1PWR = 1).

Figure 4-1-15-2. Rewriting Counters During Clock Operation



Note Be sure to confirm that RWST = 0 before setting the stop mode.

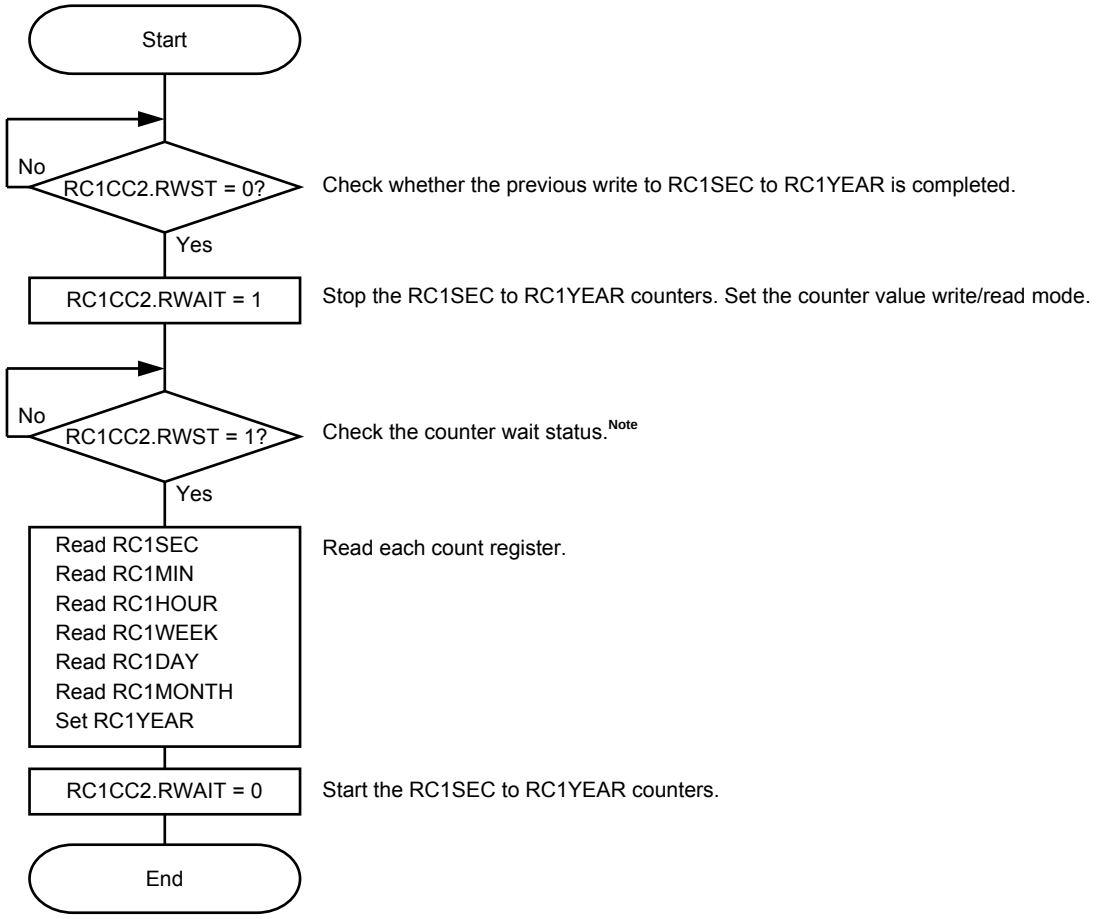
Caution **Ensure that the series of operations from setting RWAIT to 1 to clearing RWAIT to 0 are completed within 1 second. If RWAIT = 1 is set, the operation of RC1SEC to RC1YEAR is stopped. If a carry occurs from RC1SUBC while RWAIT = 1, one carry can be internally retained. However, if two or more carries occur, the number of carries cannot be retained.**

Remark RC1SEC, RC1MIN, RC1HOUR, RC1WEEK, RC1DAY, RC1MONTH, and RC1YEAR may be rewritten in any sequence.
Not all the registers have to be set or read.

<3> Reading counters during real-time counter operation

Set as follows when reading the counters (RC1SEC, RC1MIN, RC1HOUR, RC1WEEK, RC1DAY, RC1MONTH, RC1YEAR) during real-time counter operation (RC1PWR = 1, RTCE = 1).

Figure 4-1-15-3. Reading Counters During Clock Operation



Note Be sure to confirm that RWST = 0 before setting the stop mode.

Caution **Ensure that the series of operations from setting RWAIT to 1 to clearing RWAIT to 0 is completed within 1 second.**
If RWAIT = 1 is set, the operation of RC1SEC to RC1YEAR is stopped. If a carry occurs from RC1SUBC while RWAIT = 1, one carry can be internally retained. However, if two or more carries occur, the number of carries cannot be retained.

Remark RC1SEC, RC1MIN, RC1HOUR, RC1WEEK, RC1DAY, RC1MONTH, and RC1YEAR may be read in any sequence.
Not all the registers have to be set or read.

(16) Software coding example

A section of the sample program is given below as an example of how to write the software code for setting the real-time counter.

The following operations are specified in this sample code:

- Setting the subclock (f_{XT}) as the real-time operating clock.
- Setting the fixed-cycle interrupt to be generated every 0.5 seconds.
- Setting the count start time to 13:00:00 on April 1, 2010 (Thursday)
- Enabling the alarm interrupt and setting the alarm time to 14:00 everyday
- Setting the interval operation to about 3.9 ms
- Enabling the INTRTC0 and INTRTC1 interrupts

<1> Specify the interrupt handlers of the INTRTC0 and INTRTC1 interrupt.
<2> Stop the real-time counter by using bit 7 (RC1PWR) of the RC1CC0 register.
<3> Specify f_{XT} as the real-time counter operating clock by using bit 6 (RC1CKS) of the RC1CC0 register.
<4> Enable the real-time counter by using bit 7 (RC1PWR) of the RC1CC0 register.
<5> Specify the 24-hour format by using bit 3 (AMPM) of the RC1CC1 register. Set the generation cycle of the fixed-cycle interrupt (INTRTC0) to 0.5 seconds by using bits 0 to 2 (CT0 to CT2) of this register.
<6> Set the count start time to 13:00:00 on April 1, 2010 (Thursday) by using the RC1SEC, RC1MIN, RC1HOUR, RC1WEEK, RC1DAY, RC1MONTH, and RC1YEAR registers.
<7> Disable alarm matching by using bit 7 (WALE) of the RC1CC2 register.
<8> Set the alarm time to 14:00 on Friday by using the RC1ALM, RC1ALH, and RC1ALW registers.
<9> Enable alarm matching by using bit 7 (WALE) of the RC1CC2 register.
<10> Set the cycle of the interval interrupt (INTRTC2) to about 3.9 ms by using bits 0 to 2 of the RC1CC3 register. Enable the interval interrupt by using bit 7 (RINTE) of this register.
<11> Clear the INTRTC0 interrupt request flag (RTC0IF), INTRTC1 interrupt request flag (RTC1IF), and INTRTC2 interrupt request flag (RTC2IF).
<12> Enable the INTRTC0 and INTRTC1 interrupts.
<13> Enable the real-time counter count operation by using bit 7 (RTCE) of the RC1CC1 register.

```
#pragma interrupt INTRTC0  fn_IntRtc0 /* Fixed-cycle interrupt processing */ }<1>
#pragma interrupt INTRTC1  fn_IntRtc1 /* Alarm interrupt processing */
```

```
  RC1CC0.7 = 0;    /* Stop real-time counter */ }<2>
/*---------------------------------------------------------------------
Specify operation of fixed-cycle interrupt
---------------------------------------------------------------------*/
  RC1CC0 = 0b00000000;                          }<3>
  RC1CC0.7 = 1;    /* Enable real-time counter */ }<4>
  RC1CC1 = 0b00001001;                          }<5>

/* Set count start time to 13:00:00 on April 1, 2010 (Thursday) */
  RC1SEC   = 0x00;
  RC1MIN   = 0x00;
  RC1HOUR  = 0x13;
  RC1WEEK  = 0x04;                              }<6>
  RC1DAY   = 0x01;
  RC1MONTH = 0x04;
  RC1YEAR  = 0x10;
/*---------------------------------------------------------------------
Specify operation of alarm interrupt
---------------------------------------------------------------------*/
  RC1CC2 = 0b00000000;                          }<7>

  /* Specify initial setting for alarm time (14:00 everyday) */
  RC1ALW = 0x7F;
  RC1ALH = 0x14;                                }<8>
  RC1ALM = 0x00;

  RC1CC2.7 = 1;    /* Enable alarm interrupt */ }<9>
/*---------------------------------------------------------------------
Specify interval
---------------------------------------------------------------------*/
  RC1CC3 = 0b10000001;                          }<10>
```

```
  RTC0IF = 0;  /* Clear INTRTC0 interrupt request flag */
  RTC1IF = 0;  /* Clear INTRTC1 interrupt request flag */                }<11>
  RTC2IF = 0;  /* Clear INTRTC2 interrupt request flag */

  RTC0MK = 0;  /* Enable INTRTC0 interrupt */
  RTC1MK = 0;  /* Enable INTRTC1 interrupt */                            }<12>
  RTC2MK = 1;  /* Disable INTRTC2 interrupt */

   __EI(); /* Enable acknowledgment of interrupt request signals */
   RC1CC1.7 = 1;                                                         }<13>
```

4.2 Setting RTC Backup Mode

The following registers control the RTC backup mode:

- RTC backup control register 0 (RTCBUMCTL0)
- Subclock low-power operation control register (SOSCAMCTL)

(1) RTC backup control register 0 (RTCBUMCTL0)

The RTCBUMCTL0 register controls the RTC backup mode.

Figure 4-2-1. Format of RTC Backup Control Register 0 (RTCBUMCTL0)

RTC backup control register 0 (RTCBUMCTL0)

Address: FFFFFB00H

7	6	5	4	3	2	1	0
RBMEN	0	0	0	0	0	0	RBMSET

RBMEN	RTC backup mode control
0	Using RTC backup mode is disabled
1	Using RTC backup mode is enabled

RBMSET	RTC backup mode setting
0	Exiting the RTC backup mode
1	Setting the RTC backup mode When the RBMSET bit is set (to 1), switch the RTC status to the following: • Select a divided subclock (f_{XT}) as the RTC input clock • Stop RTC pin output • Stop RTC time error correction

Cautions

1. **Do not set the RBMEN and RBMSET bits to 1 at the same time. If they are set to 1 at the same time, the RTC backup mode might not operate correctly. Set the RBMEN bit to 1 first, and then set the RBMSET bit to 1.**
2. **Do not set the RBMSET bit to 1 while the RBMEN bit is 0.**
   If the RBMSET bit is set to 1 at this time, the bit is set to 1, but the RTC backup mode is not specified.
3. **The RTCBUMCTL0 register is a special register that can only be written in a specific sequence. For details about special registers, see the V850ES/Jx3-L user's manual.**
4. **Be sure to set bits 1 to 6 to "0".**

Remarks

1. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.
2. The parts written in red in the above figure are the values set by using this sample program.

(2) Subclock low-power operation control register (SOSCAMCTL)

The SOSCAMCTL register is used to set the subclock (f_{XT}) to perform low-power operations in the RTC backup mode.

Figure 4-2-2. Format of Subclock Low-Power Operation Control Register (SOSCAMCTL)

Format of subclock low-power operation control register (SOSCAMCTL)

Address: FFFFFB03H

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	AMPHS

AMPHS	Subclock (f_{XT}) oscillation mode selection
0	Normal oscillation
1	Ultra-low consumption oscillation

Cautions
1. **The SOSCAMCTL register is a special register that can only be written in a specific sequence. For details about special registers, see the V850ES/Jx3-L user's manual.**
2. **Be sure to set bits 7 to 1 to "0".**

Remarks
1. When the subclock (f_{XT}) is oscillating in the ultra-low consumption mode, the effects of noise can more easily cause an incorrect number of oscillation cycles to be counted. Before deciding to use this mode, thoroughly evaluate the effects of noise.
2. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.
3. The parts written in red in the above figure are the values set by using this sample program.

(3) Software coding example

A section of the sample program is given below as an example of the software code for setting the RTC backup mode.

(a) Initial settings

Make the following initial settings after a reset has occurred:

<1> Define the interrupt processing routine.
<2> Set RTCBUMCTL0.RBMEN to 1 to enable use of the RTC backup mode. If the system has entered the state in which it is preparing to enter RTC backup mode after reset is canceled, exit this state here.
<3> Set SOSCAMCTL.AMPHS to 0 to specify normal oscillation for the subclock (f_{XT}).
<4> Execute NOP.
<5> Specify use of the low-voltage detector as an interrupt. Set the low voltage detection level to 2.8 V.
<6> Specify the operation of the real-time counter's fixed-cycle interrupt. Specify the subclock (f_{XT}) as the operating clock.
<7> Enable the INTLVI interrupt.
<8> Enable the real-time counter.

```
#pragma  interrupt INTLVI  fn_IntLvi    /* Low-voltage detector interrupt processing */
```
<1>

```
static void fn_InitRtcBum( void )
{

 __asm("push r10                  ");
 __asm("mov  0x80,    r10         ");
 __asm("st.b r10,     PRCMD       ");
 __asm("st.b r10,     RTCBUMCTL0  ");
 __asm("pop  r10                  ");

 __asm("st.b r0,      PRCMD       ");
 __asm("st.b r0,      SOSCAMCTL   ");
 __NOP();
```
<2>
<3>
<4>

```
static void fn_InitLvi( void )
{

LVIMK        = 1;             /* Disable INTLVI interrupt */
LVIIF        = 0;             /* Clear INTLVI interrupt request flag */
__asm("st.b  r0,      PRCMD"); /* Specify use of low-voltage detector as interrupt */
__asm("st.b  r0,      LVIM");
LVIS= 0b00000000;             /* Set low voltage detection level to 2.8 V */

 /* Enable low-voltage detector */
__asm("push  r10              ");
__asm("mov   0x80,    r10     ");
__asm("st.b  r10,     PRCMD   ");
__asm("st.b  r10,     LVIM    ");
__asm("pop   r10              ");
```
<5>

```
static void fn_InitRtc( void )
{

 RC1CC0.7    = 0;     /* Stop real-time counter */

 RC1CC0      = 0b00000000;

 RC1CC0.7    = 1;     /* Enable real-time counter */

 RC1CC1      = 0b00001001;

 /* Set count start time to 13:00:00 on April 1, 2010 (Thursday) */
 RC1SEC      = 0x00;
 RC1MIN      = 0x00;
 RC1HOUR     = 0x13;
 RC1WEEK     = 0x04;
 RC1DAY      = 0x01;
 RC1MONTH    = 0x04;
 RC1YEAR     = 0x10;

LVIMK    = 0;     /* Enable INTLVI interrupt */

__EI();           /* Enable acknowledgment of interrupt request signals */

RC1CC1.7 = 1;     /* Enable real-time counter */
```
<6>
<7>
<8>

Cautions 1. RTC backup control register 0 (RTCBUMCTL0), the subclock low-power operation control register (SOSCAMCTL), and the low voltage detection register (LVIM) are special registers that can only be written in a specific sequence. For details about special registers, see the V850ES/Jx3-L user's manual.

2. For details about the low voltage detector, see the V850ES/Jx3-L user's manual.

(b) Settings when preparing to shift to RTC backup mode

In the INTLVI interrupt processing routine, check whether the voltage is low (that is, whether the LVIM.LVIF bit is 1)[Note 1] and then specify the following settings:

<1> Stop the PLL and specify the low-voltage clock (5 MHz) as the CPU clock.
<2> Disable DMA[Note 2].
<3> Disable all maskable interrupts except the NMI and INTLVI interrupts.
<4> Set SOSCAMCTL.AMPHS to 1 to specify ultra-low consumption oscillation for the subclock (f_{XT}).
<5> Set RTCBUMCTL0.RBMSET to 1 to prepare to shift to RTC backup mode.
<6> Specify the normal stop mode. (If the supply of V_{DD} stops, shift to RTC backup mode.)

```
__interrupt void fn_IntLvi( void )
{

  if( LVIF )
  {

    SELPLL = 0;          } <1>
    PLLON  = 0;

    E00    = 0;          }
    E11    = 0;          } <2>
    E22    = 0;          }
    E33    = 0;          }

    INTF0 &= 0b01111000;
    INTR0 &= 0b01111000;

    IMR3 = 0b1111111111111111;
    IMR2 = 0b1111111111111111;
    IMR1 = 0b1111111111111111;
    IMR0 = 0b1111111111111110;

    asm("push  r10         ");
    asm("mov   0x01,   r10     ");
    asm("st.b  r10,    PRCMD       ");
    asm("st.b  r10,    SOSCAMCTL   ");
    asm("pop   r10         ");

    asm("mov   0x81,   r10     ");
    asm("st.b  r10,    PRCMD       ");
    asm("st.b  r10,    RTCBUMCTL0  ");
```

```
static void fn_SetStandby( void )
{

  PSMR = 0b0000001;

  asm("push r10     ");
  asm("mov  0x02,   r10 ");
  asm("st.b r10,    PRCMD   ");
  asm("st.b r10,    PSC ");
  asm("pop  r10     ");
```

Notes 1. In this sample program, low voltage is defined as 2.2 V < V_{DD} < 2.8 V.

2. If a DMA occurs before DMA is disabled in the INTLVI interrupt routine and the power supply voltage reaches the minimum guaranteed voltage level, it will not be possible to set RTC backup mode.

Caution The subclock low-power operation control register (SOSCAMCTL) and RTC backup control register 0 (RTCBUMCTL0) are special registers that can only be written in a specific sequence. For details about special registers, see the V850ES/Jx3-L user's manual.

(c) Settings when cancelling the preparation to shift to RTC backup mode

In the INTLVI interrupt processing routine, check whether the system is returning from the low-voltage state (that is, whether the LVIM.LVIF bit is 0)[Note 1] and then specify the following settings:

<1> Set SOSCAMCTL.AMPHS to 0 to specify normal oscillation for the subclock (f_{XT}).
<2> Set RTCBUMCTL0.RBMSET to 0 to prepare to exit the RTC backup mode.
<3> Execute NOP.
<4> Enable the maskable interrupts to be used (interrupts other than INTLVI).
<5> Specify the normal clock (5 MHz × 4 = 20 MHz) as the CPU clock.

```
__interrupt void fn_IntLvi( void )
{

  if( LVIF )
  {
    Omitted if LVIM.LVIF = 1
  }
  else
  {

    asm("st.b  r0, PRCMD          ");
    asm("st.b  r0, SOSCAMCTL      ");                    <1>

    asm("push  r10            ");
    asm("mov   0x80,   r10    ");
    asm("st.b  r10,    PRCMD      ");                    <2>
    asm("st.b  r10,    RTCBUMCTL0 ");
    asm("pop   r10                ");

    NOP();                                              <3>

    KRIF    = 0;
    RTC0IF  = 0;
    RTC1IF  = 0;                                        <4>
    KRMK    = 0;
    RTC0MK  = 0;
    RTC1MK  = 0;

    PLLON   = 1;
    while( LOCK )
    {
       __NOP();                                         <5>
    }
    SELPLL = 1;
```

Caution The subclock low-power operation control register (SOSCAMCTL), RTC backup control register 0 (RTCBUMCTL0), and processor clock control register (PCC) are special registers that can only be written in a specific sequence. For details about special registers, see the V850ES/Jx3-L user's manual.

4.3 Watch Error Correction

The watch error correction feature of the real-time counter is described below.

(1) Watch error correction register (RC1SUBU)

The watch error correction register (RC1SUBU) is used when the watch error correction feature of the real-time counter is used. RC1SUBU is an 8-bit register that accurately corrects the watch when the watch is fast or slow, by changing the value (reference value: 7FFFH) overflowing from the sub-count register (RC1SUBC) to the second counter register.

Figure 4-3-1. Format of Watch Error Correction Register (RC1SUBU)

Watch error correction register (RC1SUBU)

Address: FFFFFAD9H

7	6	5	4	3	2	1	0
DEV	F6	F5	F4	F3	F2	F1	F0

DEV	Setting of watch error correction timing
0	Correct watch errors when RC1SEC (second counter) is at 00, 20, or 40 seconds (every 20 seconds).
1	Correct watch errors when RC1SEC (second counter) is at 00 seconds (every 60 seconds).

F6	Setting of watch error correction value
0	Increment the RC1SUBC count value by the value set by using the F5 to F0 bits (positive correction). Expression for calculating increment value: (Setting value of F5 to F0 bits − 1) × 2
1	Decrement the RC1SUBC count value by the value set by using the F5 to F0 bits (negative correction). Expression for calculating decrement value: (Inverted value of setting value of F5 to F0 bits + 1) × 2

Caution If the F6 to F0 bit values are {1/0, 0, 0, 0, 0, 0, 1/0}, watch error correction is not performed.

Remarks 1. Watch error correction is disabled in RTC backup mode.

2. This register is reset to 00H when the RV_{DD} power supply is applied. If a reset is triggered by a factor other than application of RV_{DD}, the values of the register at the time the reset was triggered are retained.

3. This register is set to an initial value of 00H in this sample program.

(2) Watch error correction example

The watch can be accurately counted by incrementing the RC1SUBC count value, if a positive error faster than 32.768 kHz occurs at the resonator. Similarly, if a negative error slower than 32.768 kHz occurs at the resonator, the watch can be accurately counted by decrementing the RC1SUBC count value.

The RC1SUBC correction value is determined by using the RC1SUBU.F6 to RC1SUBU.F0 bits.

The F6 bit is used to determine whether to increment or decrement RC1SUBC and the F5 to F0 bits are used to determine the RC1SUBC value.

(1) Incrementing the RC1SUBC count value

The RC1SUBC count value is incremented by the value set by using the F5 to F0 bits, by setting the F6 bit to 0.

Expression for calculating the increment value: (F5 to F0 bit value − 1) × 2

[Example of incrementing the RC1SUBC count value: F6 bit = 0]

If 15H (010101B) is set to the F5 to F0 bits

(15H − 1) × 2 = 40 (increments the RC1SUBC count value by 40)

RC1SUBC count value = 32,768 + 40 = 32,808

(2) Decrementing the RC1SUBC count value

The RC1SUBC count value is decremented by an inverted value of the value set by using the F5 to F0 bits, by setting the F6 bit to 1.

Expression for calculating the decrement value: (Inverted value of F5 to F0 bit value + 1) × 2

[Example of decrementing the RC1SUBC count value: F6 bit = 1]

If 15H (010101B) is set to the F5 to F0 bits

Inverted data of 15H (010101B) = 2AH (101010B)

(2AH + 1) × 2 = 86 (decrements the RC1SUBC count value by 86)

RC1SUBC count value = 32,768 − 86 = 32,682

(3) DEV bit

The DEV bit determines when the setting by the F6 to F0 bits is enabled.
The value set by the F6 to F0 bits is applied to the RC1SUBC count value at the next timing, not every time.

Table 4-2. DEV Bit Setting

DEV Bit Value	Timing of Applying Value to RC1SUBC
0	When RC1SEC is 00, 20, or 40 seconds.
1	When RC1SEC is 00 seconds.

[Example when F6 to F0 bits are set to 0010101B]

- If the DEV bit is 0:
  The RC1SUBC count value is 32,808 at 00, 20, or 40 seconds.
  Otherwise, it is 32,768.
- IF DEV bit is 1:
  The RC1SUBC count value is 32,808 at 00 seconds.
  Otherwise, it is 32,768.

As described above, the RC1SUBC count value is corrected every 20 seconds or 60 seconds, instead of every second, in order to match the RC1SUBC count value with the deviation of the resonator.
The range in which the resonator frequency can be actually corrected is shown below.

- If the DEV bit is 0: 32.76180000 kHz to 32.77420000 kHz
- If the DEV bit is 1: 32.76593333 kHz to 32.77006667 kHz

The range in which the frequency can be corrected when the DEV bit is 0 is three times wider than when the DEV bit is 1.
However, the accuracy of setting the frequency when the DEV bit is 1 is three times that when the DEV bit is 0.
Tables 4-3 and 4-4 show the setting values of the DEV, and F6 to F0 bits, and the corresponding frequencies that can be corrected.

Table 4-3. Range of Frequencies That Can Be Corrected When DEV Bit = 0

F6	F5 to F0	RC1SUBC Correction Value	Frequency of Connected Clock (Including Steady-State Deviation)
0	000000	No correction	–
0	000001	No correction	–
0	000010	Increments RC1SUBC count value by 2 once every 20 seconds	32.76810000 kHz
0	000011	Increments RC1SUBC count value by 4 once every 20 seconds	32.76820000 kHz
0	000100	Increments RC1SUBC count value by 6 once every 20 seconds	32.76830000 kHz
		⋮	
0	111011	Increments RC1SUBC count value by 120 once every 20 seconds	32.77400000 kHz
0	111110	Increments RC1SUBC count value by 122 once every 20 seconds	32.77410000 kHz
0	111111	Increments RC1SUBC count value by 124 once every 20 seconds	32.77420000 kHz (upper limit)
1	000000	No correction	–
1	000001	No correction	–
1	000010	Decrements RC1SUBC count value by 124 once every 20 seconds	32.76180000 kHz (lower limit)
1	000011	Decrements RC1SUBC count value by 122 once every 20 seconds	32.76190000 kHz
1	000100	Decrements RC1SUBC count value by 120 once every 20 seconds	32.76200000 kHz
		⋮	
1	111011	Decrements RC1SUBC count value by 6 once every 20 seconds	32.76770000 kHz
1	111110	Decrements RC1SUBC count value by 4 once every 20 seconds	32.76780000 kHz
1	111111	Decrements RC1SUBC count value by 2 once every 20 seconds	32.76790000 kHz

Table 4-4. Range of Frequencies That Can Be Corrected When DEV Bit = 1

F6	F5 to F0	RC1SUBC Correction Value	Frequency of Connected Clock (Including Steady-State Deviation)
0	000000	No correction	–
0	000001	No correction	–
0	000010	Increments RC1SUBC count value by 2 once every 60 seconds	32.76803333 kHz
0	000011	Increments RC1SUBC count value by 4 once every 60 seconds	32.76806667 kHz
0	000100	Increments RC1SUBC count value by 6 once every 60 seconds	32.76810000 kHz
		⋮	
0	111011	Increments RC1SUBC count value by 120 once every 60 seconds	32.77000000 kHz
0	111110	Increments RC1SUBC count value by 122 once every 60 seconds	32.77003333 kHz
0	111111	Increments RC1SUBC count value by 124 once every 60 seconds	32.77006667 kHz (upper limit)
1	000000	No correction	–
1	000001	No correction	–
1	000010	Decrements RC1SUBC count value by 124 once every 60 seconds	32.76593333 kHz (lower limit)
1	000011	Decrements RC1SUBC count value by 122 once every 60 seconds	32.76596667 kHz
1	000100	Decrements RC1SUBC count value by 120 once every 60 seconds	32.76600000 kHz
		⋮	
1	111011	Decrements RC1SUBC count value by 6 once every 60 seconds	32.76790000 kHz
1	111110	Decrements RC1SUBC count value by 4 once every 60 seconds	32.76793333 kHz
1	111111	Decrements RC1SUBC count value by 2 once every 60 seconds	32.76796667 kHz

(3) Software coding example

This sample program provides a function for setting the watch error correction feature.

This function is used to set the 1-byte data specified by the argument to the watch error correction register (RC1SUBU).

Figure 4-3-2. Function for Setting Watch Error Correction

```
static void fn_RtcCorrect( unsigned char ucReg )
{
  RC1SUBU = ucReg;

}
```

An example of using this function is shown below.

- To increment the RC1SUBC count by 2 every 20 seconds (when the frequency of the connected clock is 32.76810000 kHz), specify as follows:

```
fn_RtcCorrect( 0b00000010 );
```

- To decrement the RC1SUBC count by 4 every 60 seconds (when the frequency of the connected clock is 32.76793333 kHz), specify as follows:

```
fn_RtcCorrect( 0b11111110 );
```

5. RELATED DOCUMENTS

Document Name	PDF/Document No.
V850ES/JG3-L Hardware User's Manual	U20305J
V850ES/JG3-L (On-chip USB Controller) Hardware User's Manual	U20305E
V850ES Architecture	U15943E
PM+ Ver.6.30 User's Manual	U18416E
CA850 Ver.3.20 C Compiler Package Operation	U18512E
CA850 Ver.3.20 C Compiler Package C Language	U18513E
CA850 Ver.3.20 C Compiler Package Link Directive	U18515E
QB-MINI2 On-Chip Debug Emulator with Programming Function	U18371E
ID850QB Ver.3.40 Integrated Debugger Operation	U18604E

Document search URL http://www2.renesas.com/micro/en/documentation.html

APPENDIX A PROGRAM LIST

An excerpt from the sample program is provided below.

● main.c (C language)

```
/*********************************************************************************

    Renesas Electronics     V850ES/Jx3-L Series

*********************************************************************************
       V850ES/JG3-L Series  Sample Program
*********************************************************************************
       Real-Time Counter (RTC Backup Mode)
*********************************************************************************
Revision History
       July 2010  1st edition
*********************************************************************************
```

Overview

By using the watch feature, alarm feature, and RTC backup mode as specified in this sample program, you can achieve a watch-with-alarm that continues operating even if the supply voltage (VDD) stops.

<Main initial settings>

(Specification using option byte)

- Specify the oscillation stabilization time after a reset

(Settings for initialization after a reset)

- Specify the Initial settings for RTC backup mode
  → Set the subclock oscillation mode to normal oscillation
  → Enable the RTC backup mode
- Set the system wait control register to 1 wait
- Set the on-chip debug mode register to normal operation mode
- Specify stopping the internal oscillator
- Stop watchdog timer 2
- Specify the I/O port settings
  → Specify P50 to P53 (KR0 to KR3) as key input
  → Specify P70 as buzzer control
  → Specify P90 to P96 as LCD module control
  → Set the unused ports
- Specify the low-voltage detector settings
  → Set the low-voltage detection level to 2.8 V
  → Enable low-voltage detection
  → Wait until the supply voltage becomes 2.8 V or higher
- Specify PLL mode (5 MHz × 4 = 20 MHz operation)
- Set the CPU clock oscillation stabilization time after the standby mode is released to about 1.6 ms
- Specify the real-time counter settings
  → Set the count start time to 13:00:00 on April 1, 2010 (Thursday)
  → Set the alarm time to 14:00 everyday
  → Set the fixed-cycle interrupt operation to a cycle of 0.5 seconds
  → Enable the alarm interrupt

→ Set the interval operation to about 3.9 ms
→ Start the count operation

- Specify the initial settings for the LCD module
- Enable the key interrupt (KR0 to KR3) operation
- Specify the interrupt settings
  → Enable the INTKR interrupt
  → Enable the INTRTC0 interrupt
  → Enable the INTRTC1 interrupt
  → Enable the INTLVI interrupt
  → Enable acknowledgment of maskable interrupt request signals

<Processing after specifying initial settings>

After specifying the initial settings, the system shifts to stop mode. An interrupt is then generated and program execution starts according to the generated interrupt.

- If the 0.5-second-cycle INTRTC0 interrupt is generated, processing such as updating the watch display is executed.
- If the INTRTC1 interrupt is triggered by the occurrence of an alarm, buzzer output processing is executed.
- If the INTKR interrupt is triggered by the occurrence of a key input, key sampling starts using the 3.9-millisecond-cycle INTRTC2 interrupt. If sampling results in the detection of a valid key input, processing to refresh the watch display and the real time counter count value is executed.
- If the INTLVI interrupt is triggered by the detection of low voltage, the system prepares to enter the RTC backup mode and then shifts to stop mode. If supply of the power supply voltage (VDD) then stops, the system shifts to the RTC backup mode. However, if the INTLVI interrupt is triggered by the power supply voltage being restored from the low voltage state after the system has shifted to stop mode, the system cancels the preparation to enter the RTC backup mode and then returns to the normal operation mode.

<I/O port settings>

```
Input pins: P50 to P53
Output pins: P70, P90 to P96
Specify all unused pins as output pins.

*****************************************************************************/


/*============================================================================
  Preprocessing directive (#pragma directive)
============================================================================*/
#pragma ioreg                /* Enable peripheral I/O registers to be described*/

/*============================================================================
  Define instructions
============================================================================*/
#define __NOP()__asm("nop")   /* Enable NOP instruction to be defined as __NOP() */

/*============================================================================
  Specify interrupt handlers
============================================================================*/
#pragma interrupt INTRTC0  fn_IntRtc0     /* Fixed-cycle interrupt processing */
#pragma interrupt INTRTC1  fn_IntRtc1     /* Alarm interrupt processing */
#pragma interrupt INTLVI   fn_IntLvi      /* Low-voltage detection interrupt processing */
#pragma interrupt INTKR    fn_IntKr       /* Key interrupt processing */

/*============================================================================
    Declare function prototypes
============================================================================*/
   void main( void );                /* Main processing */
static  void fn_InitFirst( void );    /* Specify the settings of the registers to be set first */
static  void fn_InitPort( void );     /* Specify the initial settings for the ports */
static  void fn_InitLvi( void );      /* Perform low voltage detection processing */
static  void fn_InitClock( void );    /* Specify the initial settings for the clock */
static  void fn_InitRtc( void );      /* Specify the initial settings for the real-time counter */
static  void fn_InitRtcBum( void );   /* Specify the initial settings for RTC backup mode */
static  void fn_WriteRtcCounter( unsigned char ucInc );
                                      /* Processing to overwrite RTC count */
static  void fn_WriteRtcAlerm( unsigned char ucInc );
                                      /* Processing to overwrite RTC alarm */
static  void fn_RtcCorrect( unsigned char ucReg );
                                      /* Watch error correction processing */
static  void fn_KeySampling( void );  /* Perform key sampling */
static  void fn_SetStandby( void );   /* Shift to standby */
static  void fn_Display( void );      /* Refresh the display */
static  unsigned char fn_BcdCount
( unsigned char ucNum, unsigned char ucInc,
unsigned char ucMin, unsigned char ucMax );
                                      /* Count the BCD value */

/* External reference */
extern  void fn_InitLcd( void );          /* Specify the initial settings for the LCD module */
extern  void fn_LcdChrDisp
( unsigned char ucChar, unsigned char ucPos, unsigned char ucType );
                                          /* Perform character display processing */
extern  void fn_LcdCurDisp( unsigned char ucPos );
                                          /* Perform cursor display processing */

/*============================================================================
```

```
  Define variables and constants
================================================================================*/
unsigned char ucMode;                   /* Status classification */
  #define   MODE_WATCH          0x00    /* Normal operation */
  #define   MODE_TIMESETUP      0x01    /* Setting date and time */
  #define   MODE_ALARMSETUP     0x02    /* Setting alarm */
  #define   MODE_RESET          0x03    /* Post reset */
unsigned char ucDisplay;                /* Display classification */
  #define   DISP_WATCH          0x00    /* Watch display */
  #define   DISP_RESET          0x01    /* Post-reset display */
unsigned char ucSetupTime;              /* Watch setting items */
  #define   SETUP_TMNONE        0xff    /* No items */
  #define   SETUP_TMYEAR        0x00    /* Year */
  #define   SETUP_TMMONTH       0x01    /* Month */
  #define   SETUP_TMDAY         0x02    /* Day */
  #define   SETUP_TMWEEK        0x03    /* Day-of-week */
  #define   SETUP_TMHOUR        0x04    /* Hour */
  #define   SETUP_TMMIN         0x05    /* Minute */
  #define   SETUP_TMSEC         0x06    /* Second */
  #define   SETUP_TIME_NUM      7       /* Number of items */
unsigned char ucSetupAlarm;             /* Alarm setting items */
  #define   SETUP_ALNONE        0xff    /* No items */
  #define   SETUP_ALWEEK        0x00    /* Alarm day-of-week */
  #define   SETUP_ALHOUR        0x01    /* Alarm hour */
  #define   SETUP_ALMIN         0x02    /* Alarm minute */
  #define   SETUP_ALARM_NUM     3       /* Alarm second */
unsigned char ucColonBlink;             /* ":" display (False: Off, True: on) */
unsigned char ucBlinkDispTimer;         /* Timer for blinking display */
  #define   BLINKDISPTIME       3       /* Switching interval (= n * 0.5s ) */
unsigned char ucDispReq;                /* Display refresh request (False: No request, True:
                                           Request) */

unsigned char ucKeySamplingCounter;     /* Counter for key sampling */
  #define   KEYSAMPLINGCOUNT    3       /* Sampling count (at 3.9 ms (approx.) intervals) */
unsigned char ucCodeBuffer;             /* Code buffer */
unsigned char ucKeyCode;                /* Key code */
  #define   KEYCODEOFF          0x00    /* Off */
  #define   KEYCODE1            0x01    /* Key 1 */
  #define   KEYCODE2            0x02    /* Key 2 */
  #define   KEYCODE3            0x03    /* Key 3 */
  #define   KEYCODE4            0x04    /* Key 4 */
  #define   KEYCODEMLT          0xff    /* More than one key pressed at the same time */
unsigned char ucKeyEdgeInfo;            /* Key detection notification (False: No notification,
                                           True: Notification) */
unsigned char ucKeyValidInfo;           /* Valid key input notification (False: No notification,
                                           True: Notification) */

unsigned char ucBuzzerTimer;            /* Buzzer timer */
  #define   BUZZER_TIME         2       /* Buzzer output period (= n * 0.5 s ) */



/*******************************************************************************
* Title: Main processing
*******************************************************************************
* Module: void main( void )
* Arg:
* Ret :
```

```
*-------------------------------------------------------------------------------
* Note: This is the main processing executed by this sample program.
*******************************************************************************/
void main( void )
{
  __DI();             /* Disable acknowledgment of interrupt request signals */

/*-----------------------------------------------------------------------
  Specify initial settings for peripheral I/O circuits
-----------------------------------------------------------------------*/
  fn_InitRtcBum();    /* Specify the initial settings for RTC backup mode */
  RC1CC2.0 = 0;       /* Enable count register counting */
                      /* (In case the CPU is reset while counting is stopped) */

  fn_InitFirst();     /* Specify the settings of the registers to be set first */
  fn_InitPort();      /* Specify the initial settings for the ports */
  fn_InitLvi();       /* Perform low voltage detection processing */
  fn_InitClock();     /* Specify the initial settings for the clock */

  /* If the real-time counter is initialized */
  if( !RC1CC1.7 )
  {
    fn_InitRtc();     /* Specify the initial settings for the real-time counter */
  }

  fn_InitLcd();       /* Specify the initial settings for the LCD module */

/*-----------------------------------------------------------------------
  Specify the initial settings for the variables
-----------------------------------------------------------------------*/
  ucKeySamplingCounter = KEYSAMPLINGCOUNT;
                                  /* Key sampling count is reset */
  ucCodeBuffer     = KEYCODEOFF;      /* Key code buffer: Off */
  ucKeyCode        = KEYCODEOFF;      /* Key code: Off */
  ucKeyEdgeInfo    = 0;               /* Key detection: None */
  ucKeyValidInfo   = 0;               /* Valid key input: None */
  ucBuzzerTimer    = 0;               /* Buzzer timer stopped */
  ucSetupTime      = SETUP_TMNONE;    /* No time set */
  ucSetupAlarm     = SETUP_ALNONE;    /* No alarm set */
  ucColonBlink     = 1;               /* ":" display on */
  ucMode           = MODE_RESET;      /* Set to post reset */
  ucDisplay        = DISP_RESET;      /* Switch to post-reset display */
  ucBlinkDispTimer = BLINKDISPTIME;   /* Timer for blinking display: Start counting */
  ucDispReq        = 1;               /* Issue display refresh request */

/*-----------------------------------------------------------------------
  Specify settings for interrupts
-----------------------------------------------------------------------*/
  KRM      = 0b00001111;      /* Enable key interrupts (KR0 to KR3) */

  KRIF     = 0;               /* Clear INTKR interrupt request flag */
  RTC0IF   = 0;               /* Clear INTRTC0 interrupt request flag */
  RTC1IF   = 0;               /* Clear INTRTC1 interrupt request flag */
  RTC2IF   = 0;               /* Clear INTRTC2 interrupt request flag */
  LVIIF    = 0;               /* Clear INTLVI interrupt request flag */
  KRMK     = 0;               /* Enable INTKR interrupt */
  RTC0MK   = 0;               /* Enable INTRTC0 interrupt */
  RTC1MK   = 0;               /* Enable INTRTC1 interrupt */
```

```
    RTC2MK   = 1;              /* Disable INTRTC2 interrupt */
    LVIMK    = 0;              /* Enable INTLVI interrupt */

    CB3RMK   = 0;              /* Enable INTCB3R interrupt (for on-chip debugging) */

    __EI();                    /* Enable acknowledgment of interrupt request signals */

    RC1CC1.7 = 1;              /* Enable real-time counter */

/*---------------------------------------------------------------------
    Main loop
---------------------------------------------------------------------*/
    while (1)
    {
      /* Confirm no key detection or display refresh */
      if( !ucKeyEdgeInfo && !ucDispReq )
      {
          fn_SetStandby();     /* Shift to standby */
      }

      /*=============================================================*/
      /*  Key sampling                                               */
      /*=============================================================*/
      /* Cycle of about 3.9 ms */
      if( RTC2IF )
      {
          RTC2IF = 0;          /* Clear INTRTC2 interrupt request flag */

          /* Key detected */
          if( ucKeyEdgeInfo )
          {
              fn_KeySampling();  /* Perform key sampling */
          }
      }

      /*=============================================================*/
      /*  Key processing                                             */
      /*=============================================================*/
      /* There is a valid key input */
      if( ucKeyValidInfo )
      {
          ucKeyValidInfo = 0;    /* Clear valid key input notification */

          /* Determine status */
          switch( ucMode )
          {
              /*-----------------------------------------------------*/
              /*  Normal operation                                   */
              /*-----------------------------------------------------*/
              case MODE_WATCH:
                  /* Determine key code */
                  switch( ucKeyCode )
                  {
                      /* Key 1 */
                      case KEYCODE1:
                          /* Shift to setting date and time */
                          ucMode = MODE_TIMESETUP;
                          /* Specify year as setting item */
```

```
            ucSetupTime = SETUP_TMYEAR;
            /* Issue display refresh request */
            ucDispReq = 1;
            break;
        /* Key 2/3/4 */
        default:
            /* NOP */
            break;
    }
    break;
/*-----------------------------------------------------*/
/*  Setting date and time                               */
/*-----------------------------------------------------*/
case MODE_TIMESETUP:
    /* Determine key code */
    switch( ucKeyCode )
    {
        /* Key 1 */
        case KEYCODE1:
            /* Shift to setting alarm */
            ucMode = MODE_ALARMSETUP;
            /* Specify alarm day-of-week as setting item */
            ucSetupAlarm = SETUP_ALWEEK;
            ucSetupTime = SETUP_TMNONE;
            break;
        /* Key 2 */
        case KEYCODE2:
            /* Increment time */
            fn_WriteRtcCounter( 0 );
            break;
        /* Key 3 */
        case KEYCODE3:
            /* Decrement time */
            fn_WriteRtcCounter( 1 );
            break;
        /* Key 4 */
        case KEYCODE4:
            /* Change to next item */
            ucSetupTime++;
            ucSetupTime %= SETUP_TIME_NUM;
            break;
        default:
            break;
    }
    ucDispReq = 1; /* Issue display refresh request */
    break;
/*-----------------------------------------------------*/
/*  Setting alarm                                      */
/*-----------------------------------------------------*/
case MODE_ALARMSETUP:
    /* Determine key code */
    switch( ucKeyCode )
    {
        /* Key 1 */
        case KEYCODE1:
            /* Shift to normal operation */
            ucMode = MODE_WATCH;
            /* Setting item: None */
```

```
                    ucSetupAlarm = SETUP_ALNONE;
                    break;
                /* Key 2 */
                case KEYCODE2:
                    /* Increment alarm */
                    fn_WriteRtcAlerm( 0 );
                    break;
                /* Key 3 */
                case KEYCODE3:
                    /* Decrement alarm */
                    fn_WriteRtcAlerm( 1 );
                    break;
                /* Key 4 */
                case KEYCODE4:
                    /* Change to next item */
                    ucSetupAlarm++;
                    ucSetupAlarm %= SETUP_ALARM_NUM;
                    break;
                default:
                    break;
            }
            ucDispReq = 1; /* Issue display refresh request */
            break;
        /*-------------------------------------------------------*/
        /*  Post reset                                           */
        /*-------------------------------------------------------*/
        case MODE_RESET:
            ucMode = MODE_WATCH;       /* Shift to normal operation */
            ucDisplay = DISP_WATCH;    /* Change to watch display */
            ucBlinkDispTimer = 0;      /* Stop blinking display */
            ucDispReq = 1;             /* Issue display refresh request */
            break;
        default:
            break;
    }
  }

  /*=============================================================*/
  /*  Display refresh processing                                 */
  /*=============================================================*/
  /* There is a display refresh request */
  if( ucDispReq )
  {
      ucDispReq = 0; /* Clear the display refresh request */
      fn_Display();  /* Refresh the display */
  }
 }
}

/*********************************************************************
* Title: Processing to specify the settings of the registers to be set first
*********************************************************************
* Module: static void fn_InitFirst( void )
* Arg:
* Ret:
*--------------------------------------------------------------------
* Note: This processing sets the VSWC, OCDM, RCM, and WDTM2 registers.
*********************************************************************/
```

```
static void fn_InitFirst( void )
{
/*--------------------------------------------------------------------
  Specify the number of bus waits when accessing a peripheral I/O register
--------------------------------------------------------------------*/
  VSWC = 0x01;               /* Set the number of waits to 1 */

/*--------------------------------------------------------------------
  Specify the settings for the on-chip debug mode register
--------------------------------------------------------------------*/
  __asm("st.b  r0, PRCMD   "); /* Set to normal mode */
  __asm("st.b  r0, OCDM    ");

/*--------------------------------------------------------------------
  Specify the settings for watchdog timer 2
--------------------------------------------------------------------*/
  RSTOP = 1;                 /* Specify stopping the internal oscillator */
  WDTM2 = 0b00000000;        /* Specify stopping the operation of watchdog timer 2 */

}

/*********************************************************************
* Title: Processing to specify the initial settings of the ports
*********************************************************************
* Module: static void fn_InitPort( void )
* Arg:
* Ret:
*--------------------------------------------------------------------
* Note: This processing specifies the settings of the I/O ports.
*********************************************************************/
static void fn_InitPort( void )
{
/*--------------------------------------------------------------------
  Specify the setting of port 0
--------------------------------------------------------------------*/
  P0       = 0b00000000;      /* Set the output data of P02 to P06 to 0 */
  PM0      = 0b10000011;      /* Set P02 to P06 to output mode */
                              /* P02 to P06: Not used*/

/*--------------------------------------------------------------------
  Specify the setting of port 1
--------------------------------------------------------------------*/
  P1       = 0b00000000;      /* Set the output data of P10 and P11 to 0 */
  PM1      = 0b11111100;      /* Set P10 and P11 to output mode */
                              /* P10 and P11: Not used */

/*--------------------------------------------------------------------
  Specify the setting of port 3
--------------------------------------------------------------------*/
  P3       = 0b0000000000000000;  /* Set the output data of P30 to P39 to 0 */
  PM3      = 0b1111110000000000;  /* Set P30 to P39 to output mode */
                                  /* P30 to P39: Not used */

/*--------------------------------------------------------------------
  Specify the setting of port 4
--------------------------------------------------------------------*/
  P4       = 0b00000000;          /* Set the output data of P40 to P42 to 0 */
  PM4      = 0b11111000;          /* Set P40 to P42 to output mode */
```

```
                              /* P40 to P42: Not used */

/*-------------------------------------------------------------------
  Specify the setting of port 5
-------------------------------------------------------------------*/
  P5       = 0b00001111;      /* Set the output data of P50 to P53 to 1 */
                              /* Set the output data of P54 and P55 to 0 */
  PM5      = 0b11001111;      /* Set P50 to P53 to input mode */
                              /* Set P54 and P55 to output mode */
  PFCE5= 0b00000000;          /* Specify P50 to P53 to be used as KR0 to KR3 */
  PFC5  = 0b00001111;
  PMC5  = 0b00001111;
                              /* P50 to P53: Used as key inputs */
                              /* P54 and P55: Not used */

/*-------------------------------------------------------------------
  Specify the setting of port 7
-------------------------------------------------------------------*/
  P7L   = 0b00000001;         /* Set the output data of P70 to 1 */
                              /* Set the output data of P71 to P77 to 0 */
  P7H   = 0b00000000;         /* Set the output data of P78 to P711 to 0 */
  PM7L  = 0b00000000;         /* Set P70 to P77 to output mode */
  PM7H  = 0b11110000;         /* Set P78 to P711 to output mode */
                              /* P70: Used for buzzer control */
                              /* P71 to P711: Not used */

/*-------------------------------------------------------------------
  Specify the setting of port 9
-------------------------------------------------------------------*/
  P9    = 0b0000000000000000; /* Set the output data of P90 to P915 to 0 */
  PM9   = 0b0000000000000000; /* Set P90 to P915 to output mode */
                              /* P90 to P96: Used for LCD module */
                              /* P97 to P915: Not used */
                              /* * When using CSIB3 for communication */
                              /*   during on-chip debugging, do not change */
                              /*   the settings of PFC910 to PFC912 */
                              /*   or PMC910 to PMC912 */

/*-------------------------------------------------------------------
  Specify the setting of port CM
-------------------------------------------------------------------*/
  PCM.1    = 0;               /* Set the output data of PCM1 to 0 */
  PCM.2    = 0;               /* Set the output data of PCM2 to 0 */
  PCM.3    = 0;               /* Set the output data of PCM3 to 0 */
  PMCM.1   = 0;               /* Set PCM1 to output mode */
  PMCM.2   = 0;               /* Set PCM2 to output mode */
  PMCM.3   = 0;               /* Set PCM3 to output mode */
                              /* PCM0 to PCM3: Not used */
                              /* * When using CSIB3 for communication
                              /*   during on-chip debugging, do not */
                              /*   change the settings of PCM0 or PMCM0 */

/*-------------------------------------------------------------------
  Specify the setting of port CT
-------------------------------------------------------------------*/
  PCT      = 0b00000000;      /* Set the output data of PCT0, PCT1, PCT4, and PCT6 to 0 */
  PMCT     = 0b10101100;      /* Set PCT0, PCT1, PCT4, and PCT6 to output mode */
                              /* PCT0, PCT1, PCT4, and PCT6: Not used */
```

```
/*-----------------------------------------------------------------
  Specify the setting of port DH
-----------------------------------------------------------------*/
  PDH      = 0b00000000;      /* Set the output data of PDH0 to PDH4 to 0 */
  PMDH  = 0b11100000;         /* Set PDH0 to PDH4 to output mode */
                              /* PDH0 to PDH4: Not used */

/*-----------------------------------------------------------------
  Specify the setting of port DL
-----------------------------------------------------------------*/
  PDL      = 0b0000000000000000;  /* Set the output data of PDL0 to PDL15 to 0 */
  PMDL  = 0b0000000000000000;     /* Set PDL0 to PDL15 to output mode */
                                  /* PDL0 to PDL15: Not used */

}

/*********************************************************************
* Title: Low voltage detection processing
*********************************************************************
* Module: static void fn_InitLvi( void )
* Arg:
* Ret:
*--------------------------------------------------------------------
* Note: This processing starts low voltage detection by checking
*     : whether the power supply voltage is within the range required to operate the CPU clock.
* ********************************************************************/
static void fn_InitLvi( void )
{
  unsigned char ucCounter;

  LVIMK = 1;                  /* Disable the INTLVI interrupt */
  LVIIF = 0;                  /* Clear the INTLVI interrupt request flag */
  __asm("st.b  r0, PRCMD   "); /* Specify that low voltage detection is to be used as an
                                  interrupt */
  __asm("st.b  r0, LVIM    ");
  LVIS  = 0b00000000;         /* Set the low voltage detection level to 2.80 V */

  /* Enable low voltage detection */
  __asm("push  r10            ");
  __asm("mov   0x80,   r10    ");
  __asm("st.b  r10,    PRCMD  ");
  __asm("st.b  r10,    LVIM   ");
  __asm("pop   r10            ");

  /* Wait for the low voltage detector to stabilize (about 0.2 ms) */
  for( ucCounter=0; ucCounter<15; ucCounter++ ){
    __NOP();
  }

  /* Wait until the power supply voltage is higher than the low voltage detection level */
  while( LVIF )
  {
    __NOP();
  }

}
```

```
/************************************************************************
* Title: Processing to specify the initial settings of the clock
************************************************************************
* Module   : static void fn_InitClock( void )
* Arg:
* Ret:
*-----------------------------------------------------------------------
* Note: This processing specifies the clock.
************************************************************************/
static void fn_InitClock( void )
{
  SELPLL   = 1;    /* Specify the PLL mode */

  /* Specify no division of the CPU clock and specify normal oscillation for the subclock */
  __asm("st.b  r0, PRCMD   ");
  __asm("st.b  r0, PCC     ");

  /* Specify the CPU clock oscillation stabilization time after the system exits standby mode */
  OSTS  = 0b00000011;
/*          |||||+++---OSTS2-OSTS0 [Select the oscillation stabilization time/setup time]
            |||||          000: 2^10/fX
            |||||          001: 2^11/fX
            |||||          010: 2^12/fX
            |||||          011: 2^13/fX
            |||||          100: 2^14/fX
            |||||          101: 2^15/fX
            |||||          110: 2^16/fX
            |||||          111: Setting prohibited
            +++++------Always set to 0
*/

}

/************************************************************************
* Title: Processing to specify the initial settings of the real-time counter
************************************************************************
* Module: static void fn_InitRtc( void )
* Arg:
* Ret:
*-----------------------------------------------------------------------
* Note: This processing does the following:
*     : Sets the count start time to 13:00:00 on April 1, 2010 (Thursday).
*     : Specifies that the fixed-cycle interrupt (INTRTC0) occurs every 0.5 seconds.
*     : Specifies that the alarm interrupt (INTRTC1) occurs at 14:00 everyday.
*     : Specifies that the interval interrupt (INTRTC2) occurs about every 3.9 ms.
*
************************************************************************/
static void fn_InitRtc( void )
{
  RC1CC0.7 = 0;    /* Stop the real-time counter */

/*---------------------------------------------------------------------
  Specify the setting of the fixed-cycle interrupt
---------------------------------------------------------------------*/
  RC1CC0   = 0b00000000;
/*             ||||||||
               ||++++++---Always set to 0
```

```
                |+---------RC1CKS [Select the operating clock]
                |           0: The operating clock is fXT
                |           1: The operating clock is fBRG
                +----------RC1PWR [Enable or disable the real-time counter]
                            0: Disable
                            1: Enable
*/

  RC1CC0.7 = 1;    /* Enable the real-time counter */

  RC1CC1   = 0b00001001;
/*              |||||+++---CT2-CT0 [Select the cycle of the fixed-cycle interrupt (INTRTC0)]
                |||||          000: Not used
                |||||          001: Occurs once every 0.5 seconds (synchronized with the second
counter)
                |||||          010: Occurs once a second (when the second counter increments)
                |||||          011: Occurs once a minute (at 00 seconds every minute)
                |||||          100: Occurs once an hour (at 00 minutes and 00 seconds every hour)
                |||||          101: Occurs once a day (at 00 hours, 00 minutes and 00 seconds every
day)
                |||||          11x: Occurs once a month (on the first day of the month at 00 hours,
                                    00 minutes and 00 seconds)
                ||||+------AMPM [Select the 12-hour format or 24-hour format]
                ||||           0: 12-hour format (AM and PM are displayed)
                ||||           1: 24-hour format
                |||+-------CLOE0 [Enable or disable output from the RTCCL pin]
                |||            0: Disable output from the RTCCL pin (32.768 kHz)
                |||            1: Enable output from the RTCCL pin (32.768 kHz)
                ||+--------CLOE1 [Enable or disable output from the RTC1HZ pin]
                ||             0: Disable output from the RTC1HZ pin (1 Hz)
                ||             1: Enable output from the RTC1HZ pin (1 Hz)
                |+---------Always set to 0
                +----------RTCE [Enable or disable the counters]
                            0: Disable the counters
                            1: Enable the counters
*/

  /* Specify the settings for the watch error correction feature */
  fn_RtcCorrect( 0b00000000 );

  /* Set the initial value of the count start time to 13:00:00 on April 1, 2010 (Thursday) */
  RC1SEC       = 0x00;
  RC1MIN       = 0x00;
  RC1HOUR      = 0x13;
  RC1WEEK      = 0x04;
  RC1DAY       = 0x01;
  RC1MONTH     = 0x04;
  RC1YEAR      = 0x10;

/*-----------------------------------------------------------------
  Specify the settings for the alarm interrupt
-----------------------------------------------------------------*/
  RC1CC2   = 0b00000000;
/*              |||||||+---RWAIT [Specify the wait condition for the real-time counter]
                |||||||        0: Counter operation enabled
                |||||||        1: Date and time counter operation stopped
                |||||||           (The count value is read out and the system shifts to write mode)
                ||||||+----RWST [The wait status of the real-time counter]
```

```
                |||||         0: Counter is operating
                |||||         1: Date and time counters have stopped counting (The count value
                |||||            has been read out and the system is in write mode)
                |++++-----Always set to 0
                +----------WALE [Enable or disable generation of the alarm interrupt (INTRTC1)]
                           0: Do not generate an interrupt upon an alarm match
                           1: Generate an interrupt upon an alarm match
*/

  /* Set the initial value of the alarm time to 14:00 everyday */
  RC1ALW   = 0x7F; /* Day-of-week */
  RC1ALH   = 0x14; /* Hour */
  RC1ALM   = 0x00; /* Minute */

  RC1CC2.7 = 1;    /* Enable the alarm interrupt */

/*-----------------------------------------------------------------
  Specify the interval operation
-----------------------------------------------------------------*/
  RC1CC3   = 0b10000001;
/*              |||||+++---ICT2-ICT0 [Select the interval of the interval interrupt (INTRTC2)]
                |||||          000: 2^6/fXT (1.953125 ms)
                |||||          001: 2^7/fXT (3.90625 ms)
                |||||          010: 2^8/fXT (7.8125 ms)
                |||||          011: 2^9/fXT (15.625 ms)
                |||||          100: 2^10/fXT (31.25 ms)
                |||||          101: 2^11/fXT (62.5 ms)
                |||||          11x: 2^12/fXT (125 ms)
                |||++------Always set to 0
                ||+--------CKDIV [Select the output frequency of the RTCDIV pin]
                ||              0: Output 512 Hz (1.95 ms) from the RTCDIV pin
                ||              1: Output 16.384 kHz (0.061 ms) from the RTCDIV pin
                |+---------CLOE2 [Enable or disable RTCDIV pin output]
                |              0: Disable RTCDIV pin output
                |              1: Enable RTCDIV pin output
                +----------RINTE [Enable or disable generation of the interval interrupt (INTRTC2)]
                               0: Do not generate interrupt
                               1: Generate interrupt
*/

}

/***********************************************************************
* Title: Processing to specify the initial settings of RTC backup mode
***********************************************************************
* Module: static void fn_InitRtcBum( void )
* Arg:
* Ret:
*----------------------------------------------------------------------
* Note: This processing enables the use of RTC backup mode.
***********************************************************************/
static void fn_InitRtcBum( void )
{
  /* Enable the use of RTC backup mode */
  /* (If the system is in the state in which it is preparing to enter backup mode after reset is
cancelled, exit this state here) */
  __asm("push  r10               ");
  __asm("mov   0x80,   r10       ");
```

```
    __asm("st.b  r10,    PRCMD      ");
    __asm("st.b  r10,    RTCBUMCTL0 ");
    __asm("pop   r10                ");

    /* Set the subclock to normal oscillation mode */
    __asm("st.b  r0,     PRCMD      ");
    __asm("st.b  r0,     SOSCAMCTL  ");

    __NOP();

}

/***********************************************************************
* Title: Processing to overwrite the real-time counter count values
***********************************************************************
* Module: static void fn_WriteRtcCounter( unsigned char ucInc )
* Arg: False: Increment, True: Decrement
* Ret:
*----------------------------------------------------------------------
* Note: This processing overwrites the count values of the real-time counter.
***********************************************************************/
static void fn_WriteRtcCounter( unsigned char ucInc )
{
  /* Specify the number of days based on the month */
  const unsigned char aDays[12]
   = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  unsigned char ucWork, ucRet;

  while( RC1CC2.1 )    /* Check that the previous overwrite processing is complete */
  {
    __NOP();
  }
  RC1CC2.0 = 1;        /* Stop the count registers counting */
  while( !RC1CC2.1 )   /* Check the wait status of the counter */
  {
    __NOP();
  }

  /* Determine the setting item */
  switch( ucSetupTime )
  {
    /* Year */
    case SETUP_TMYEAR:
        RC1YEAR = fn_BcdCount( RC1YEAR, ucInc, 0, 99 );
        break;
    /* Month */
    case SETUP_TMMONTH:
        RC1MONTH = fn_BcdCount( RC1MONTH, ucInc, 1, 12 );
        break;
    /* Day */
    case SETUP_TMDAY:
        /* Convert month to integer value */
        ucWork = ( (RC1MONTH >> 4)*10 + (RC1MONTH & 0x0f) );

        /* February if it is a leap year */
        if( !( ( (RC1YEAR >> 4)*10 + (RC1YEAR & 0x0f) ) % 4 )
        && (RC1MONTH == 0x02) )
        {
```

```
            RC1DAY = fn_BcdCount( RC1DAY, ucInc, 1, aDays[ucWork-1]+1 );
        }
        /* Otherwise */
        else
        {
            RC1DAY = fn_BcdCount( RC1DAY, ucInc, 1, aDays[ucWork-1] );
        }
        break;
    /* Day-of-week */
    case SETUP_TMWEEK:
        RC1WEEK = fn_BcdCount( RC1WEEK, ucInc, 0, 6 );
        break;
    /* Hour */
    case SETUP_TMHOUR:
        RC1HOUR = fn_BcdCount( RC1HOUR, ucInc, 0, 23 );
        break;
    /* Minute */
    case SETUP_TMMIN:
        RC1MIN = fn_BcdCount( RC1MIN, ucInc, 0, 59 );
        break;
    /* Second */
    case SETUP_TMSEC:
        RC1SEC = fn_BcdCount( RC1SEC, ucInc, 0, 59 );
        break;
    default:
        break;
  }

  RC1CC2.0 = 0;    /* Enable count register counting */
}

/************************************************************************
* Title: Processing to overwrite the alarm setting of the real-time counter
************************************************************************
* Module: static void fn_WriteRtcAlerm( unsigned char ucInc )
* Arg: False: Increment, True: Decrement
* Ret:
*-----------------------------------------------------------------------
* Note: This processing overwrites the alarm setting of the real-time counter.
************************************************************************/
static void fn_WriteRtcAlerm( unsigned char ucInc )
{
  unsigned char ucCnt;

  RTC1MK = 1;      /* Disable the INTRTC1 interrupt */
  RC1CC2.7 = 0;    /* Disable the alarm interrupt */

  /* Determine the setting item */
  switch( ucSetupAlarm )
  {
    /* Day-of-week */
    case SETUP_ALWEEK:
        /* If incrementing */
        if( !ucInc )
        {
            /* If "everyday" */
            if( RC1ALW == 0b01111111 )
            {
```

```
                RC1ALW = 0b00000001;   /* "Sunday" */
            }
            /* If "Saturday" */
            else if( RC1ALW == 0b01000000 )
            {
                RC1ALW = 0b01111111;   /* "Everyday" */
            }
            /* Otherwise */
            else
            {
                RC1ALW <<= 1;          /* Next day */
            }
        }
        /* If decrementing */
        else
        {
            /* If "everyday" */
            if( RC1ALW == 0b01111111 )
            {
                RC1ALW = 0b01000000;   /* "Saturday" */
            }
            /* If "Sunday" */
            else if( RC1ALW == 0b00000001 )
            {
                RC1ALW = 0b01111111;   /* "Everyday" */
            }
            else
            {
                RC1ALW >>= 1;          /* Previous day */
            }
        }
        break;
    /* Hour */
    case SETUP_ALHOUR:
        RC1ALH = fn_BcdCount( RC1ALH, ucInc, 0, 23 );
        break;
    /* Minute */
    case SETUP_ALMIN:
        RC1ALM = fn_BcdCount( RC1ALM, ucInc, 0, 59 );
        break;
    default:
        break;
  }

  RTC1IF = 0;      /* Clear the INTRTC1 interrupt request flag */
  RTC1MK = 0;      /* Enable the INTRTC1 interrupt */

  RC1CC2.7 = 1;    /* Enable the alarm interrupt */
}

/************************************************************************
* Title: Processing to count the BCD value
************************************************************************
* Module   : static unsigned char fn_BcdCount
*          : ( unsigned char ucNum, unsigned char ucInc,
*          :   unsigned char ucMin, unsigned char ucMax )
* Arg      : ucNum: Added value (BCD format)
*          : ucInc: Inc/Dec (True: Increment, False: Decrement)
```

```
*           : ucMin: Minimum value
*           : ucMax: Maximum value
* Ret       : Calculation result (BCD format)
*-----------------------------------------------------------------------
* Note: This processing increments and decrements the BCD value.
************************************************************************/
static unsigned char fn_BcdCount
( unsigned char ucNum, unsigned char ucInc,
unsigned char ucMin, unsigned char ucMax )
{
  unsigned char ucWork;

  /* Convert the added value to an integer value */
  ucWork = ((ucNum & 0xf0)>>4)*10 + (ucNum & 0x0f);

  /* Increment */
  if( !ucInc )
  {
    ucWork++;

    /* If the maximum value is exceeded */
    if( ucWork > ucMax )
    {
        ucWork = ucMin;/* Set the minimum value */
    }
  }
  /* Decrement */
  else
  {
    /* If the value is the minimum value */
    if( ucWork == ucMin )
    {
        ucWork = ucMax;/* Set the maximum value */
    }
    /* Otherwise */
    else
    {
        ucWork--;
    }
  }

  /* Convert the calculation result to the BCD format */
  ucWork = ((ucWork / 10)<< 4) + (ucWork % 10);

  return ucWork;
}

/************************************************************************
* Title: Processing to shift to standby
************************************************************************
* Module: static void fn_SetStandby( void )
* Arg:
* Ret:
*-----------------------------------------------------------------------
* Note: This processing shifts the system to standby.
************************************************************************/
static void fn_SetStandby( void )
{
```

```
  /* Standby mode used: Normal stop mode */
  PSMR = 0b0000001;

  /* Shift to stop mode */
  __asm("push  r10            ");
  __asm("mov   0x02,   r10    ");
  __asm("st.b  r10,    PRCMD  ");
  __asm("st.b  r10,    PSC    ");
  __asm("pop   r10            ");

  __NOP();
  __NOP();
  __NOP();
  __NOP();
  __NOP();
}

/************************************************************************
* Title: Watch error correction processing
************************************************************************
* Module: static void fn_RtcCorrect( unsigned char ucReg )
* Arg: Set value
* Ret:
*-----------------------------------------------------------------------
* Note: This processing specifies settings for the watch error correction feature of the real-time
counter.
************************************************************************/
static void fn_RtcCorrect( unsigned char ucReg )
{
  RC1SUBU = ucReg;

/*RC1SUBU  = 0bxxxxxxxx;
             ||++++++--- F5-F0
             |+--------- F6 [Set the watch error correction value]
             |               0: Increment the RC1SUBC count value by the value set to the
             |                  F5 to F0 bits (positive correction)
             |                  Expression for calculating increment:
             |                  (Value set to bits F5 to F0 – 1) x 2
             |               1: Decrement the RC1SUBC count value by the value set to bits
             |                  F5 to F0 (negative correction)
             |                  Expression for calculating decrement:
             |                  (Inverted value of value set to bits F5 to F0 + 1) x 2
             |                  If the value set to bits F6 to F0 is {1/0, 0, 0, 0, 0, 0, 1/0},
             |                  watch error correction is not executed.
             |
             +---------- DEV [Specify the watch error correction timing]
                              0: Execute watch error correction when RC1SEC (second counter)
                                 is 00, 20, and 40 (every 20 seconds)
                              1: Execute watch error correction when RC1SEC (second counter)
                                 is 00 (every 60 seconds)
*/

}

/************************************************************************
* Title: Fixed-cycle interrupt processing (using the INTRTC0 interrupt)
************************************************************************
```

```
* Module: __interrupt void fn_IntRtc0( void )
* Arg:
* Ret:
*-----------------------------------------------------------------------
* Note: This processing executes INTRTC0 interrupt processing.
************************************************************************/
__interrupt void fn_IntRtc0( void )
{
  /* If the buzzer is being output */
  if( ucBuzzerTimer )
  {
    ucBuzzerTimer--;   /* Count the value of the buzzer timer */

    /* If the buzzer output period has ended */
    if( !ucBuzzerTimer )
    {
        P7L.0 = 1; /* Stop buzzer output */
    }
  }

  /* If the display is blinking */
  if( ucBlinkDispTimer )
  {
    ucBlinkDispTimer--;    /* Count the value of the timer for blinking display */

    /* If it is time to switch the display */
    if( !ucBlinkDispTimer )
    {
        /* Switch the watch display to the post-reset display */
        if( ucDisplay == DISP_WATCH )
        {
            ucDisplay = DISP_RESET;
        }
        /* Switch the post-reset display to the watch display */
        else
        {
            ucDisplay = DISP_WATCH;
        }

        /* Resume counting the value of the timer for blinking display */
        ucBlinkDispTimer = BLINKDISPTIME;
    }
  }

  ucColonBlink ^= 1;   /* Switch the ":" display from on to off and vice versa */
  ucDispReq = 1;       /* Issue display refresh request */

  /*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*/
  /*                                                                      */
  /*                                                                      */
  /* Add any processing using the fixed-cycle interrupt here */
  /*                                                                      */
  /*                                                                      */
  /*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*/

}

/************************************************************************
```

```
* Title: Alarm interrupt processing (using the INTRTC1 interrupt)
************************************************************************
* Module   : __interrupt void fn_IntRtc1( void )
* Arg      :
* Ret      :
*-----------------------------------------------------------------------
* Note: This processing executes INTRTC1 interrupt processing.
***********************************************************************/
__interrupt void fn_IntRtc1( void )
{
  /* If normal operation */
  if( ucMode == MODE_WATCH )
  {
    P7L.0 = 0;                    /* Start buzzer output */
    ucBuzzerTimer = BUZZER_TIME;  /* Start counting value of timer for buzzer output */
  }

  /*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*/
  /*                                                             */
  /*                                                             */
  /*    Add any processing using the alarm interrupt here        */
  /*                                                             */
  /*                                                             */
  /*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*/

}

/***********************************************************************
* Title: Low voltage detection interrupt processing (using the INTLVI interrupt)
************************************************************************
* Module: __interrupt void fn_IntLvi( void )
* Arg:
* Ret:
*-----------------------------------------------------------------------
* Note: This processing sets the state in which the system is preparing to enter or cancel the
preparation to enter the RTC backup mode.
***********************************************************************/
__interrupt void fn_IntLvi( void )
{
  /*---------------------------------------------------------------*/
  /*    If the voltage has dropped from the normal operating voltage */
  /*    level to the low voltage detection level                     */
  /*---------------------------------------------------------------*/
  if( LVIF )
  {
    /* Set the operating clock to a frequency that accords with the low voltage detection state */
    SELPLL  = 0;    /* Specify clock-through mode */
    PLLON   = 0;    /* Stop the PLL */

    /* Disable DMA */
    E00     = 0;
    E11     = 0;
    E22     = 0;
    E33     = 0;

    /* Disable NMI pin edge detection */
    INTF0   &= 0b01111000;
    INTR0   &= 0b01111000;
```

```
/*              +|||||++--Always set to 0
                 ||||+----NMI
                 |||+-----INTP0
                 ||+------INTP1
                 |+-------INTP2
                 +--------INTP3
*/

    /* Disable all interrupts except INTLVI */
    IMR3    = 0b1111111111111111;
    IMR2    = 0b1111111111111111;
    IMR1    = 0b1111111111111111;
    IMR0    = 0b1111111111111110;

    /* Set the subclock to ultra-low consumption mode */
    __asm("push    r10               ");
    __asm("mov     0x01,   r10       ");
    __asm("st.b    r10,    PRCMD     ");
    __asm("st.b    r10,    SOSCAMCTL ");
    __asm("pop     r10               ");

    /* Set the state in which the system is preparing to enter RTC backup mode */
    __asm("mov     0x81,   r10       ");
    __asm("st.b    r10,    PRCMD     ");
    __asm("st.b    r10,    RTCBUMCTL0 ");

    /* Shift to standby */
    fn_SetStandby();
  }
  /*-----------------------------------------------------------------*/
  /*    If the voltage has risen from the low voltage detection    */
  /*    level to the normal operating voltage level                */
  /*-----------------------------------------------------------------*/
  else
  {
    /* Set the subclock to normal oscillation mode */
    __asm("st.b    r0,     PRCMD     ");
    __asm("st.b    r0,     SOSCAMCTL ");

    /* Set the state in which the system is preparing to exit RTC backup mode */
    __asm("push    r10               ");
    __asm("mov     0x80,   r10       ");
    __asm("st.b    r10,    PRCMD     ");
    __asm("st.b    r10,    RTCBUMCTL0 ");
    __asm("pop     r10               ");

    __NOP();

    /* Disable all interrupts used except INTLVI */
    KRIF    = 0;    /* Clear the INTKR interrupt request flag */
    RTC0IF  = 0;    /* Clear the INTRTC0 interrupt request flag */
    RTC1IF  = 0;    /* Clear the INTRTC1 interrupt request flag */
    KRMK    = 0;    /* Enable the INTKR interrupt */
    RTC0MK  = 0;    /* Enable the INTRTC0 interrupt */
    RTC1MK  = 0;    /* Enable the INTRTC1 interrupt */

    /* Specify the normal clock as the operating clock */
    PLLON   = 1;    /* Enable the PLL */
```

```
    while( LOCK )
    {
        __NOP();    /* Wait for PLL operation to stabilize */
    }
    SELPLL  = 1;    /* Specify the PLL mode */

    ucDispReq = 1; /* Issue the display refresh request */
  }
}

/************************************************************************
* Title: Key interrupt processing (using the INTKR interrupt)
************************************************************************
* Module: __interrupt void fn_IntKr( void )
* Arg:
* Ret:
*-----------------------------------------------------------------------
* Note: This processing issues a key detection notification.
************************************************************************/
__interrupt void fn_IntKr( void )
{
  ucKeyEdgeInfo = 1;   /* Issue a key detection notification */
}

/************************************************************************
* Title: Key sampling processing
************************************************************************
* Module: static void fn_KeySampling( void )
* Arg:
* Ret:
*-----------------------------------------------------------------------
* Note: This processing samples the key input ports.
************************************************************************/
static void fn_KeySampling( void )
{
  unsigned char ucWork;

  /* Obtain the status of the key input ports and convert it into the key code */
  switch( P5 & 0x0f )
  {
    /* Active level: No data */
    case 0b00001111:
        /* Code: Off */
        ucWork = KEYCODEOFF;
        break;
    /* Active level: Bit 0 */
    case 0b00001110:
        /* Code: Key 1 */
        ucWork = KEYCODE1;
        break;
    /* Active level: Bit 1 */
    case 0b00001101:
        /* Code: Key 2 */
        ucWork = KEYCODE2;
        break;
    /* Active level: Bit 2 */
    case 0b00001011:
        /* Code: Key 3 */
```

```
        ucWork = KEYCODE3;
        break;
    /* Active level: Bit 3 */
    case 0b00000111:
        /* Code: Key 4 */
        ucWork = KEYCODE4;
        break;
    /* Active level: Multiple data */
    default:
        /* Code: More than one key pressed at the same time */
        ucWork = KEYCODEMLT;
        break;
  }

  /* If the code does not match the previous code */
  if( ucCodeBuffer != ucWork )
  {
    /* Reset the sampling count */
    ucKeySamplingCounter = KEYSAMPLINGCOUNT;
    /* Save the current code */
    ucCodeBuffer = ucWork;
  }
  /* If the code matches the previous code */
  else
  {
    /* Count the number of samplings (sampling count) */
    ucKeySamplingCounter--;

    /* If the sampling count has reached the specified number */
    if( !ucKeySamplingCounter )
    {
        /* Reset the sampling count */
        ucKeySamplingCounter = KEYSAMPLINGCOUNT;

        /* To update the key code */
        if( ucCodeBuffer != ucKeyCode )
        {
            /* To update the key code because more than one key was pressed at the same time */
            if( ucKeyCode == KEYCODEMLT )
            {
                /* To turn the key off after more than one key was pressed at the same time */
                if( ucCodeBuffer == KEYCODEOFF )
                {
                    /* Updated key code: Key code off */
                    ucKeyCode = KEYCODEOFF;
                }
            }
            /* Otherwise */
            else
            {
                /* If the key code is not off, and more than one key was not pressed at the same
time */
                if( (ucCodeBuffer != KEYCODEOFF)
                 && (ucCodeBuffer != KEYCODEMLT) )
                {
                    /* Issue a valid key input notification */
                    ucKeyValidInfo = 1;
                }
```

```
                /* Update the key code */
                ucKeyCode = ucCodeBuffer;
            }

            /* To turn off the key code */
            if( ucKeyCode == KEYCODEOFF )
            {
                /* Clear the key detection notification */
                ucKeyEdgeInfo = 0;
            }
        }
    }
  }
}

/*****************************************************************************
* Title: Processing to refresh the display
*****************************************************************************
* Module: static void fn_Display( void )
* Arg:
* Ret:
*----------------------------------------------------------------------------
* Note: This processing applies display data to the display device.
*****************************************************************************/
static void fn_Display( void )
{
  unsigned char ucPos, ucBar;

  ucPos = 0;   /* Specify the initial display position setting */

  switch( ucDisplay )
  {
    /*=========================================================*/
    /*  Watch display                                          */
    /*=========================================================*/
    /*      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+           */
    /*      |1|0|.|0|4|.|0|1|T|H|U| |1|3|:|0|0|:|0|0|           */
    /*      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+           */
    /*      | | |A|L|A|R|M| |A|L|L| |1|4|:|0|0| | | |           */
    /*      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+           */
    /*=========================================================*/
    case DISP_WATCH:
        /*-------------------------------------------------------*/
        /*  Specify the character display settings    */
        /*-------------------------------------------------------*/
        while( RC1CC2.1 )  /* Check that the previous overwrite processing is complete */
        {
            __NOP();
        }
        RC1CC2.0 = 1;      /* Stop the count registers counting */
        while( !RC1CC2.1 ) /* Check the wait status of the counters */
        {
            __NOP();
        }

        /* "Year" */
        fn_LcdChrDisp( ((RC1YEAR >> 4) & 0x0f), ucPos++, 0 );       /* Upper digit */
```

```
fn_LcdChrDisp( (RC1YEAR & 0x0f), ucPos++, 0 );          /* Lower digit */

/* '.' */
fn_LcdChrDisp( '.', ucPos++, 1 );

/* "Month" */
fn_LcdChrDisp( ((RC1MONTH >> 4) & 0x0f), ucPos++, 0 );   /* Upper digit */
fn_LcdChrDisp( (RC1MONTH & 0x0f), ucPos++, 0 );          /* Lower digit */

/* '.' */
fn_LcdChrDisp( '.', ucPos++, 1 );

/* "Day" */
fn_LcdChrDisp( ((RC1DAY >> 4) & 0x0f), ucPos++, 0 ); /* Upper digit */
fn_LcdChrDisp( (RC1DAY & 0x0f), ucPos++, 0 );        /* Lower digit */

/* "Day-of-week" */
switch( RC1WEEK )
{
    /* Sunday */
    case 0x00:
        fn_LcdChrDisp( 'S', ucPos++, 1 );
        fn_LcdChrDisp( 'U', ucPos++, 1 );
        fn_LcdChrDisp( 'N', ucPos++, 1 );
        break;
    /* Monday */
    case 0x01:
        fn_LcdChrDisp( 'M', ucPos++, 1 );
        fn_LcdChrDisp( 'O', ucPos++, 1 );
        fn_LcdChrDisp( 'N', ucPos++, 1 );
        break;
    /* Tuesday */
    case 0x02:
        fn_LcdChrDisp( 'T', ucPos++, 1 );
        fn_LcdChrDisp( 'U', ucPos++, 1 );
        fn_LcdChrDisp( 'E', ucPos++, 1 );
        break;
    /* Wednesday */
    case 0x03:
        fn_LcdChrDisp( 'W', ucPos++, 1 );
        fn_LcdChrDisp( 'E', ucPos++, 1 );
        fn_LcdChrDisp( 'D', ucPos++, 1 );
        break;
    /* Thursday */
    case 0x04:
        fn_LcdChrDisp( 'T', ucPos++, 1 );
        fn_LcdChrDisp( 'H', ucPos++, 1 );
        fn_LcdChrDisp( 'U', ucPos++, 1 );
        break;
    /* Friday */
    case 0x05:
        fn_LcdChrDisp( 'F', ucPos++, 1 );
        fn_LcdChrDisp( 'R', ucPos++, 1 );
        fn_LcdChrDisp( 'I', ucPos++, 1 );
        break;
    /* Saturday */
    case 0x06:
        fn_LcdChrDisp( 'S', ucPos++, 1 );
```

```
        fn_LcdChrDisp( 'A', ucPos++, 1 );
        fn_LcdChrDisp( 'T', ucPos++, 1 );
        break;
    default:
        break;
}

/* Blank */
fn_LcdChrDisp( ' ', ucPos++, 1 );

/* "Hour" */
fn_LcdChrDisp( ((RC1HOUR >> 4) & 0x0f), ucPos++, 0 );    /* Upper digit */
fn_LcdChrDisp( (RC1HOUR & 0x0f), ucPos++, 0 );           /* Lower digit */

/* ':' on */
if( ucColonBlink )
{
    fn_LcdChrDisp( ':', ucPos++, 1 );
}
/* ':' off */
else
{
    fn_LcdChrDisp( ' ', ucPos++, 1 );
}

/* "Minute" */
fn_LcdChrDisp( ((RC1MIN >> 4) & 0x0f), ucPos++, 0 ); /* Upper digit */
fn_LcdChrDisp( (RC1MIN & 0x0f), ucPos++, 0 );        /* Lower digit */

/* ':' on */
if( ucColonBlink )
{
    fn_LcdChrDisp( ':', ucPos++, 1 );
}
/* ':' off */
else
{
    fn_LcdChrDisp( ' ', ucPos++, 1 );
}

/* "Second" */
fn_LcdChrDisp( ((RC1SEC >> 4) & 0x0f), ucPos++, 0 ); /* Upper digit */
fn_LcdChrDisp( (RC1SEC & 0x0f), ucPos++, 0 );        /* Lower digit */

RC1CC2.0 = 0;  /* Enable count register counting */

/* Fixed alarm display */
fn_LcdChrDisp( ' ', ucPos++, 1 );
fn_LcdChrDisp( ' ', ucPos++, 1 );
fn_LcdChrDisp( 'A', ucPos++, 1 );
fn_LcdChrDisp( 'L', ucPos++, 1 );
fn_LcdChrDisp( 'A', ucPos++, 1 );
fn_LcdChrDisp( 'R', ucPos++, 1 );
fn_LcdChrDisp( 'M', ucPos++, 1 );
fn_LcdChrDisp( ' ', ucPos++, 1 );

/* Alarm day-of-week */
switch( RC1ALW )
```

```
{
    /* Everyday */
    case 0b01111111:
        fn_LcdChrDisp( 'A', ucPos++, 1 );
        fn_LcdChrDisp( 'L', ucPos++, 1 );
        fn_LcdChrDisp( 'L', ucPos++, 1 );
        break;
    /* Sunday */
    case 0b00000001:
        fn_LcdChrDisp( 'S', ucPos++, 1 );
        fn_LcdChrDisp( 'U', ucPos++, 1 );
        fn_LcdChrDisp( 'N', ucPos++, 1 );
        break;
    /* Monday */
    case 0b00000010:
        fn_LcdChrDisp( 'M', ucPos++, 1 );
        fn_LcdChrDisp( 'O', ucPos++, 1 );
        fn_LcdChrDisp( 'N', ucPos++, 1 );
        break;
    /* Tuesday */
    case 0b00000100:
        fn_LcdChrDisp( 'T', ucPos++, 1 );
        fn_LcdChrDisp( 'U', ucPos++, 1 );
        fn_LcdChrDisp( 'E', ucPos++, 1 );
        break;
    /* Wednesday */
    case 0b00001000:
        fn_LcdChrDisp( 'W', ucPos++, 1 );
        fn_LcdChrDisp( 'E', ucPos++, 1 );
        fn_LcdChrDisp( 'D', ucPos++, 1 );
        break;
    /* Thursday */
    case 0b00010000:
        fn_LcdChrDisp( 'T', ucPos++, 1 );
        fn_LcdChrDisp( 'H', ucPos++, 1 );
        fn_LcdChrDisp( 'U', ucPos++, 1 );
        break;
    /* Friday */
    case 0b00100000:
        fn_LcdChrDisp( 'F', ucPos++, 1 );
        fn_LcdChrDisp( 'R', ucPos++, 1 );
        fn_LcdChrDisp( 'I', ucPos++, 1 );
        break;
    /* Saturday */
    case 0b01000000:
        fn_LcdChrDisp( 'S', ucPos++, 1 );
        fn_LcdChrDisp( 'A', ucPos++, 1 );
        fn_LcdChrDisp( 'T', ucPos++, 1 );
        break;
    default:
        break;
}

/* Blank */
fn_LcdChrDisp( ' ', ucPos++, 1 );

/* Alarm time */
fn_LcdChrDisp( ((RC1ALH >> 4) & 0x0f), ucPos++, 0 );     /* Upper digit */
```

```
fn_LcdChrDisp( (RC1ALH & 0x0f), ucPos++, 0 );        /* Lower digit */

/* ':' */
fn_LcdChrDisp( ':', ucPos++, 1 );

/* Alarm minute */
fn_LcdChrDisp( ((RC1ALM >> 4) & 0x0f), ucPos++, 0 ); /* Upper digit */
fn_LcdChrDisp( (RC1ALM & 0x0f), ucPos++, 0 );        /* Lower digit */

/* Blank */
fn_LcdChrDisp( ' ', ucPos++, 1 );
fn_LcdChrDisp( ' ', ucPos++, 1 );
fn_LcdChrDisp( ' ', ucPos++, 1 );

/*-------------------------------------------------------*/
/*  Specify the cursor display settings                  */
/*-------------------------------------------------------*/
/* Determine the setting item */
switch( ucSetupTime )
{
    /* "Year" is being set */
    case SETUP_TMYEAR:
        /* Cursor position: Lower digit of year*/
        fn_LcdCurDisp( 1 );
        break;
    /* "Month" is being set */
    case SETUP_TMMONTH:
        /* Cursor position: Lower digit of month */
        fn_LcdCurDisp( 4 );
        break;
    /* "Day" is being set */
    case SETUP_TMDAY:
        /* Cursor position: Lower digit of day */
        fn_LcdCurDisp( 7 );
        break;
    /* "Day-of-week" is being set */
    case SETUP_TMWEEK:
        /* Cursor position: First letter of day-of-week */
        fn_LcdCurDisp( 8 );
        break;
    /* "Hour" is being set */
    case SETUP_TMHOUR:
        /* Cursor position: Lower digit of hour*/
        fn_LcdCurDisp( 13 );
        break;
    /* "Minute" is being set */
    case SETUP_TMMIN:
        /* Cursor position: Lower digit of minute */
        fn_LcdCurDisp( 16 );
        break;
    /* "Second" is being set */
    case SETUP_TMSEC:
        /* Cursor position: Lower digit of second */
        fn_LcdCurDisp( 19 );
        break;
    default:
        break;
}
```

```
        switch( ucSetupAlarm )
        {
            /* "Alarm day-of-week" is being set */
            case SETUP_ALWEEK:
                /* Cursor position: First letter of alarm day-of-week */
                fn_LcdCurDisp( 28 );
                break;
            /* "Alarm hour" is being set */
            case SETUP_ALHOUR:
                /* Cursor position: Lower digit of alarm hour */
                fn_LcdCurDisp( 33 );
                break;
            /* "Alarm minute" is being set */
            case SETUP_ALMIN:
                /* Cursor position: Lower digit of alarm minute */
                fn_LcdCurDisp( 36 );
                break;
            default:
                break;
        }

        break;
    /*============================================================*/
    /*  Post-reset display                                        */
    /*============================================================*/
    /*      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+             */
    /*      | |C|P|U| |R|E|S|E|T| | | | | | | | | | |             */
    /*      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+             */
    /*      | | | | | | | | | | | | | | | | | | | | |             */
    /*      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+             */
    /*============================================================*/
    case DISP_RESET:
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( 'C', ucPos++, 1 );
        fn_LcdChrDisp( 'P', ucPos++, 1 );
        fn_LcdChrDisp( 'U', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( 'R', ucPos++, 1 );
        fn_LcdChrDisp( 'E', ucPos++, 1 );
        fn_LcdChrDisp( 'S', ucPos++, 1 );
        fn_LcdChrDisp( 'E', ucPos++, 1 );
        fn_LcdChrDisp( 'T', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
```

```
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        fn_LcdChrDisp( ' ', ucPos++, 1 );
        break;
    default:
        break;
  }
}
```

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 30, 2010	—	First edition issued

NOTES FOR CMOS DEVICES

(1) VOLTAGE APPLICATION WAVEFORM AT INPUT PIN: Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (MAX) and VIH (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (MAX) and VIH (MIN).

(2) HANDLING OF UNUSED INPUT PINS: Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

(3) PRECAUTION AGAINST ESD: A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

(4) STATUS BEFORE INITIALIZATION: Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

(5) POWER ON/OFF SEQUENCE: In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

(6) INPUT OF SIGNAL DURING POWER OFF STATE : Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

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

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

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

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

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

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0