To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

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

Application Note

78K0/Kx2-L

Settings for Low Power Consumption Operation

This document describes the setting contents in order to achieve microcontroller operation with low power consumption using the low power consumption mode and standby function of the 78K0/Kx2-L microcontrollers.

Target devices

78K0/KY2-L microcontroller
78K0/KA2-L microcontroller
78K0/KB2-L microcontroller
78K0/KC2-L microcontroller

CONTENTS

CHAPTER 1 OVERVIEW 3
1.1 Background Required for Low Power Consumption and Features of 78K0/Kx2-L Microcontrollers 4
CHAPTER 2 CLOCK GENERATOR AND STANDBY FUNCTIONS 6
2.1 Clock Generator 6
2.2 Standby Function 11
2.3 Comparison of Total Current of Standby Function 16
2.4 Comparison of Return Times through Interrupt of Standby Function 17
2.5 Return through Reset of Standby Function 20
2.6 Cautions on Clock Generator and Standby Function 24
CHAPTER 3 REGULATOR 25
3.1 Regulator Overview 25
3.2 Register Controlling Regulator 25
3.3 Cautions on Self Programming 26
CHAPTER 4 LOW POWER CONSUMPTION PROGRAM EXAMPLES 27
4.1 Specifications and Overall Flow 27
4.2 Initial Settings and Work Areas 32
4.3 Main Processing 35
4.4 Real-Time Counter Interrupt Servicing 40
4.5 INTP1, INTP4 Interrupt Servicing 43
4.6 Low Voltage Detection Interrupt Servicing 46
CHAPTER 5 RELATED DOCUMENTS 49

Document No. U19612EJ1V0AN00 (1st edition)
Date Published October 2009 N

© NEC Electronics Corporation 2009
Printed in Japan

EEPROM is a trademark of NEC Electronics Corporation.

- The information in this document is current as of July, 2009. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific". The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"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.

"Special": 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, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

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

(Note 2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

(M8E0909E)

CHAPTER 1 OVERVIEW

This application note describes the settings to achieve low power consumption operation of the 78K0/Kx2-L microcontrollers and is intended to give the reader an understanding of how to reduce the power consumption of these microcontrollers.

Chapter 1 provides the background required mainly for low power consumption operation and gives an overview of the functions for low power consumption of the 78K0/Kx2-L microcontrollers.

Target devices:

78K0/KY2-L: μPD78F0550, μPD78F0551, μPD78F0552, μPD78F0555, μPD78F0556, μPD78F0557
78K0/KA2-L: μPD78F0560, μPD78F0561, μPD78F0562, μPD78F0565, μPD78F0566, μPD78F0567
78K0/KB2-L: μPD78F0571, μPD78F0572, μPD78F0573, μPD78F0576, μPD78F0577, μPD78F0578
78K0/KC2-L: μPD78F0581, μPD78F0582, μPD78F0583, μPD78F0586, μPD78F0587, μPD78F0588

1.1 Background Required for Low Power Consumption and Features of 78K0/Kx2-L Microcontrollers

With the spread of portable devices and the expansion of the security devices market, long battery life is increasingly becoming a key requirement. Further, as environmental awareness increases, lowering the power consumption of semiconductors and various other devices is being demanded.

The 78K0/Kx2-L microcontrollers were developed to answer needs to fill the need for products in fields requiring battery operation and maintenance free operation, as well as energy-saving products.

The various functions for realizing low power consumption of the 78K0/Kx2-L microcontrollers are introduced below.

(1) Standby functions

The internal circuits of a microcontroller are operated by inputting a clock signal, and a current passes in the microcontroller during operation. When the clock supply is stopped, the operation of the internal circuits also stops, as does the flow of current.

The standby functions are functions that allow stopping of the clock signal flow by executing the standby instruction by program, or stopping of the operation of the clock oscillator itself.

Power consumption can be efficiently reduced by executing a standby instruction while processing is not being performed, such as during external event wait, in order to set the microcontroller in the standby status (in which the clock signal flow is stopped, or in which the oscillation of the clock oscillator itself is stopped and the microcontroller is in a stopped status).

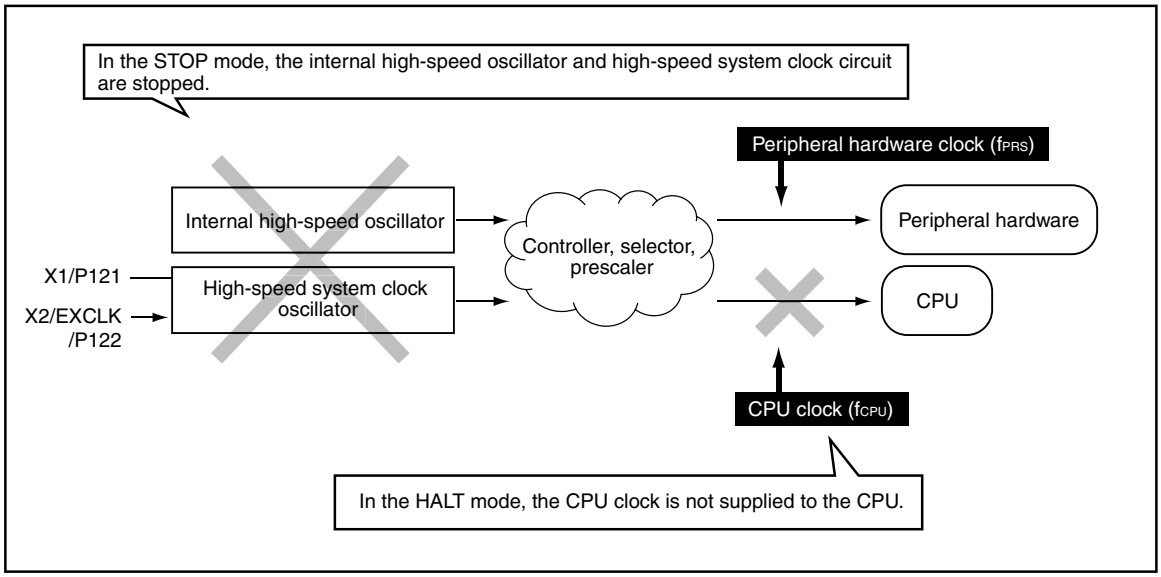
The 78K0/Kx2-L microcontrollers have two standby functions, the HALT mode and the STOP mode.

STOP mode: The microcontroller is made to enter this mode by executing the STOP instruction.
The clock oscillator itself is stopped in this mode.

HALT mode: The microcontroller is made to enter this mode by executing the HALT instruction.
The clock oscillator continues to operate in this mode, but clock signal supply to the CPU is stopped.

Power consumption decreases in the following order: Normal operation mode > HALT mode > STOP mode.

Figure 1-1. Standby Functions and Clock Generator

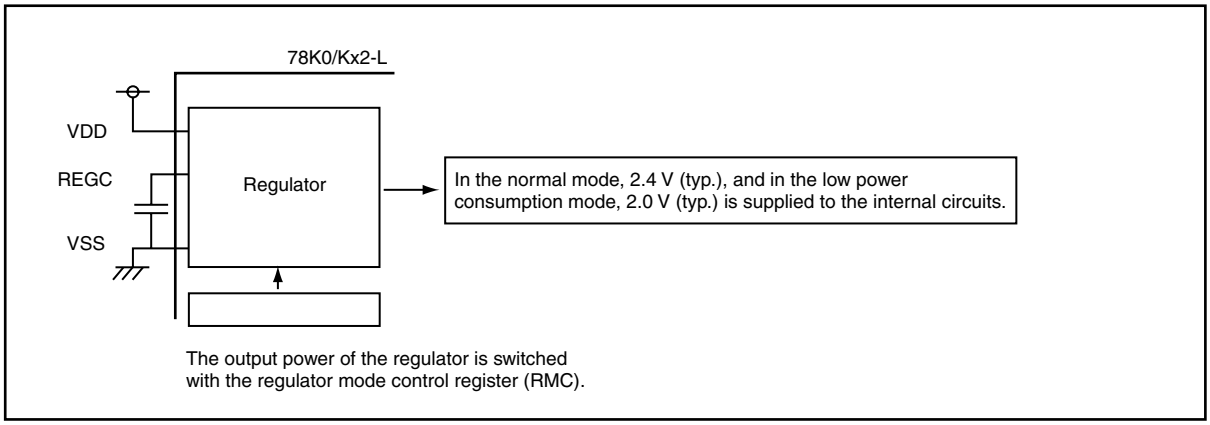


(2) Low power consumption mode through regulator setting

Each 78K0/Kx2-L microcontroller incorporates a regulator circuit for supplying stable power to the circuits inside the microcontroller from the power supply voltage outside the microcontroller. The regulator can control the voltage output inside the microcontroller by setting the regulator mode register (RMC) that controls this regulator circuit.

The regulator supplies 2.4 V (typ.) in the normal mode, and 2.0 V (typ.) in the low power consumption mode. The power consumption is the same even when the power supply voltage is higher than the voltage supplied by the regulator.

Figure 1-2. Regulator



[Column] Handling of unused port pins to further lower power consumption

Setting unused I/O ports to the input status and leaving them unconnected can cause the occurrence of through current, which increases the power consumption. This problem can be avoided by setting port mode registers to output and making them open. In the case of input-only ports, the occurrence of through current can be minimized by either pulling up or pulling down these ports.

CHAPTER 2 CLOCK GENERATOR AND STANDBY FUNCTIONS

The standby function controls the clock generator. The power consumption of the microcontroller is largely influenced by the frequency, in addition to whether the clock is operating or is stopped.

The power consumption can be reduced by having a good understanding regarding the flow of the clock signal from the clock generator, selecting the best frequency clock for the processing, and stopping the clock when it is not needed.

This chapter explains the clock generator and standby functions of the 78K0/Kx2-L microcontrollers.

2.1 Clock Generator

(1) Clock generator after reset release

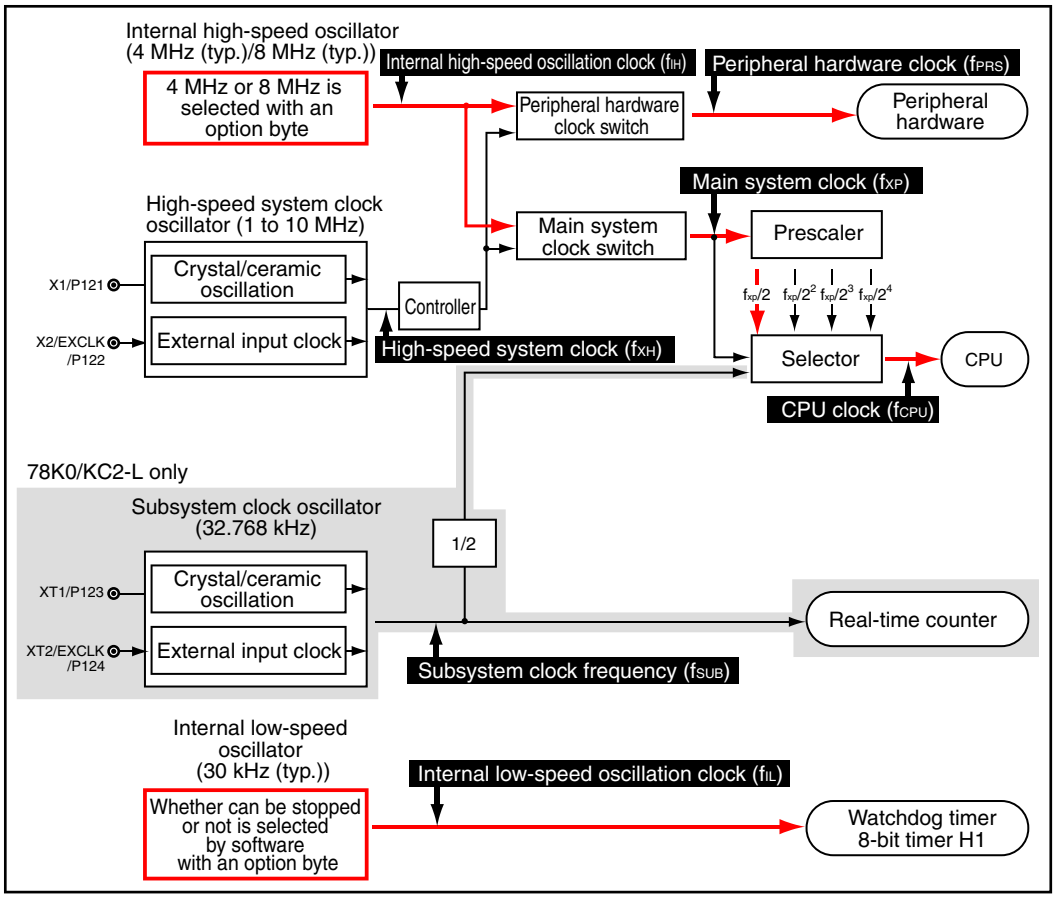
After reset release, the internal high-speed oscillator (8 MHz (typ.) or 4 MHz (typ.) can be selected by option byte setting) and internal low-speed oscillator (30 kHz (typ.)) operate.

The signal of the internal high-speed oscillator is supplied to the main system clock, is divided in half by a prescaler, and the main system clock (f_{XP}/2) is supplied to the CPU clock (f_{CPU}).

The internal high-speed oscillator is used as the clock source of the peripheral hardware.

The internal low-speed oscillator (30 kHz (typ.)) is used as the clock source of the watchdog timer and timer H1.

Figure 2-1. Clock Generator After Reset Release (Red Frames and Lines Indicate Operation)



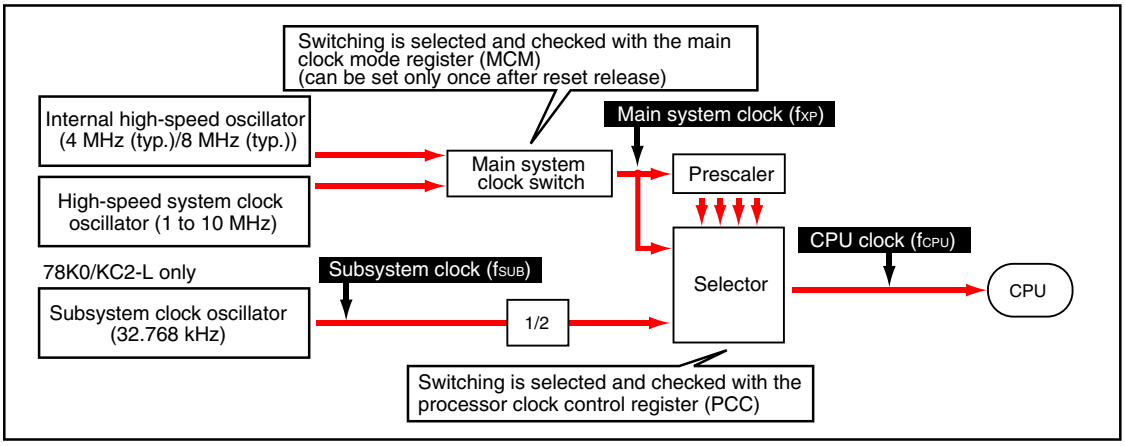
(2) Flow of the CPU clock (f_{CPU}) and switching of the clock source

The CPU clock (f_{CPU}) is a clock signal that is used for reading/executing instructions supplied to the CPU, and reading/writing internal registers and memory.

The supply source of the CPU clock (f_{CPU}) can be selected from the main system clock (no division, or divided by 2, 4, 8, or 16 by prescaler) and the subsystem clock divided by 2 ($f_{SUB}/2$). The supply source of the CPU clock is switched with the processor clock control register (PCC).

The supply source of the main system clock (f_{XP}) can be switched from the internal high-speed oscillator to the high-speed system clock oscillator. Only one switch can be done following reset release. Switching is done by setting the main clock mode register (MCM).

Figure 2-2. Flow of CPU Clock (f_{CPU}) from Each Clock Generator (Red Frames) and Registers for Switching



Main clock mode register (MCM) After reset: 0000 0000B

0	0	0	0	0	XSEL	MCS	MCM0

XSEL	MCM0	Main system clock (f_{XP}) clock source	Peripheral hardware clock (f_{PRS}) clock source
0	0	Internal high-speed oscillation clock (f_{IH}) (default)	Internal high-speed oscillation clock (f_{IH}) (default)
0	1		
1	0		High-speed system clock (f_{XH})
1	1	High-speed system clock (f_{XH})	

MCS	Main system clock (f_{XP}) status (clock source)
0	Internal high-speed oscillation clock (f_{IH}) (default)
1	High-speed system clock (f_{XH})

Processor clock control register (PCC) After reset: 0000 0001B

0	★ XSTART	★ CLS	★ CSS	0	PCC2	PCC1	PCC0

★ 78K0/KC2-L only

CSS	PCC2	PCC1	PCC0	CPU clock (f_{CPU}) clock source
1	0	0	0	Main system clock, undivided (f_{XP})
	0	0	1	Main system clock divided by 2 ($f_{XP}/2$) (default)
	0	1	0	Main system clock divided by 2^2 ($f_{XP}/2^2$)
	0	1	1	Main system clock divided by 2^3 ($f_{XP}/2^3$)
	1	0	0	Main system clock divided by 2^4 ($f_{XP}/2^4$)

Other than above: Setting prohibited

CSS	PCC2	PCC1	PCC0	CPU clock (f_{CPU}) clock source
0	0	0	0	Subsystem clock divided by 2 ($f_{SUB}/2$)
	0	0	1	
	0	1	0	
	0	1	1	
	1	0	0	

Other than above: Setting prohibited

CLS	CPU clock (f_{CPU}) status (clock source)
0	Main system clock (f_{XP}) (default)
1	Subsystem clock divided by 2 ($f_{SUB}/2$)

When the supply source of the main system clock (f_{XP}) or CPU clock (f_{CPU}) is switched, the clock prior to the switch continues to be supplied for a few clocks.

In the case of the main system clock, whether the switch is complete can be checked with the MCS bit of the main clock mode register (MCM).

If the CPU clock (f_{CPU}) has been switched from the main system clock (f_{XP}) to the subsystem clock divided by 2 ($f_{SUB}/2$), or vice-versa, whether the switch is complete can be checked with the CLS bit of the processor clock control register (PCC).

If the division ratio of the main system clock has been switched by changing the setting of the processor clock control register (PCC), a wait period equal to the times listed below is required.

Table 2-1. Maximum Time Required for Switching CPU Clock When Division Ratio of Main System Clock Has Been Switched

Division Ratio Before Switching	Division Ratio After Switching				
	Undivided	1/2	$1/2^2$	$1/2^3$	$1/2^4$
Undivided		16 clocks	16 clocks	16 clocks	16 clocks
1/2	8 clocks		8 clocks	8 clocks	8 clocks
$1/2^2$	4 clocks	4 clocks		4 clocks	4 clocks
$1/2^3$	2 clocks	2 clocks	2 clocks		2 clocks
$1/2^4$	1 clock	1 clock	1 clock	1 clock	

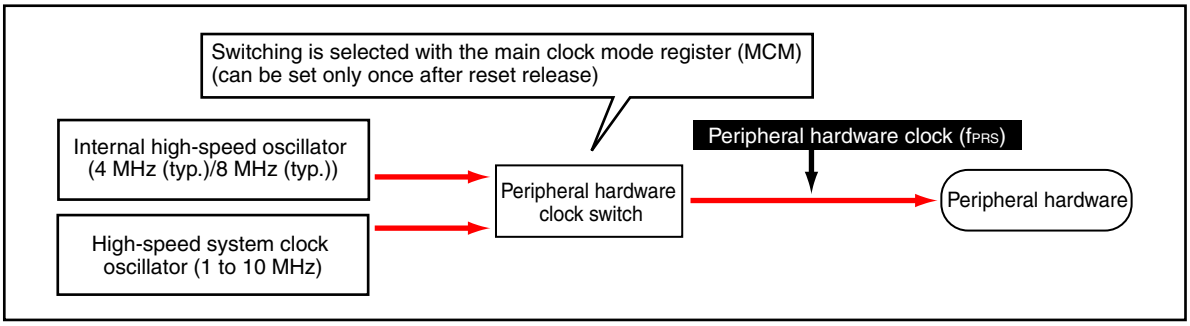
Remark The number of clocks is the number of CPU clocks (f_{CPU}) before switching.

(3) Flow of the peripheral hardware clock and switching of the clock source

The peripheral hardware clock is supplied to the peripheral hardware, which uses it as the operating clock.

The supply source of the peripheral hardware clock can be switched from the internal high-speed oscillator to the high-speed system clock oscillator. Only one switch can be done following reset release. Switching is done by setting the main clock mode register (MCM). (For details about the main clock mode register (MCM), refer to **Figure 2-2**.)

Figure 2-3. Flow of Peripheral Hardware Clock (f_{PRS}) from Each Clock Generator (Red Lines)



(4) Stopping and starting clock generators

In the case of clock generators whose signals are not supplied anywhere, or if the function at the supply destination is not used, the clock generator can be stopped (The internal low-speed oscillator can be stopped if "Can be stopped by software" has been set with an option byte). Before a clock generator is stopped, whether its signal is supplied to currently operating functions (CPU, peripheral hardware) must be checked.

When the operation of a clock generator that was stopped is started again, before supplying that signal to the CPU clock, peripheral hardware clock, and real-time counter clock, the lapse of the oscillation stabilization time must be waited for.

<1> Internal high-speed oscillator and internal low-speed oscillator

The oscillation of the internal oscillators is started or stopped with the internal oscillation mode register (RCM). Oscillation stabilization of the internal high-speed oscillator can be checked with the RSTS bit.

Figure 2-4. Internal Oscillation Mode Register

Internal oscillation mode register (RCM) After reset: 1000 0000B

RSTS	0	0	0	0	0	LSRSTOP	RSTOP

LSRSTOP	Internal low-speed oscillator oscillating/stopped
0	Internal low-speed oscillator oscillating (default)
1	Internal low-speed oscillator stopped

RSTOP	Internal high-speed oscillator oscillating/stopped
0	Internal high-speed oscillator oscillating (default)
1	Internal high-speed oscillator stopped

RSTS	Status of internal high-speed oscillator
0	Waiting for accuracy stabilization of internal high-speed oscillator
1	Stability operating of internal high-speed oscillator (default)

Caution The RSTS bit is 0 immediately after a reset but changes to 1 after internal high-speed oscillator has been stabilized.

<2> High-speed system clock oscillator

The oscillation of the high-speed system clock oscillator is started or stopped with the MSTOP bit of the main OSC control register (MOC).

In the case of the X1 oscillation mode, the oscillation stabilization time wait can be measured with the X1 oscillation stabilization time counter. The stabilization time is set to the oscillation stabilization time select register (OSTS), and whether the time set with the oscillation stabilization time counter status register (OSTC) has elapsed can be checked. Set a sufficient length of time according to the resonator that is used. (The X1 oscillation stabilization time counter measures only the time that has been set, and does not check whether the X1 oscillation is stable.)

The oscillation stabilization time select register (OSTS) is set to the longest measurement time (2^{16}/fx) after reset is released.

Figure 2-5. Main OSC Control Register

Main OSC control register (MOC) After reset: 1000 0000B

MSTOP	0	0	0	0	0	0	0

MSTOP	Control of high-speed system clock operation	
	X1 oscillation mode	External clock input mode
0	X1 oscillator operating	External clock from EXCLK pin is enabled
1	X1 oscillator stopped (default)	External clock from EXCLK pin is disabled (default)

Figure 2-6. Registers of X1 Oscillation Stabilization Time Counter

Oscillation stabilization time select register (OSTS) After reset: 0000 0101B

0	0	0	0	0	OSTS2	OSTS1	OSTS0

OSTS2	OSTS1	OSTS0	Oscillation stabilization time selection	f_X = 10 MHz
0	0	1	$2^{11}/f_X$	204.8 μs
0	1	0	$2^{13}/f_X$	819.2 μs
0	1	1	$2^{14}/f_X$	1.64 ms
1	0	0	$2^{15}/f_X$	3.27 ms
1	0	1	$2^{16}/f_X$ (default)	6.55 ms

Other than above: Setting prohibited

Oscillation stabilization time counter status register (OSTC) After reset: 0000 0000B

0	0	0	MOST11	MOST13	MOST14	MOST15	MOST16

MOST11	MOST13	MOST14	MOST15	MOST16	Oscillation stabilization time selection	f_X = 10 MHz
1	0	0	0	0	$2^{11}/f_X$ min.	204.8 μs min.
1	1	0	0	0	$2^{13}/f_X$ min.	819.2 μs min.
1	1	1	0	0	$2^{14}/f_X$ min.	1.64 ms min.
1	1	1	1	0	$2^{15}/f_X$ min.	3.27 ms min.
1	1	1	1	1	$2^{16}/f_X$ min.	6.55 ms min.

→ Becomes 1 with the lapse of time

<3> Subsystem clock oscillator (78K0/KC2-L only)

The oscillation of the subsystem clock oscillator is started or stopped with each bit of the processor clock control register (PCC) and clock operation mode select register (OSCCTL).

Following the start of operation in the XT1 oscillation mode, the stabilization time must be waited for according to the resonator that is used.

Figure 2-7. Registers of Subsystem Clock Oscillator

Processor clock control register (PCC) After reset: 0000 0001B

0	★ XSTART	★ CLS	★ CSS	0	PCC2	PCC1	PCC0

★ 78K0/KC2-L only

Clock operation mode select register (OSCCTL) After reset: 0000 0000B

EXCLK	OSCSEL	★ EXCLKS	★ OSCSELS	0	★ RSWOSC	★ AMPHXT	0

★ 78K0/KC2-L only

XSTART	EXCLKS	OSCSELS	Subsystem clock operating mode
0	0	1	XT1 oscillation mode (a crystal or ceramic resonator must be connected to P123/XT1 and P124/XT2)
1	×	×	
0	1	1	External clock input mode (external clock must be input to P124/EXCLKS)

To stop the subsystem clock, set OSCSELS to 0 (default = stopped) ×: don't care

RSWOSC	AMPHXT	XT1 oscillator oscillation mode selection
0	0	Low power consumption oscillation (default)
0	1	Normal oscillation
1	×	Ultra-low power consumption oscillation

× : don't care

CLS	CPU clock (f_{CPU}) status (clock source)
0	Operates with main system clock (f_{XP})
1	Operates with subsystem clock (f_{SUB})

2.2 Standby Function

Using the standby function, the power consumption can be reduced by controlling the clock at an arbitrary timing of the program, by executing the HALT instruction or the STOP instruction to enter the HALT mode or the STOP mode, respectively. The normal operation mode can be returned to from the HALT mode or STOP mode by issuing an unmasked interrupt request signal or reset signal.

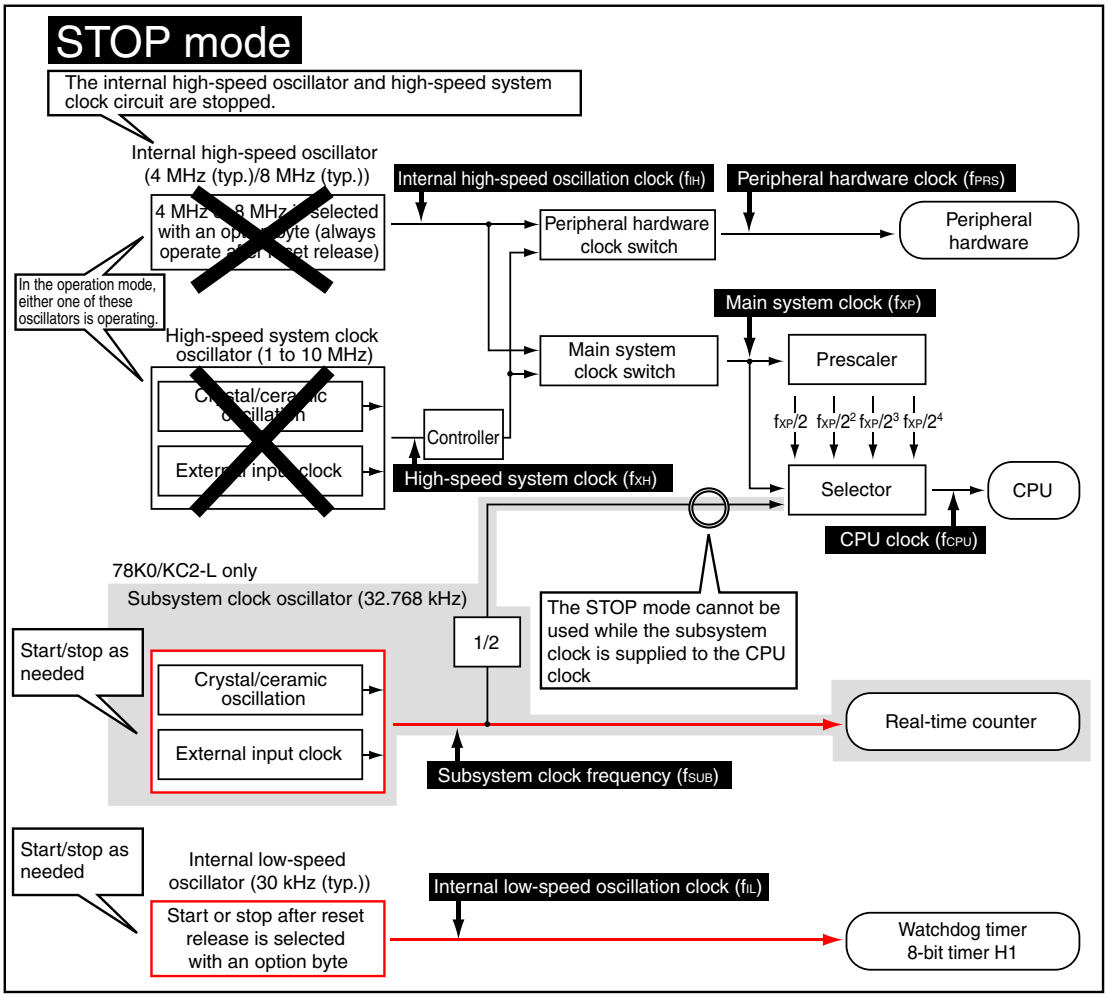
The HALT mode does not reduce the operating current as much as the STOP mode, but since the clock is oscillating, the time to return to the normal operation mode is shorter. Compared to the HALT mode, the STOP mode reduces the operating current to a greater extent, but the oscillator of the clock stops, so the time to return to the normal operation mode takes longer.

Which mode to use for standby is determined based on the delay concomitant to the frequency of intermittent program operation and the power consumption requirements. For a comparison of operating currents, refer to **2.3 Comparison of Total Current of Standby Function**, and for a comparison of the return times, refer to **2.4 Comparison of Return Times through Interrupt of Standby Function**.

(1) STOP mode

When the STOP instruction is executed, the internal high-speed oscillator and the high-speed system clock oscillator stop. The operation continues if the subsystem clock oscillator and the internal low-speed oscillator are operating.

Figure 2-8. Clock Generator in STOP Mode



<1> CPU clock (f_{CPU}) conditions

The STOP mode can be entered only when the main system clock (f_{XP}) has been selected as the CPU clock (f_{CPU}). The subsystem clock (f_{SUB}: 78K0/KC2-L only) does not stop even when the STOP instruction is executed, so the STOP mode cannot be entered when the CPU clock is selected.

<2> CPU and peripheral hardware status

- The CPU is stopped because the internal high-speed oscillator (f_{IH}) and the high-speed system clock oscillator (f_{XH}) are stopped.
- The functions of the peripheral hardware that uses the peripheral hardware clock (f_{PRS}) as the source are stopped. However, the functions that use the external clock input from a pin, the internal low-speed oscillation clock (f_{IL}), or subsystem clock (f_{SUB}) as the source in the peripheral hardware remain in the same status as immediately before the STOP instruction was executed (if they were operating, they continue to operate). Before executing the STOP instruction, the peripheral hardware that will use the peripheral hardware clock (f_{PRS}) as the source must be stopped if it is operating.
- The various registers of the CPU, RAM, and peripheral hardware maintain their status before the execution of the STOP instruction.
- If “Internal low-speed oscillator can be stopped by software” is selected by option byte setting, clock supply from the internal low-speed oscillator to the watchdog timer stops.

Table 2-2. Peripheral Hardware in STOP Mode

Peripheral Hardware		Status in STOP Mode
16-bit timer/event counter 00		Operation stopped
8-bit timer/event counter	50	Operable only when TI50 is selected as the count clock
	51	Operable only when TI51 is selected as the count clock
8-bit timer	H0	Operable only when TM50 output is selected as the count clock during 8-bit timer/event counter 50 operation
	H1	Operable only when f_{IL}, $f_{IL}/2^6$, or $f_{IL}/2^{15}$ is selected as the count clock
Real-time counter		Operable
Watchdog timer		Operable. Clock supply to watchdog timer stops when “Internal low-speed oscillator can be stopped by software” is set by option byte.
Clock output		Operable only when subsystem clock is selected as the count clock
A/D converter		Operation stopped
Operational amplifiers 0, 1		Operation stopped
Serial interface	UART6	Operable only when TM50 output is selected as the serial clock during 8-bit timer/event counter 50 operation
	CSI10, CSI11	Operable only when external clock is selected as the serial clock
	IICA	Wakeup (interrupt signal generation) by address match operable
Key interrupt		Operable
Power-on-clear function		
Low-voltage detection function		
External interrupt		

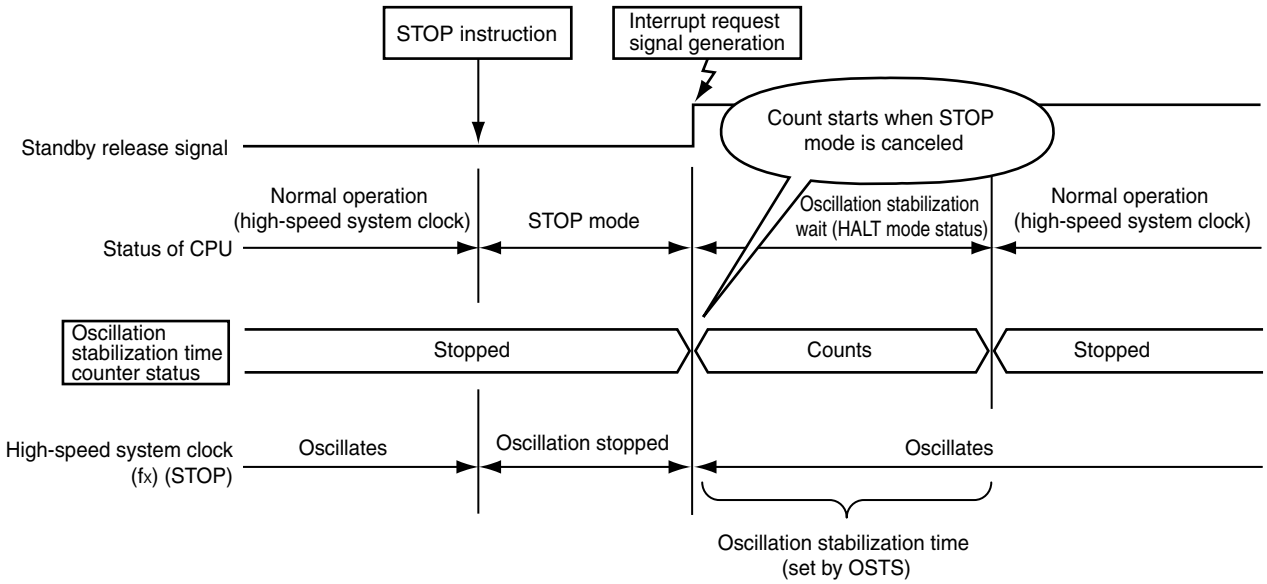
<3> Operation of oscillation stabilization time counter upon return from STOP mode

The clock generator includes a counter for measuring the stabilization time of the X1 resonator. If the X1 resonator of the high-speed system clock is oscillated, the oscillation stabilization time can be measured by setting the oscillation stabilization time select register (OSTS) and reading the oscillation stabilization time counter status register (OSTC).

The OSTC register starts counting from when the X1 oscillator starts oscillating (MSTOP = 0), while the CPU clock (f_{CPU}) is supplied from either the internal high-speed oscillator or subsystem clock oscillator.

However, counting starts immediately in the following case only: after return from the STOP mode when the STOP instruction is executed and return is effected upon occurrence of an unmasked interrupt request signal, while the CPU clock (f_{CPU}) is supplied from the X1 resonator of the high-speed system clock. While counting is executed, the operating status is the HALT status during which the clock source is not supplied to the CPU clock (f_{CPU}), and after the time set to the OSTS register has been measured, the operation mode (normal status) is returned to. Therefore, set an appropriate value to the oscillation stabilization time select register (OSTS) before executing the STOP instruction.

Figure 2-9. Operation of X1 Oscillation Stabilization Time Counter When STOP Instruction Is Canceled



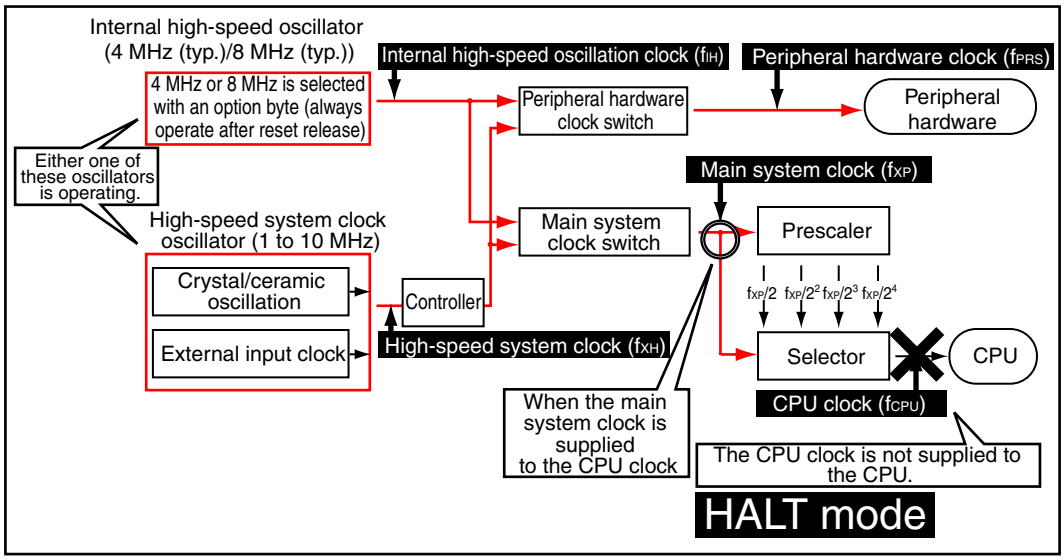
(2) HALT mode

When the HALT instruction is executed, the operation continues if the various oscillators are operating, and the CPU clock (f_{CPU}) supply to the CPU is stopped.

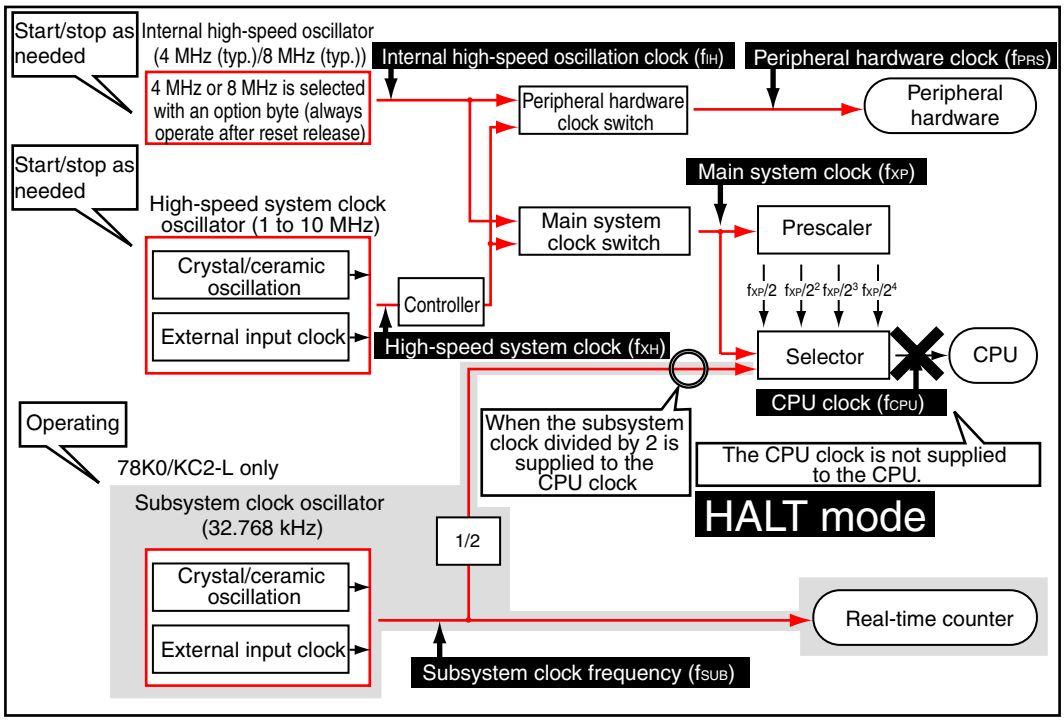
The HALT mode can be entered even when either the main system clock (f_{XP}) or the subsystem clock divided by 2 ($f_{SUB}/2$) is selected as the CPU clock (f_{CPU}).

Figure 2-10. Clock Generator in HALT Mode (Internal Low-Speed Oscillator Is Omitted)

When the main system clock is selected as the CPU clock



When the subsystem clock divided by 2 is selected as the CPU clock (78K0/KC2-L only)



<1> CPU clock conditions

The HALT mode can be entered even when either the main system clock (f_{XP}) or the subsystem clock divided by 2 ($f_{SUB}/2$) is selected as the CPU clock (f_{CPU}).

When the subsystem clock divided by 2 ($f_{SUB}/2$) is selected as the CPU clock (f_{CPU}), the power consumption in the HALT mode can be further reduced by stopping both the internal high-speed oscillator and the high-speed system oscillator (refer to **Table 2-5**).

<2> CPU and peripheral hardware status

- The CPU is stopped because the CPU clock (f_{CPU}) is not supplied to the CPU.
- The status before the HALT instruction was executed is maintained because the peripheral hardware clock (f_{PRS}) is supplied to the peripheral hardware. (The peripheral hardware continues to operate if it was operating.)
- The various registers of the CPU, RAM, and peripheral hardware maintain their status before the execution of the HALT instruction.
- If "Internal low-speed oscillator can be stopped by software" is selected by option byte setting, clock supply from the internal low-speed oscillator to the watchdog timer stops.

(3) Return through unmasked interrupt (common to STOP mode and HALT mode)

If, upon return through an unmasked interrupt request signal, the interrupt enable flag (IE) is enabled (IE = 1), interrupt servicing is executed, and if it is disabled (IE = 0), interrupt servicing is not executed and the next instruction is executed instead. In this case, the interrupt request signal is not automatically cleared and is held pending instead, so that interrupt is serviced when it is enabled (IE = 1). (An inadvertent program loop occurs if there is no program at the address that corresponds to the interrupt vector.)

To not execute interrupt servicing upon return from the STOP mode or HALT mode through an unmasked interrupt, the interrupt request flag must be cleared before the interrupt is enabled (IE = 1) after the return.

Table 2-3. Operation upon Return in Response to Interrupt Request (STOP and HALT Modes)

Release Source	MKXX (Mask Flag)	PRXXX (Priority Specification Flag)	IE	ISP	Operation
Maskable interrupt request	0 (Enable)	0 (High)	0	×	Next address execution (interrupt request flag held)
			1	×	Interrupt servicing execution
		1 (Low)	0	1	Next address execution (interrupt request flag held)
			×	0	
			1	1	Interrupt servicing execution
	1 (Disable)	×	×	×	STOP and HALT modes held
Reset	–	–	×	×	Reset processing

[Column] Use of HALT mode during event wait of peripheral hardware

If the occurrence of peripheral hardware events such as conversion wait of an A/D converter is being detected through software polling, use of the HALT mode instead should be considered. The return time from the HALT mode through an interrupt is sometimes faster than detection through polling. Moreover, since the microcontroller continues to execute instructions during polling, placing the microcontroller be in standby in the HALT mode reduces the power consumption.

2.3 Comparison of Total Current of Standby Function

The following tables list the total current (by regulator setting) that flows in the internal power supplies (V_{DD}, AV_{REF}) during normal operation, in the HALT mode, and in the STOP mode, under various conditions.

Table 2-4. CPU Clock (f_{CPU}) Supplied from Internal High-Speed Oscillation Clock (f_{IH}) or High-Speed System Clock (f_{XH})

Operating Conditions	Operation Mode (Normal Operation)	HALT Mode	STOP Mode
f_{IH} = 4 MHz (Internal high-speed oscillation clock), V_{DD} = 3.0 V RMC = 56H (Low power consumption mode)	0.5 mA (TYP.), 1.4 mA (MAX.)	0.2 mA (TYP.), 0.5 mA (MAX.)	0.3 μA (TYP.), 5.5 μA (MAX.) (Clock generator is stopped, so frequency is not included in conditions)
f_{IH} = 8 MHz (Internal high-speed oscillation clock), V_{DD} = 5.0 V RMC = 00H	1.3 mA (TYP.), 2.5 mA (MAX.)	0.3 mA (TYP.), 1.2 mA (MAX.)	–
f_{XH} = 10 MHz (External input clock), V_{DD} = 5.0 V RMC = 00H	1.6 mA (TYP.), 2.8 mA (MAX.)	0.4 mA (TYP.), 1.3 mA (MAX.)	–
f_{XH} = 10 MHz (Crystal/ceramic oscillation), V_{DD} = 5.0 V RMC = 00H	2.3 mA (TYP.), 3.9 mA (MAX.)	1.0 mA (TYP.), 2.4 mA (MAX.)	–

Table 2-5. CPU Clock (f_{CPU}) Supplied from Subsystem Clock Divided by 2 (f_{SUB}/2) (78K0/KC2-L Only)

Operating Conditions	Operation Mode (Normal Operation)	HALT Mode	STOP Mode
f_{SUB} = 32.768 MHz (Crystal/ceramic oscillation), V_{DD} = 3.0 V RMC = 56H (Low power consumption mode)	3 μA (TYP.), 9.7 μA (MAX.)	0.8 μA (TYP.), 6.7 μA (MAX.)	0.3 μA (TYP.), 5.5 μA (MAX.) (Clock generator is stopped, so frequency is not included in conditions)

Cautions 1. Tables 2-4 and 2-5 have been excerpted from CHAPTER 28 ELECTRICAL SPECIFICATIONS of the 78K0/Kx2-L User's Manual.

2. The current value of Tables 2-4 and 2-5 includes the input leakage current flowing when the level of the input pin is fixed to V_{DD} or V_{SS}. However, the current flowing into the pull-up resistors, the pull-down resistors, and the output current of the port are not included.

3. The current value of the normal mode and HALT mode does not include the current flowing into the oscillator other than the circuit which generates the clock supplied to CPU. It does not include the current flowing into the LVI circuit, A/D converter, operational amplifier, watchdog timer, real-time counter, and 8-bit timer H1 (when using the 30 kHz internal low-speed oscillation clock as the count clock).

4. The current value of the STOP mode does not include the current flowing into the LVI circuit, A/D converter, operational amplifier, watchdog timer, real-time counter, and 8-bit timer H1 (when using the 30 kHz internal low-speed oscillation clock as the count clock).

5. The current value of the STOP mode is for the conditions of V_{DD} = 3.0 V, RMC = 56H. Clock generator is stopped, so frequency is not included in conditions.

2.4 Comparison of Return Times through Interrupt of Standby Function

In the case of return through an unmasked maskable interrupt request signal, the return time differs between return from the HALT mode and return from the STOP mode. This is because in the HALT status, the internal high-speed oscillator and the high-speed system clock oscillator operate, while in the STOP status, they are stopped.

Upon occurrence of an interrupt request signal, the standby release signal (internal signal: H = release status) is released, and return from each mode is effected.

In the case of return from the STOP mode, the return time differs depending on which clock (internal high-speed oscillator (f_{IH}), high-speed system clock oscillator (both f_X and f_{EXCLK}), subsystem clock oscillator divided by 2 ($f_{SUB}/2$)) is supplied to the CPU clock (f_{CPU}). (The return time is longest when the X1 resonator (f_X) of the high-speed system clock oscillator is supplied.) In the case of return from the HALT mode, the return time is the same regardless of the clock that is supplied to the CPU clock (f_{CPU}).

Unless specified otherwise, the values in the table below are for RMC = 00H (fixed 2.4 V supply).

(1) When the CPU clock (f_{CPU}) is supplied from the internal high-speed oscillation clock (f_{IH})

Figure 2-11. Return through Interrupt Request Signal ($f_{CPU} \leftarrow f_{IH}$)

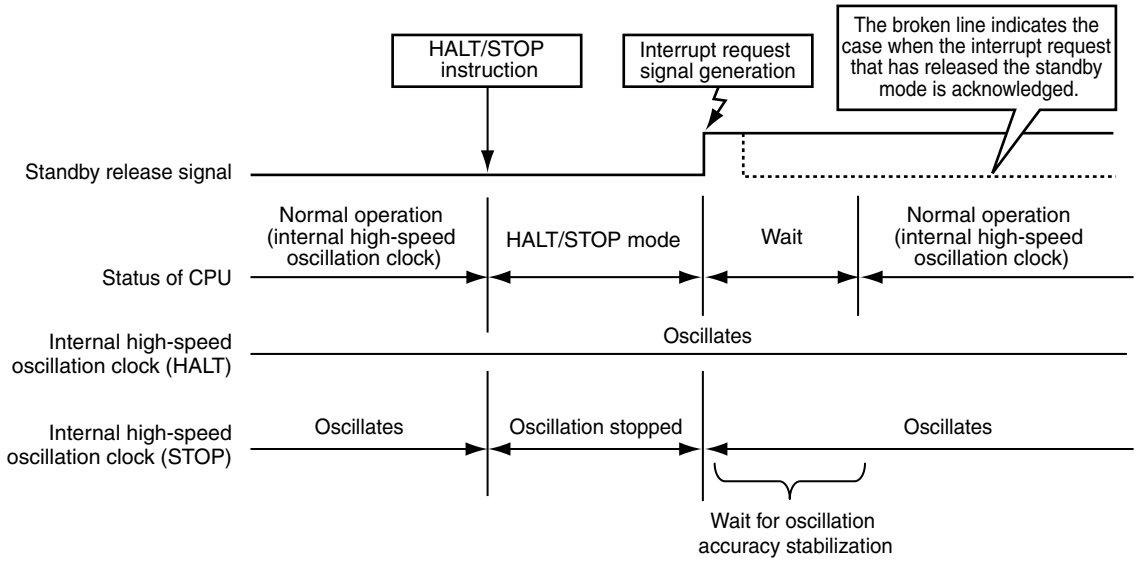


Table 2-6. Return Times in Case of Return from HALT and STOP Modes ($f_{CPU} \leftarrow f_{IH}$)

Source		HALT Mode (f_{CPU} = 8 MHz)	STOP Mode (f_{CPU} = 8 MHz)
Wait	When vectored interrupt servicing is carried out	11 or 12 clocks (1.375 to 1.5 μs)	17 or 18 clocks (2.125 to 2.25 μs)
	When vectored interrupt servicing is not carried out	4 or 5 clocks (1.375 to 1.5 μs)	11 or 12 clocks (1.375 to 1.5 μs)
Wait for oscillation accuracy stabilization		–	102 to 407 μs (RMC = 00H) 120 to 481 μs (RMC = 56H)

(2) When the CPU clock (f_{CPU}) is supplied from the X1 resonator (f_X) of the high-speed system clock

If the CPU clock (f_{CPU}) is supplied from the X1 crystal/ceramic resonator and return from the STOP mode is effected, the system waits for the oscillation stabilization time set to the OSTS register by the user according to the resonator. During this time, the microcontroller is in the HALT mode status, in which no clock is supplied to the CPU clock (f_{CPU}).

Figure 2-12. Return through Interrupt Request Signal ($f_{CPU} \leftarrow f_{XH} \leftarrow f_X$)

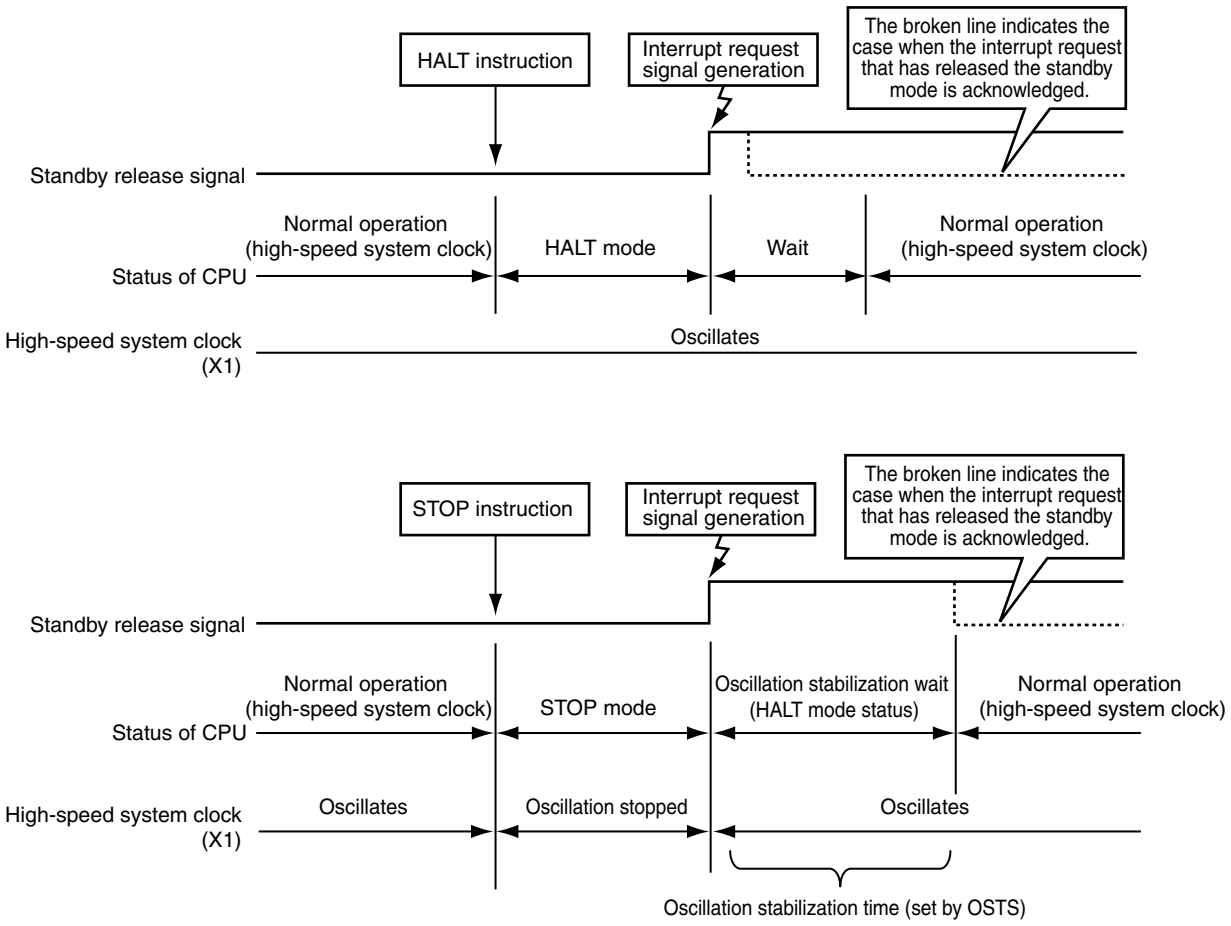


Table 2-7. Return Times in Case of Return from HALT and STOP Modes ($f_{CPU} \leftarrow f_{XH} \leftarrow f_X$)

Source		HALT Mode (f_{CPU} = 10 MHz)	STOP Mode (f_{CPU} = 10 MHz)
Wait	When vectored interrupt servicing is carried out	11 or 12 clocks (1.1 to 1.2 μs)	–
	When vectored interrupt servicing is not carried out	4 or 5 clocks (0.4 to 0.5 μs)	–
Oscillation stabilization time (set by OSTS)		–	Wait time set to OSTS register $2^{11}/f_X$ (204.8 μs) $2^{13}/f_X$ (819.2 μs) $2^{14}/f_X$ (1.64 ms) $2^{15}/f_X$ (3.27 ms) $2^{16}/f_X$ (6.55 ms)

(3) When the CPU clock (f_{CPU}) is supplied from the X2 external clock (f_{EXCLK}) of the high-speed system clock

Figure 2-13. Return through Interrupt Request Signal ($f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$)

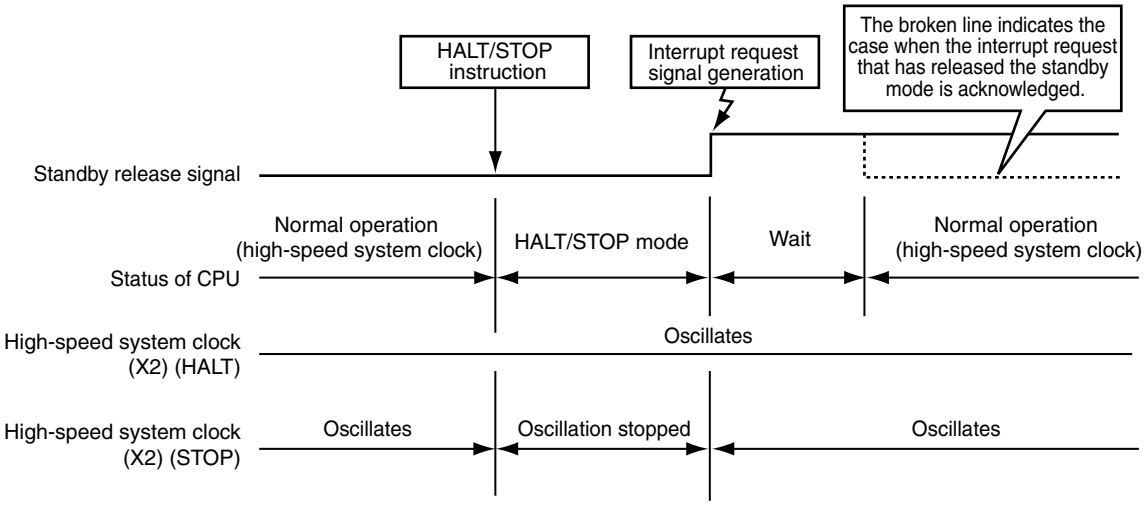


Table 2-8. Return Times in Case of Return from HALT and STOP Modes ($f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$)

Source		HALT Mode (f_{CPU} = 8 MHz)	STOP Mode (f_{CPU} = 8 MHz)
Wait	When vectored interrupt servicing is carried out	11 or 12 clocks (1.375 to 1.5 μs)	17 or 18 clocks (2.125 to 2.25 μs)
	When vectored interrupt servicing is not carried out	4 or 5 clocks (1.375 to 1.5 μs)	11 or 12 clocks (1.375 to 1.5 μs)

(4) When the CPU clock (f_{CPU}) is supplied from the subsystem clock ($f_{SUB}/2 \leftarrow f_{XT}$, f_{EXCLKS})

Only the HALT instruction is enabled for both the XT1 input (f_{XT}: crystal resonator/ceramic resonator) and the XT2 input (f_{EXCLKS}: external clock). (Execution of the STOP instruction is prohibited.) The return timing chart and the return time are the same as in the case of return from the HALT instruction, sections (1) to (3).

2.5 Return through Reset of Standby Function

In the case of return through reset signal generation, all the clock generators stop during the reset period, so the return time is the same in the STOP status and in the HALT status.

(1) Return through reset signal

In the case of return through reset signal, all the clock generators stop during the reset period, so the return time is the same in the HALT mode and in the STOP mode.

<1> When the CPU clock (f_{CPU}) is supplied from the internal high-speed oscillation clock (f_{IH})

Figure 2-14. Return through Reset Signal ($f_{CPU} \leftarrow f_{IH}$)

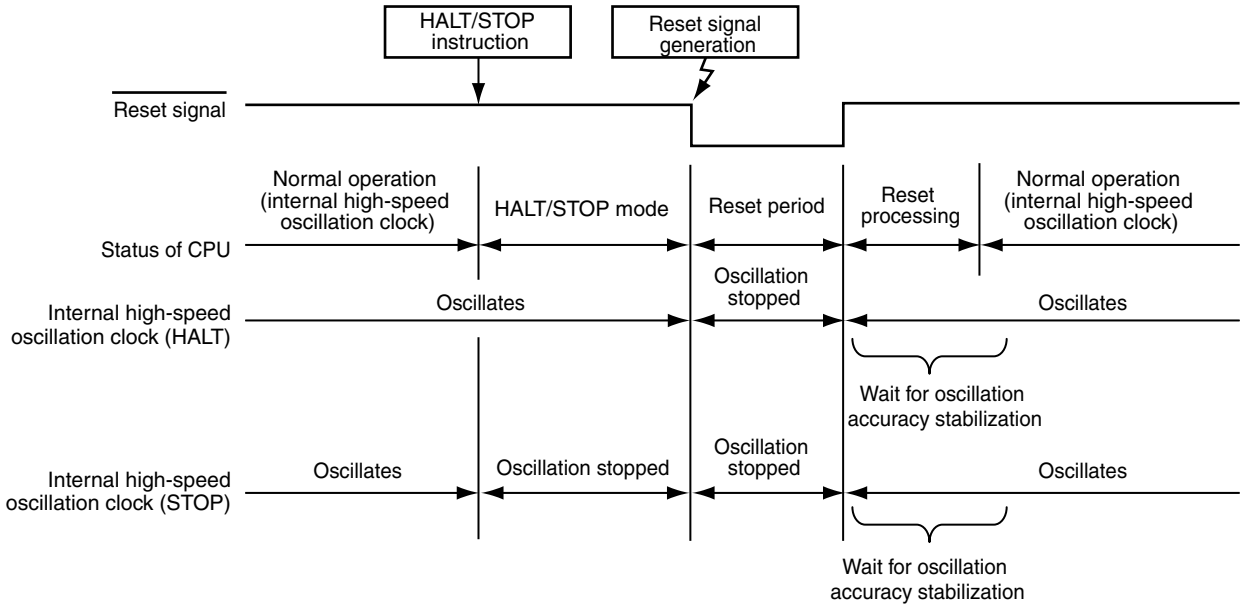


Table 2-9. Return Times in Case of Return from HALT and STOP Modes ($f_{CPU} \leftarrow f_{IH}$)

Source	HALT Mode/STOP Mode
Reset processing	12 to 51 μs
Wait for oscillation accuracy stabilization	102 to 407 μs

<2> When the CPU clock (f_{CPU}) is supplied from the X1 resonator (f_X) of the high-speed system clock

Figure 2-15. Return through Reset Signal ($f_{CPU} \leftarrow f_{XH} \leftarrow f_X$)

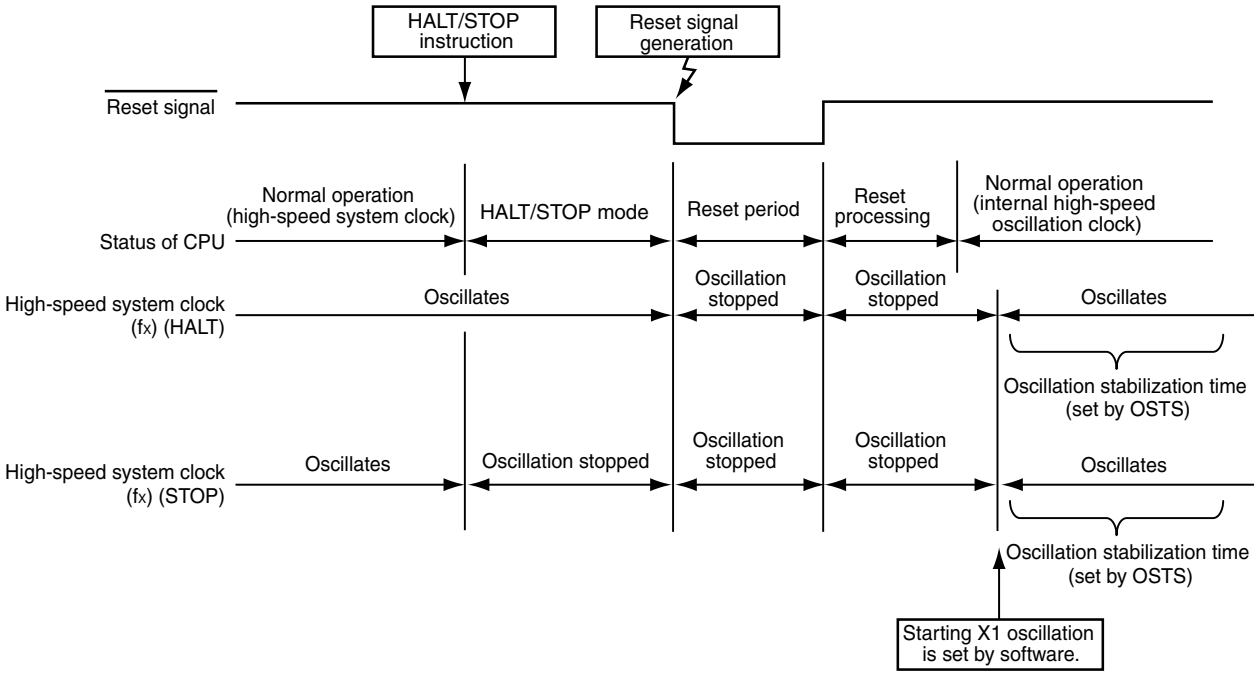


Table 2-10. Return Times in Case of Return from HALT and STOP Modes ($f_{CPU} \leftarrow f_{XH} \leftarrow f_X$)

Source	HALT Mode/STOP Mode
Reset processing	12 to 51 μs
Oscillation stabilization time (set by OSTS)	Wait time set to OSTS register (f_{CPU} = 10 MHz) $2^{11}/f_X$ (204.8 μs) $2^{13}/f_X$ (819.2 μs) $2^{14}/f_X$ (1.64 ms) $2^{15}/f_X$ (3.27 ms) $2^{16}/f_X$ (6.55 ms)

<3> When the CPU clock (f_{CPU}) is supplied from the X2 external clock (f_{EXCLK}) of the high-speed system clock

Figure 2-16. Return through Reset Signal ($f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$)

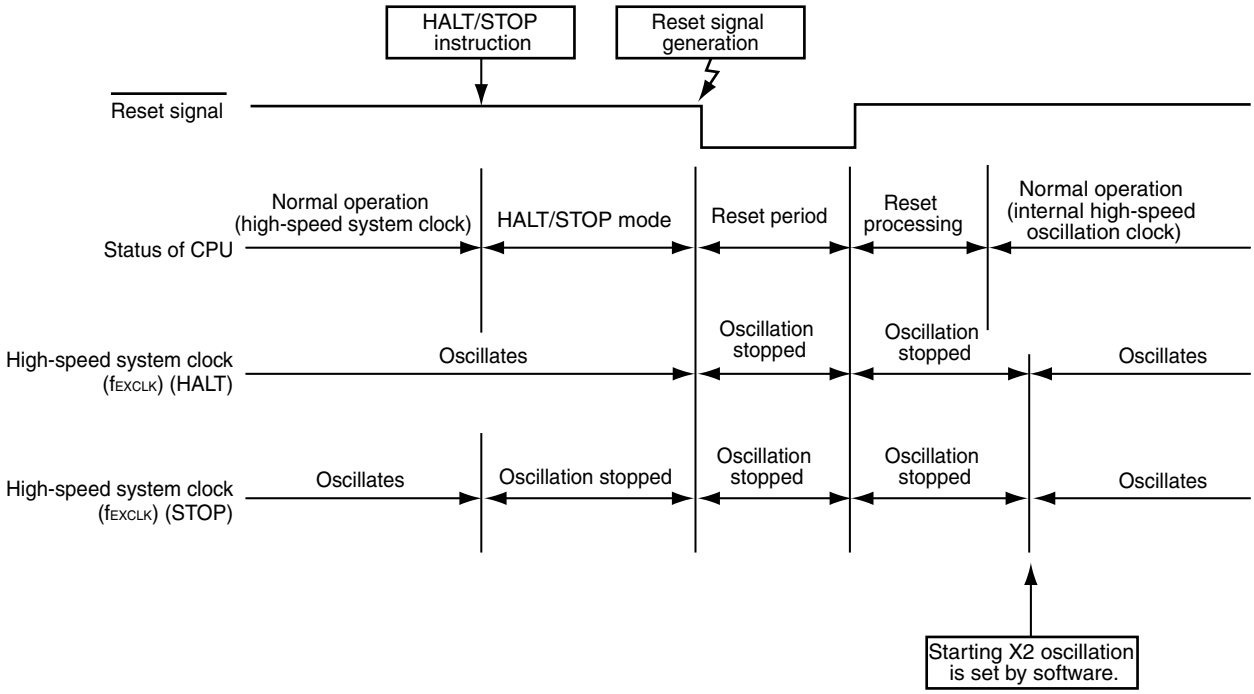


Table 2-11. Return Times in Case of Return from HALT and STOP Modes ($f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$)

Source	HALT Mode/STOP Mode
Reset processing	12 to 51 μs

<4> When the CPU clock (f_{CPU}) is supplied from the subsystem clock divided by 2 ($f_{SUB}/2$)
Only the HALT instruction is enabled for both the XT1 input (f_{XT}: crystal resonator/ceramic resonator) and the XT2 input (f_{EXCLKS}: external clock). (Execution of the STOP instruction is prohibited.)
The oscillation stabilization time of the XT1 input is the wait time measured by the user with a timer, etc., according to the resonator that is used.

Figure 2-17. Return through Reset Signal ($f_{CPU} \leftarrow f_{SUB}/2$)

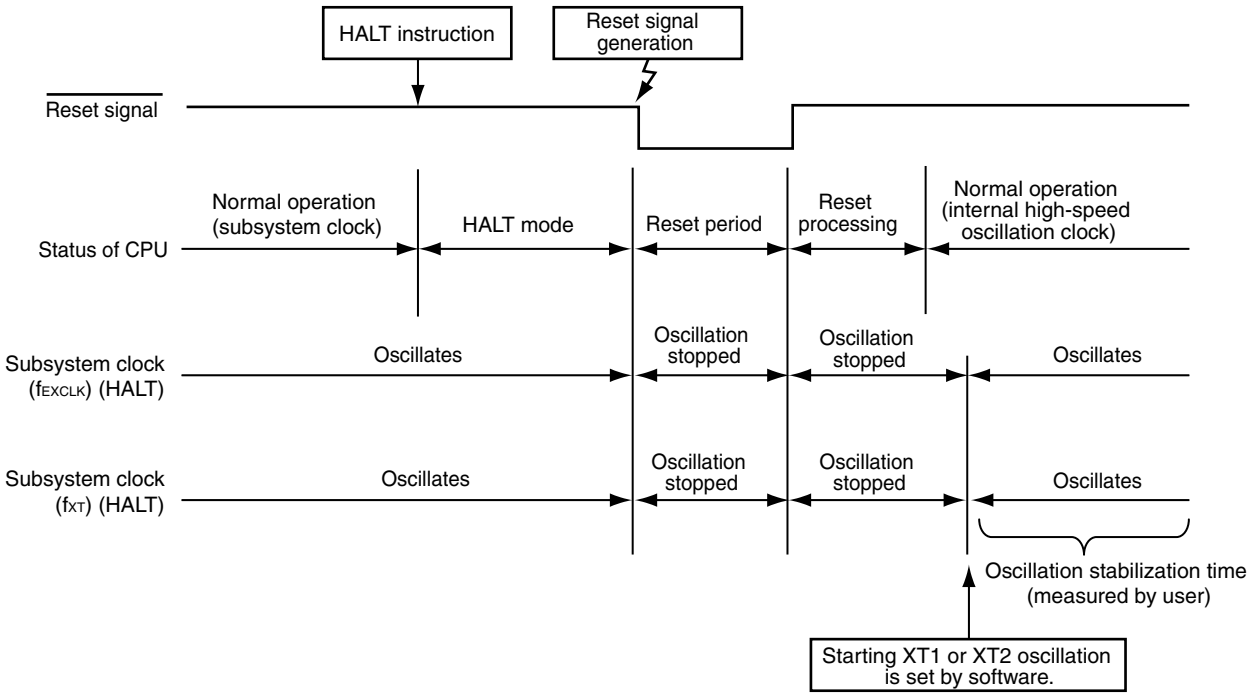


Table 2-12. Return Times in Case of Return from HALT Mode ($f_{CPU} \leftarrow f_{SUB}/2 \leftarrow f_{XT}$, f_{EXCLKS})

Source	HALT Mode/STOP Mode
Reset processing	12 to 51 μs
Oscillation stabilization time (measured by user) ($f_{CPU} \leftarrow f_{SUB}/2 \leftarrow f_{XT}$ only)	The system waits for the wait time suitable for the resonator that is used, as measured with a timer, etc.

2.6 Cautions on Clock Generator and Standby Function

- If the subsystem clock divided by 2 ($f_{SUB}/2$) is supplied to the CPU clock (f_{CPU}), the self programming function of the flash memory cannot be used. The self programming function can be used if the internal high-speed oscillation clock or the high-speed system clock is supplied to the CPU clock (f_{CPU}).

- If the peripheral hardware clock (f_{PRS}) is stopped through execution of the STOP instruction or by program, be sure to stop the peripheral hardware that uses the peripheral hardware clock as the clock source.

[Column] Self programming function

The 78K0/Kx2-L microcontrollers have a self programming function that allows flash memory rewriting via a user program. When using the self programming function, use the self programming library provided by NEC Electronics. For details about the self programming library, refer to the **About the EEPROM™ emulation library and self programming library (1)** column in **4.2 Initial Settings and Work Areas of CHAPTER 4**.

CHAPTER 3 REGULATOR

3.1 Regulator Overview

The 78K0/Kx2-L microcontrollers contain a circuit for operating the device with a constant voltage. At this time, in order to stabilize the regulator output voltage, connect the REGC pin to V_{SS} via a capacitor (0.47 to 1 μF). However, when using the STOP mode that has been entered since operation of the internal high-speed oscillation clock and external main system clock, 0.47 μF is recommended. Also, use a capacitor with good characteristics, since it is used to stabilize internal voltage.

The regulator output voltage is normally 2.4 V (typ.), and in the low power consumption mode, 2.0 V (typ.).

3.2 Register Controlling Regulator

(1) Regulator mode control register (RMC)

This register sets the output voltage of the regulator.

RMC is set with an 8-bit memory manipulation instruction.

Reset input sets this register to 00H.

Figure 3-1. Format of Regulator Mode Control Register (RMC)

Address: FF3DH After reset: 00H R/W

Symbol	7	6	5	4	3	2	1	0
RMC								

RMC[7:0]	Control of output voltage of regulator
56H	Fixed to low power consumption mode (2.0 V)
00H	Switches normal power mode (2.4 V) and low power consumption mode (2.0 V) according to the condition (refer to **Table 3-1**)
Other than above	Setting prohibited

Cautions 1. To change the RMC register setting value from 56H to 00H and use a CPU operating frequency of 5 MHz or more, change the PCC and RCM registers when 10 μs or more has elapsed after the RMC register was set.

2. When using the setting fixed to the low power consumption mode, the RMC register can be used in the following cases.

<When X1 clock is selected as the CPU clock>

$f_X \leq 5$ MHz and $f_{CPU} \leq 5$ MHz

<When the internal high-speed oscillation clock, external input clock, or subsystem clock is selected as the CPU clock>

$f_{CPU} \leq 5$ MHz

Table 3-1. Regulator Output Voltage Conditions

Mode	Output Voltage	Condition
Low power consumption mode	2.0 V	In STOP mode
		When both the high-speed system clock (f_{XH}) and the internal high-speed oscillation clock (f_{IH}) are stopped during CPU operation with the subsystem clock (f_{XT})
		When both the high-speed system clock (f_{XH}) and the internal high-speed oscillation clock (f_{IH}) are stopped during the HALT mode when the CPU operation with the subsystem clock (f_{XT}) has been set
Normal power mode	2.4 V	Other than above

3.3 Cautions on Self Programming

1. When executing self programming or EEPROM emulation, fix the mode of the regulator output voltage.
2. The power supply voltage range in which the flash memory can be rewritten in the normal power mode is $V_{DD} \geq$ 2.5 V. Moreover, in the case of the normal power mode, the program area can be rewritten by using the self programming library.
3. Observe the following points when rewriting the flash memory in low power consumption mode.
   - The data area can be rewritten in low power consumption mode, but the program area cannot.
   - The flash memory cannot be rewritten in low power consumption mode if the power supply voltage is equal to or lower than the output voltage of the regulator (2.0 V).
   - Flash memory that has been written and erased in the low power consumption mode cannot be accessed in the normal power mode. To use this data in the normal power mode, switch to the low power consumption mode and transfer the flash memory contents to RAM.
   - When executing rewrite during EEPROM emulation, overwriting the same block is possible. However, when executing rewrite by using the self programming library, overwriting the same block is not possible, so be sure to erase a block first before rewriting data to it.
   - A wait time of 2 ms is required before executing self programming after switching from the normal power mode to the low power consumption mode.

Remark For details about the self programming function and the self programming library, refer to the **78K0 Microcontrollers Self Programming Library Type01 User's Manual (U18274E)** and **78K0 Self Programming Library Type01 User's Manual 78K0/Kx2-L, 78K0/lx2 Specification Differences (ZBB-CB-09-0019)**.

For details about the EEPROM emulation, refer to the **78K0 Microcontrollers EEPROM Emulation Library Type01 User's Manual (U18275E)** and **78K0 EEPROM Emulation Library Type01 User's Manual 78K0/Kx2-L, 78K0/lx2 Specification Differences (ZBB-CB-09-0020)**.

CHAPTER 4 LOW POWER CONSUMPTION PROGRAM EXAMPLES

This chapter introduces sample programs that use the various low power consumption functions of the 78K0/KC2-L microcontroller.

The program lists in this chapter consist mainly of programs related to the standby function and the regulator. For all the programs, refer to the C source code.

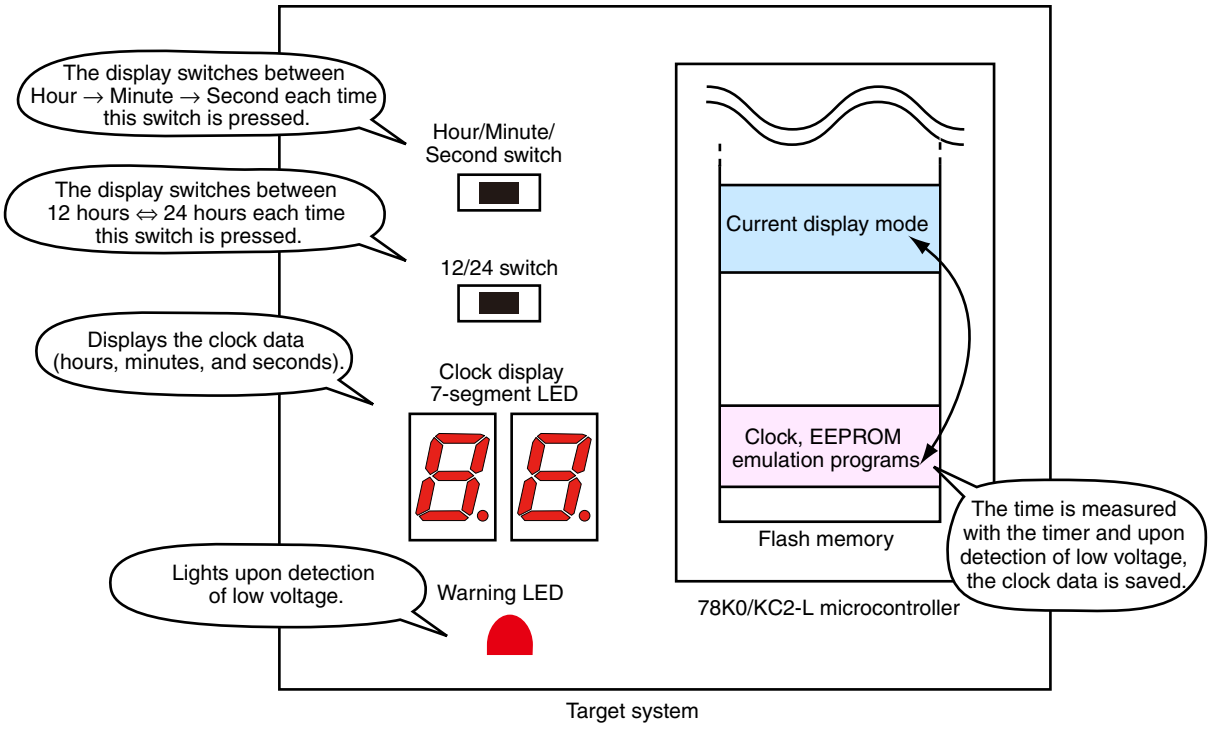
4.1 Specifications and Overall Flow

A clock function is realized by using a real-time counter. The display mode (hours, minutes, seconds, and 12-/24-hour display) can be changed by using switches.

Upon detection of a low voltage (2.53 V ±0.1 V) by the low-voltage detector, a warning LED lights up and the current display mode (hours, minutes, seconds, and 12-/24-hour display) is self programmed to the flash memory by using the EEPROM emulation library.

By cutting off the power once and then reapplying it after a low voltage has been detected, the display mode is regenerated and the data written to the flash memory can be checked.

Figure 4-1. Program Operation (Image)



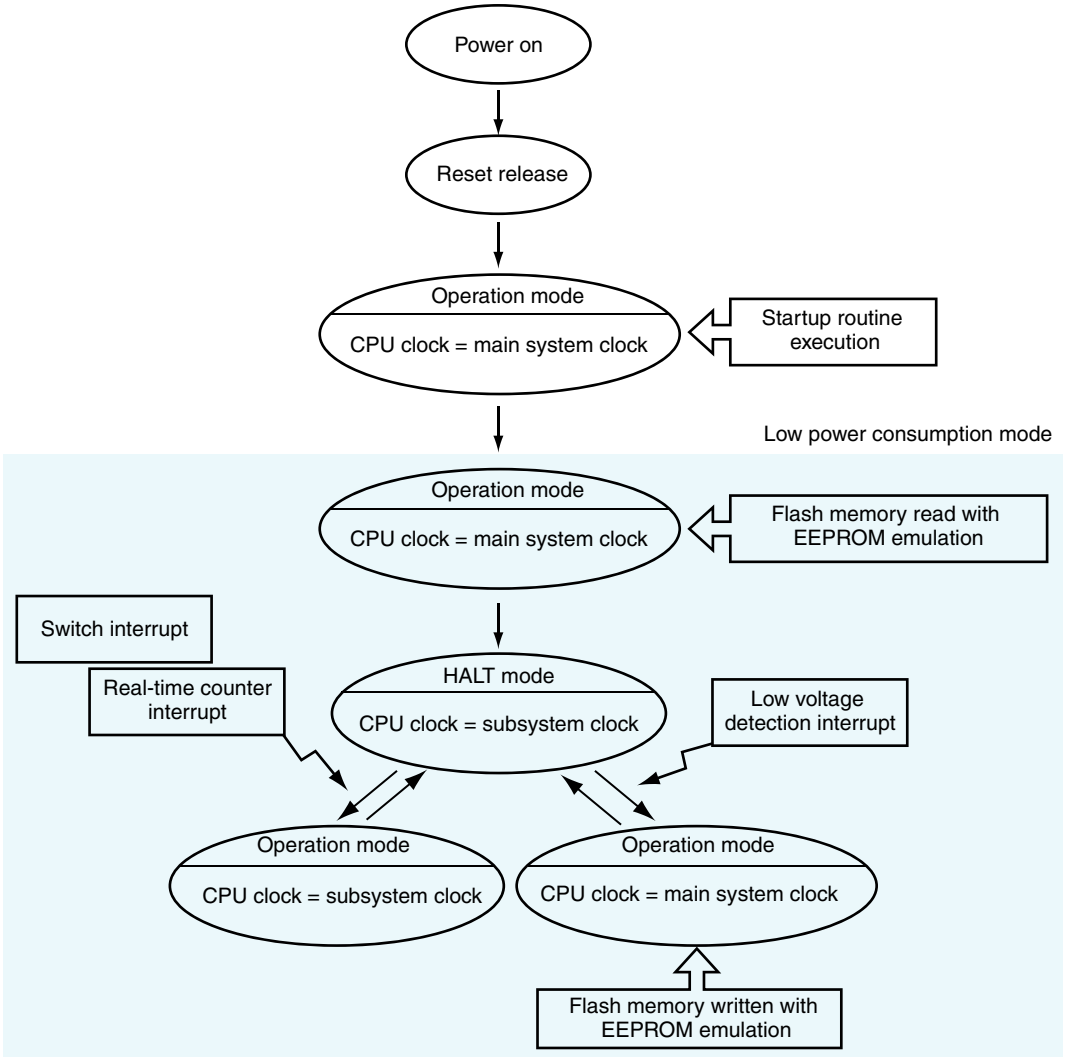
The status transition diagram of the sample program is shown below.

Following the execution of the startup routine, the low power consumption mode is set and the display mode is read using the EEPROM emulation library. Then the CPU clock is switched to clock supply from the subsystem clock and the HALT mode is entered.

The HALT mode is returned from and the mode becomes the operation mode (normal operating status) upon generation of an interrupt by the real-time counter, interrupt generation through switch input, or generation of a low voltage detection interrupt.

During the low voltage detection interrupt servicing, the CPU clock is switched to clock supply from the main system clock, and the display mode is written to the flash memory using the EEPROM emulation library.

Figure 4-2. Program Status Transition Diagram

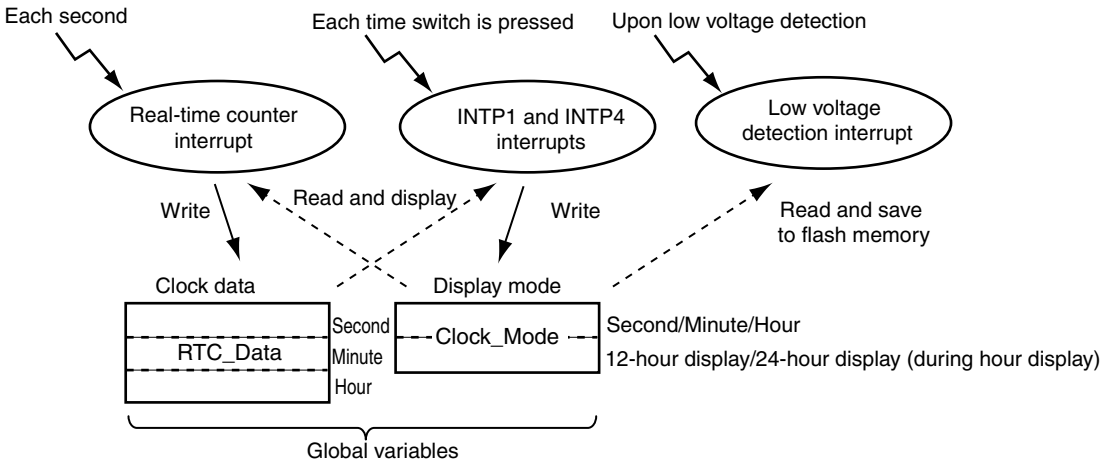


The interrupt servicing and data flow of the sample programs are shown below.

During real-time counter interrupt servicing, the hour, minute, and second data of the real-time counter is saved as variable to RTC_Data. During INTP1 and INTP4 interrupt servicing, the display mode variables are rewritten to Clock_Mode.

During low voltage detection interrupt servicing, the display mode is read and saved to the flash memory.

Figure 4-3. Interrupt Servicing and Data



In the sample programs, the CPU clock (f_{CPU}) supply source is switched according to the processing.

The program flow and clock status are shown below. The red characters indicate the changes made as the result of the program settings.

Figure 4-4. Program Processing and Clock Status

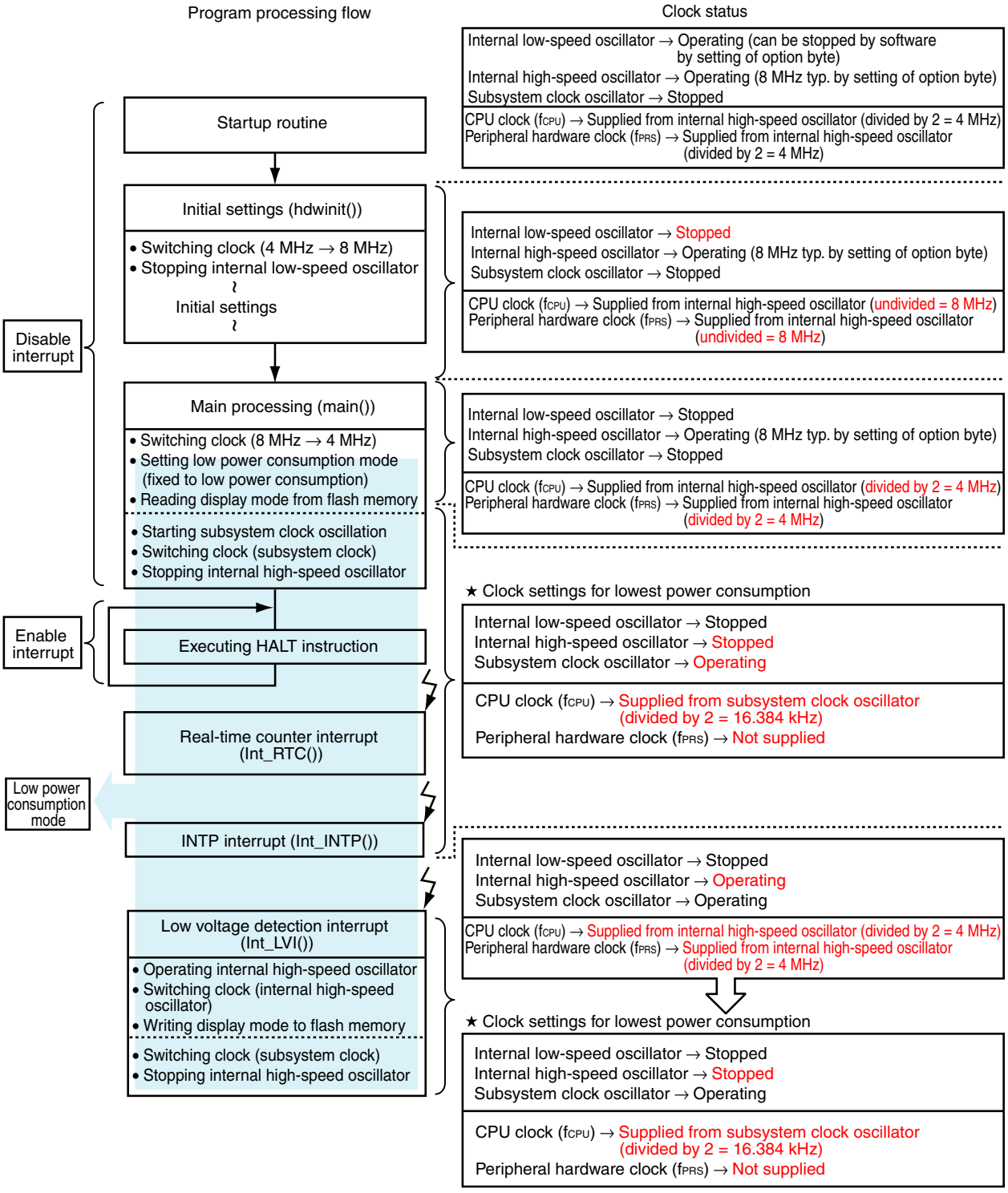


Table 4-1. File Configuration of Program (Folder Name: U19612_Kx2L_LowPower_01_AN)

Folder Name		File Name	Description
U19612_Kx2L_LowPower_01_AN		Kx2L_LowPower.lmf	Load module file
		Kx2L_LowPower.hex	HEX format object module file
		Kx2L_LowPower.prw	Project file
		–	Other generated files and project-related files
	/src	initial.c	Initial settings
		main.c	Main program
		Int_RTC.c	Real-time counter interrupt function
		Int_Intp.c	INTP edge detection interrupt function
		Int_LVI.c	Low voltage detection interrupt function
		option.asm	Option byte setting file
		LowPower.h	Header file
		Kx2_eee.h	Header file (for EEPROM emulation)

This sample program is designed for the TK-78K0/KC2L evaluation board of Tessera Technology Inc. (which incorporates the μPD78F0588GA-GAM-AX). For the operation method of the TK-78K0/KC2L, refer to the TK-78K0/KC2L documentation.

The I/Os and the special function registers (SFR) of the microcontroller that are used by the program are shown below.

Table 4-2. TK-78K0/KC2L I/Os Used by Program

I/O	Application	Connection Pin
7-segment LED	• Display "-" following power application. • Display hours, minutes, and seconds of clock data. • Switch display between Hour → Minute → Second each time the Hour/Minute/Second switch is pressed. • Switch display between 12 hours and 24 hours each time the 12/24 switch is pressed. • The default display is Second and 12-hour display, but in the case of startup after a low-voltage interrupt has been input, the display used until then is reproduced.	 Port 2 P20 to P27 pins <Digit switching> Port 0 P00 and P01 pins
Hour/Minute/Second switch (SW5)	Switch clock display between Hour → Minute → Second each time the switch is pressed.	P30/INTP1 pin
AM/PM switch (SW6)	Switch display between 12 hours and 24 hours each time the switch is pressed.	P33/TI51/TO51/INTP4 pin
Warning LED	Lights upon input of low voltage detection interrupt. (This LED is an addition to the TK-78K0/KC2L.)	Port 0 P02 pin

Table 4-3. Microcontroller SFRs Used by Program

SFR	Application
Real-time counter	Realize clock function (interrupt).
Low-voltage detector	Detect low voltage of power supply voltage (interrupt).
Interrupt pins	INTP1: Detect switching of Hour/Minute/Second switch. INTP4: Detect switching of 12/24 switch.
Ports	Port 2: Output to 7-segment LED. Ports 0_0, 0_1: Switch display digit of 7-segment LED. (The clock signal is output to each latch.)

4.2 Initial Settings and Work Areas

Since the flash memory is written to using the EEPROM emulation library in the sample programs, the header of EEPROM emulation must be included and the work areas for the libraries must be set.

Moreover, the path of the header file must be set using a compiler, and the EEPROM emulation library and the self programming library files must be set using a linker.

List 4-1. Startup Routine Initial Setting Function (main.c)

```
/*---------------------------------------------------------------------
*       Preprocessing directive (#pragma)
*---------------------------------------------------------------------*/

#pragma   sfr               /* SFR names can be described at the C source level */
#pragma   di                /* DI instructions can be described at the C source level */
#pragma   ei                /* EI instructions can be described at the C source level */
#pragma   nop               /* NOP instructions can be described at the C source level */
#pragma   halt              /* HALT instructions can be described at the C source level */
#pragma   stop              /* STOP instructions can be described at the C source level */

/*---------------------------------------------------------------------
*   Header files
*---------------------------------------------------------------------*/
#include  "eeelib.h"        /* EEPROM emulation library header */
#include  "Kx2_eee.h"       /* Sample program header */
#include  "LowPower.h"      /* Sample program header */

/*---------------------------------------------------------------------
*    Define macros
*---------------------------------------------------------------------*/
#define NOP_4  NOP();NOP();NOP();NOP();   /* Execute NOP 4 times */
#define NOP_8  NOP_4 NOP_4                /* Execute 2 times the macro that executes NOP 4 times (4 × 2 = 8) */

/*---------------------------------------------------------------------
*    Declare prototypes
*---------------------------------------------------------------------*/

/*---------------------------------------------------------------------
*    Global variables
*---------------------------------------------------------------------*/

sreg      UCHAR EntryRAM[100];                /* Entry RAM for self programming library */
UCHAR     EEPROM_DataBuf[EEPROM_BUFFLEN];     /* Data buffer for EEPROM emulation library */
UCHAR     Clock_Mode[EEPROM_DATALEN];
UCHAR     RTC_Data[3];
UCHAR     const LED_Pattern[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x83, 0xf8, 0x80,
                                 0x98, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e, 0xbf};
/**************************************************************************
          Initialization after RESET

**************************************************************************/
void hdwinit(void)
{
          /* Disable interrupts */
          DI();
          /* Peripheral hardware initial settings */
          Init_sfr();
}
```


[Column] About the EEPROM emulation library and self programming library (1)

The EEPROM emulation and self programming libraries provided by NEC Electronics are for writing to the flash memory from a user program.

Self programming libraries are used to write and erase flash memory.

EEPROM emulation libraries are used to realize operation as if data of any size were rewritten to flash memory whose smallest write unit is 4 bytes and erase size is 1 block (= 1024 bytes). Actual write and erase of flash memory calls the self programming library from the EEPROM emulation library.

All unconnected ports are set as output ports. After the subsystem clock is started, timer TM00 is started in order to measure the oscillation stabilization time.

List 4-2. Initial Setting Function (main.c) (1)

```
/************************************************************************************************************
;         Init_sfr()
;------------------------------------------------------------------------------------------------------------
;         Peripheral hardware initial settings
;------------------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************************/
void Init_sfr(void)
{
          /* For loop counter */
          UCHAR i;

          /* Specify the ROM and RAM sizes when using uPD78F0583 or uPD78F0588 */
          IMS =     0xC8;

          /* After reset release, set the internal high-speed oscillator to 8 MHz (typ.) by option byte setting */
          /* After reset release, use main clock for internal high-speed oscillator, and main clock divided by 2 for CPU clock */
          /* Switch CPU clock from main clock divided by 2 to undivided clock */
          Change_CPUClk(MAINSYSCLK2_1);

          /* 16-bit timer TM00 setting (Do not use subsystem clock's oscillation stabilization wait (approx. 2 s) interrupt) */
          TMC00 = 0b00000000;      /* Disable 16-bit timer 00. Disable operating clock supply. Clear counter 00 */
          TMMK000 = 1;             /* Set timer 00 interrupt mask (disable interrupt) */
          TMIF000 = 0;             /* Clear timer 00 interrupt request flag */
          CRC00 = 0b00000000;      /* Set operation using CR000 as compare register */
          TOC00 = 0b00000000;      /* Set output disable for T000 pin to output control register 00 */
          PRM00 = 0b00000010;      /* Count clock for prescaler = 256 cycles (31.25 kHz (when source = 8 MHz)) */
          CR000 = 62500-1;         /* 31.25 kHz ×(1/31250) × 62500                prox. 2 s */
          CR010 = 0xFF;

          /* Operate subsystem clock */
          Start_SubSystemClk();

          /* Start timer TM00 ↓ Measure 2 s from here for subsystem clock oscillation stabilization */
          Start_TM00();

          /* Stop internal low-speed oscillator */
          LSRSTOP = 1;

          /* Ports P00 to P02 */
          P0  = 0b00000100;        /* P00 is left digit of 7-segment LED, P01 is right digit, and P02 is low
                                      voltage warning LED ← Light out (expansion) */
          PM0 = 0b00000000;        /* Set as output port */

          /* Ports P10 to P17 */
          ADPC1 = 0b00000111;      /* P10 to P12 (ANI8 to ANI10): all digital I/O */
          PU1 = 0b00011000;        /* P13 and P14: connect internal pull-up resistor */
          P1 = 0b00000000;         /* P13 and P14: TxD and RxD (connected to microcontroller), all others unconnected */
          PM1 = 0b00011000;        /* P13 and P14: input, set all others as output ports */

          /* Ports P20 to P27 */
          ADPC0 = 0b11111111;      /* P20 to P27 (ANI0 to ANI7): all digital I/O */
          P2 = 0b10111111;         /* P20 to P27: Segment outputs of 7-segment LED; Display "-" */
          PM2 = 0b00000000;        /* All set as output ports */

          /* Ports P30 to P33 */
          PU3 = 0b00001111;        /* P30/INTP1 = SW5, P33/INTP4 = SW6, P31 = SW3-2, P32 = SW3-3 */
          PM3 = 0b11111111;        /* Set as input port ↑ Connect internal pull-up resistor */
```

Start subsystem clock operation

Start measurement of oscillation stabilization time (2 s)

[Column] About the EEPROM emulation library and self programming library (2)

If the user program uses the EEPROM emulation library, the self programming library will be also used, so make settings so that the library files of the EEPROM emulation library and self programming library are linked using the linker.

List 4-2. Initial Setting Function (main.c) (2)

```
/* Ports P40 to P42 */
P4 = 0b00000000;            /* P40 to P42: all unconnected */
PM4 = 0b00000000;           /* All set as output ports */

/* Ports P60 to P63 */
P6 = 0b00000000;            /* P60 to P63: all unconnected */
PM6 = 0b00000000;           /* All set as output ports */

/* Ports P70 to P75 */
PU7 = 0b00111111;           /* P70 to P75: SW3-4 to SW3-8; Connect internal pull-up resistor */
PM7 = 0b11111111;           /* Set as input ports */

/* Ports P120 to P125 */
PU12 = 0b00000000;          /* P120: unconnected */
PM12 = 0b11111110;          /* P120: set as output port, P121 to P125: input pins (X1, X2, XT1, XT2, RESET) */

/* Output data to D-FF of 7-segment LED */
LED_LEFT = 1;               /* Display to left digit (latch D-FF) */
LED_LEFT = 0;

LED_RIGHT = 1;              /* Display to right digit (latch D-FF) */
LED_RIGHT = 0;

/* Low-voltage detector (interrupt generated at 2.53 V ±0.1 V) LVIM = All 0 with option byte (operation disabled) */
LVIS = 0b00001011;          /* Set detection voltage to 2.53 V ±1 */
LVION = 1;                  /* Set operation of low-voltage detector to enable */

for(i=0; i<2; i++)          /* During 8 MHz operation with wait of approx. 10 μs: 1 clock = 125 ns */
{                           /* Since this loop lasts 90 clocks, actually 90 × 125 ns = 11.25 μs */
    NOP_4
}

                            /* ↑During 8 MHz operation, 1 clock = 125 ns  NOP = 2 clocks  10 μs/(125 * 2 ns) = 40 */
LVIMK = 1;                  /* Mask LVI interrupt (disable interrupt) */
LVIIF = 0;                  /* Clear LVI interrupt request flag */

/* Set INTP0 and INTP4 interrupts */
EGNCTL0 = 0b00010010;       /* Set INTP1 and INTP4 as falling edge detection interrupts */
EGPCTL0 = 0b00000000;       /* Set INTP1 and INTP4 as falling edge detection interrupts */
PMK1 = 1;                   /* Mask INTP1 interrupt (disable interrupt) */
PIF1 = 0;                   /* Clear INTP1 interrupt flag */
PMK4 = 1;                   /* Mask INTP4 interrupt (disable interrupt) */
PIF4 = 0;                   /* Clear INTP4 interrupt flag */

/* Set real-time counter (interrupt used) */
RTCEN    = 1;               /* Supply control clock for write access to real-time counter */
RTCE     = 0;               /* Stop operation of real-time counter */
RTCC0 = RTCC0|0b00100010;   /* Start counter operation. RTCCL pin 32.768 kHz output 1 s fixed interval
                               interrupt AMPM 12-hour system */
                            /* Enable output of RTCCL pin (32.768 kHz) */

/* Set real-time counter start time (comment time/yymmdd is default value of header file) */
SEC      = SEC_INI;         /* seconds: 50 */
MIN      = MIN_INI;         /* minutes: 59 */
HOUR     = HOUR_INI;        /* hours: 11 */
WEEK     = WEEK_INI;        /* day of the week: Saturday */
DAY      = DAY_INI;         /* day: 01 */
MONTH    = MONTH_INI;       /* month: 01 */
YEAR     = YEAR_INI;        /* year: 00 */

RTCMK    = 1;               /* Mask fixed interval or alarm interrupt of real-time counter (disable interrupt) */
RTCIF    = 0;               /* Clear interrupt request flag after changing CT0 to CT2 */

RTCEN    = 0;               /* Stop supplying control clock for write access to real-time counter */
}
```

Expand NOP 4 times through macro definition

Supply control clock

Stop control clock supply

[Column] Setting of registers related to real-time counter

The real-time counter operates with the subsystem clock (f_{SUB}), but related registers can be set even if the subsystem clock (f_{SUB}) is stopped, as long as the CPU clock (f_{CPU}) is being supplied. Writing to related registers requires control clock supply. (Control clock supply is not required for reading these registers.)

The control clock is supplied by setting RTCEN of peripheral enable register 0 (PER0) to 1. When not performing write to related registers, stopping control clock supply (RTCEN = 0) reduces the power consumption.

4.3 Main Processing

The main() function is called after the hdwinit() function has been executed in the startup routine and the memory initialization settings, etc., have been performed.

Before the real-time counter is started, the display mode is read from the flash memory. If the display mode is written through a low voltage detection interrupt, the previous display is reproduced.

Since the operation of the subsystem clock is started in the hdwinit() function and timer TM00 is started for 2 second measurement, whether 2 second measurement has been done or not is detected before the subsystem clock is supplied to the CPU clock. If 2 seconds have not elapsed, the timer TM00 interrupt is enabled and the system enters standby in the HALT mode. The interrupt enable flag is disabled (IE = 0), so the operation does not branch to an interrupt vector. Therefore, when an interrupt request is generated, the interrupt request flag must be cleared manually after the interrupt request has been generated.

List 4-3. Main Function (main.c)

```
/************************************
Executed from startup routine
************************************/
void main(void)
{
        /* Change CPU clock from undivided main clock
        Change_CPUClk(MAINSYSCLK1_2);

        /* Set low power consumption mode */
        Set_LowPowerMode();

        /* Read display mode from flash memory */
        Read_ClockMode();

        /* Execute DI after EEPROM emulation library execution */
        DI();

        /* Set 12-/24-hour display from the display mode to the real-time counter */
        Set_ampm();

        /* Is timer TM00 request flag set (2 second lapse)? */
        if(Wait_TM00() == FALSE)
        {
                /* Clear timer 00 interrupt mask flag (enable interrupt) */
                Clear_Tm00IntMsk();
                /* HALT mode until timer TM00 interrupt is generated */
                 HALT();
        }
        /* Clear timer 00 interrupt request flag */
        Clear_Tm00IntReq();

        /* Stop timer TM00 */
        Stop_TM00();

        /* Set subsystem clock to CPU clock */
        Change_CPUClk(MAINSYSCLK_SUBCLK);

        /* Stop internal high-speed oscillator */
        Stop_HispeedIntClk();

        /* Start real-time counter */
        Start_RTC();

        /* Release interrupt mask flags */
        Clear_IntMask();

        /* Enable interrupt */
        EI();

        /* Infinite loop */
        while(1)
        {
                /* HALT mode */
                HALT();

        }
}
```

Switch the CPU clock to the main clock divided by 2 to set the low power consumption mode. (Since the main clock is the internal high-speed oscillator 8 MHz (typ.), the result is 4 MHz (typ.).) (The low power consumption mode fixed setting is only if the CPU clock is 5 MHz or lower.)

If setting interrupt disable during the period that includes execution of the EEPROM emulation library is wished, the following sequence must be followed: First set the interrupt mask flag, then execute the EEPROM emulation library, next execute DI(), and then clear the interrupt mask flag. At this point in time, the interrupt mask flag is set, so only DI() is executed. (For details, refer to **4.6 Low Voltage Interrupt Servicing**.)

Execute HALT instruction

Be sure to clear the request flag if interrupt servicing is not done.

Execute HALT instruction

(1) Setting clock generator operation/stop

When starting generation of the various clocks, the lapse of the oscillation stabilization time from when clock generator operation starts must be waited for.

Whether the oscillation of the internal high-speed oscillator is stable can be checked by referencing the status (RSTS = 1: Stable operation).

In the case of the subsystem clock using the XT1 resonator, the lapse of the oscillation stabilization time must be waited for using a timer, etc. Timer TM00 measures 2 seconds in the program. (For the oscillation stabilization time, contact the manufacturer of the resonator that is used.)

Prior to stopping the internal high-speed oscillator, check that another oscillation clock (high-speed system clock or subsystem clock) is supplied to the CPU clock (f_{CPU}).

List 4-4. Operating or Stopping Clock Generator (main.c)

```
/************************************************************************************************
;         Set_HispeedIntClk()
;-----------------------------------------------------------------------------------------------
;         Operate internal high-speed oscillator
;-----------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Start_HispeedIntClk(void)
{
          /* Operate internal high-speed oscillator */
          RSTOP = 0;

          /* Oscillation stabilization wait for internal high-speed oscillator */
          while (RSTS == 0)
          {
                    NOP();
          }
}
/************************************************************************************************
;          Stop_HispeedIntClk()
;-----------------------------------------------------------------------------------------------
;         Stop internal high-speed oscillator
;-----------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Stop_HispeedIntClk(void)
{
          /* Stop only if CPU clock is running on high-speed system clock or subsystem clock */
          if((CLS == 1)||(MCS == 1))
          {
                    RSTOP = 1;
          }
}

/************************************************************************************************
;         Set_SubSystemClk()
;-----------------------------------------------------------------------------------------------
;         Operate subsystem clock
;-----------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Start_SubSystemClk(void)
{
          /* Set the subsystem clock to XT1 oscillation mode (connected to crystal resonator) */
          XTSTART = 1;
}
```

Prior to stopping the internal high-speed oscillator, check that another oscillation clock is supplied to the CPU clock.

[Column] Startup routine

In the standard CC78K0 compiler startup routine, first the hdwinit() function is called, the initial settings of the variables (both initial value and no initial value) are done, and then the main() function is called. After a value is set to a variable in the hdwinit() function, initialization is performed, so caution is required.

(2) Switching CPU clock (f_{CPU})

When the supply source of the CPU clock or its division ratio has been switched, some time is required until the switch takes full effect.

When the supply source is switched from the main system clock to the subsystem clock, or vice-versa, this can be checked with the CLS bit, but when the clock is supplied from the main system clock and the division ratio of the prescaler is switched, the system waits for the time required for the clock switch to take effect.

List 4-5. Switching CPU Clock (main.c)

```
/************************************************************************************************************
;       Change_Clock()
;------------------------------------------------------------------------------------------------------------
;       Switch supply source of CPU clock
;------------------------------------------------------------------------------------------------------------
;       [I N]   UCHAR clk
;       [OUT]   -
;************************************************************************************************************/
void Change_CPUClk(UCHAR clk)
{
        switch(clk)
        {
                /* Change CPU clock from main system clock divided by 2 to undivided clock */
                case MAINSYSCLK2_1:
                {
                        /* PCC2, PCC1, PCC0 = 0 (undivided) */
                        PCC = PCC & 0b11111000;

                        /* Wait for time required for clock switch (fXP/2 → fXP) (8 clocks) */
                        /* NOP requires 2 clocks, so 8/2 = 4 */
                        NOP_4

                        break;
                }

                /* Change CPU clock from undivided main system clock to main system clock divided by 2 */
                case MAINSYSCLK1_2:
                {
                        /* PCC2, PCC1 = 0, PCC0 = 1 (divided by 2) */
                        PCC = PCC | 0b00000001;

                        /* Wait for time required for clock switch (fXP → fXP/2) (16 clocks) */
                        /* NOP requires 2 clocks, so 16/2 = 8 */
                        NOP_8

                        break;
                }

                /* Change CPU clock from main system clock to subsystem clock (32.768 kHz) divided by 2 */
                case MAINSYSCLK_SUBCLK:
                {
                        /* Supply CPU clock from subsystem clock */
                        CSS = 1;

                        /* Check status of CPU clock with CLS bit (1 = subsystem clock) */
                        while(CLS == 0)
                        {
                                NOP();
                        }

                        break;
                }

                /* Change CPU clock from subsystem clock (32.768 kHz) divided by 2 to main system clock */
                case SUBCLK_MAINSYSCLK:
                {
                        /* PCC2, PCC10 = 0, PCC0 = 1 (divided by 2) */
                        PCC = PCC | 0b00000001;

                        /* Supply CPU clock from main system clock */
                        CSS = 0;

                        /* Check status of CPU clock with CLS bit (0 = main system clock) */
                        while(CLS == 1)
                        {
                                NOP();
                        }
                }
        }
}
```

Expand NOP 4 times through macro definition

Wait for the required time when the division ratio of the main system clock is switched.

Expand NOP 8 times through macro definition

In the case of main system clock ↔ subsystem clock switching, check with CLS.

(3) Setting low power consumption mode

The regulator mode control register (RMC) is set to 56H.

List 4-6. Setting Low Power Consumption Mode (main.c)

```
/************************************************************************************************
;        Set_LowPowerMode()
;------------------------------------------------------------------------------------------------
;        Set low power consumption mode
;------------------------------------------------------------------------------------------------
;        [I N]    -
;        [OUT]    -
;************************************************************************************************/
void Set_LowPowerMode(void)
{
         /* Set regulator (fix to low power consumption mode (2.0 V)) */
         RMC = 0x56;
}
```

(4) Reading flash memory with EEPROM emulation

Whether the display mode has been written to the flash memory is read. If the display mode has been written (if the low voltage detection interrupt has been generated), these contents are set to the variables. If the display mode has not been written, the initial value (display seconds, 12-hour format) is set to the variables.

List 4-7. Reading Flash Memory (main.c)

```
/************************************************************************************************
;        Read_ClockMode()
;------------------------------------------------------------------------------------------------
;        Read display mode (12-/24-format) with EEPROM emulation library
;------------------------------------------------------------------------------------------------
;        [I N]    -
;        [OUT]    -
;************************************************************************************************/
void Read_ClockMode(void)
{
        UCHAR Result;

        /* Set FLMDPUP */
        FLMDPUP = 1;

        /* Initial setting for EEPROM emulation */
        Init_EEPROM();

        /* Read display mode with EEPROM emulation and set as initial value */
        Result = ucEEPROMReadEx_A( EEPROMDATA_NO, Clock_Mode);

        /* Clear FLMDPUP */
        FLMDPUP = 0;

        /* Set initial value (second, 12-hour display) in case of read error */
        if (Result != EEE_NORMAL)
        {
                Clock_Mode[ MODE_HHMMSS] = SS;
                Clock_Mode[ MODE_AMPM] = MODE_12;
        }
}
```

FLMDPUP settings

[Column] FLMDPUP bit

In the 78K0/Kx2-L Series microcontrollers, the FLMDPUP bit must be set to 1 to perform self programming. To use the EEPROM emulation library, set the FLMDPUP bit to 1 prior to executing the library, and following library execution, set the FLMDPUP bit back to 0.

(5) Setting AMPM bit of real-time counter

In the main() function, the AMPM bit of the real-time counter is set after the display mode has been read from the flash memory.

In the sample program, the value of 12-hour display is set to the hour count register HOUR in the initial setting, so the setting value is modified if the display mode is changed from 12-hour display to 24-hour display. Moreover, when the AMPM bit is changed from 12-hour display to 24-hour display, the value of the hour count register becomes 0, so be sure to save this value beforehand.

For details on how to modify the setting values, refer to **4.5 INTP1, INTP4 Interrupt Servicing**.

List 4-8. Setting AMPM Bit (main.c)

```
/************************************************************************************************
;         Set_ampm()
;------------------------------------------------------------------------------------------------
;         Change AMPM bit of real-time counter
;------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Set_ampm(void)
{

          UCHAR tmp;

          /* Supply control clock for write access to real-time counter */
          RTCEN     = 1;                                                    <- Supply control clock

          /* In the case of 12-hour display */
          if(Clock_Mode[MODE_AMPM] == 0)
          {
                    /* Set AMPM (3rd bit) to 0 (12-hour format) */
                    RTCC0 = RTCC0 &  0b11110111;

          }

          /* In the case of 24-hour display */
          else
          {
                    /* Read time register (save value of this register before changing AMPM bit) */
                    tmp = HOUR;

                    /* Set AMPM (3rd bit) to 1 (24-hour format) */
                    RTCC0 = RTCC0 |  0b00001000;

                    /* Change 12-hour display to 24-hour display and set */
                    /* If 12H, set 00H */
                    if(tmp == 0x12)
                    {
                              HOUR = 0x00;
                    }
                    /* If 32H, set 12H */
                    else if(tmp == 0x32)
                    {
                              HOUR = 0x12;
                    }
                    /* If 21H to 27H, 30H, and 31H, deduct 0EH */
                    else if((0x21 <= tmp && tmp <= 0x27)||(tmp==0x30)||(tmp==0x31))
                    {
                              HOUR = tmp - 0x0e;
                    }
                    /* If 28H and 29H, deduct 08H */
                    else if((tmp==0x28)||(tmp==0x29))
                    {
                              HOUR = tmp - 0x08;
                    }

          }

          /* Stop supplying control clock for write access to real-time counter */
          RTCEN     = 0;                                                    <- Stop control clock supply

}
```

4.4 Real-Time Counter Interrupt Servicing

In the sample program, the display is updated every second by using the alarm interrupt of the real-time counter.

When reading the hour, minute, and second count registers, the RWAIT bit of real-time counter control register 1 (RTCC1) must be set to 1 (counter stop setting, read/write mode) to stop the counter, and whether the counter is stopped must be checked with the RWST flag. However, if the various registers are set while the real-time counter is stopped (RTCE = 0), the RWAIT bit need not be set. When the real-time counter is stopped, the RWST flag remains unchanged even when the RWAIT bit is set.

List 4-9. Real-Time Counter Interrupt (Int_RTC.c) (1)

```
/*---------------------------------------------------------------------
*       Preprocessing directive (#pragma)
*--------------------------------------------------------------------*/
#pragma   sfr
#pragma   nop
#pragma interrupt INTRTC Int_RTC

/*---------------------------------------------------------------------
*       Include header files
*--------------------------------------------------------------------*/
#include  "Kx2_eee.h"                  /* Sample program header */
#include  "LowPower.h"                 /* Sample program header */
/*---------------------------------------------------------------------
*       Declare prototypes
*--------------------------------------------------------------------*/
void Set_DisplayData_by_ssmmddMode( UCHAR *);
void Set_DisplayData_by_12_24Mode( UCHAR *);
void Display_LED( UCHAR *);
/*---------------------------------------------------------------------
*       External reference (variables)
*--------------------------------------------------------------------*/
extern UCHAR Clock_Mode[EEPROM_DATALEN];
extern UCHAR RTC_Data[3];
extern UCHAR const LED_Pattern[];

/*****************************************************************************************************
;         RTC_1sec()  Real-time counter interrupt function
;-----------------------------------------------------------------------------------------------------
;         Called every 1 second
;-----------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;*****************************************************************************************************/
void Int_RTC(void)
{
          UCHAR display_data[2];

          /* Stop counter, counter value read/write mode */
          RWAIT = 1;

          /* Wait until counter value read/write mode is set */
          while(RWST == 0)
          {
                    NOP();
          }

          /* Read counter register value */
          RTC_Data[SS] = SEC;
          RTC_Data[MM] = MIN;
          RTC_Data[HH] = HOUR;

          /* Counter operating mode */
          RWAIT = 0;

          /* Set display data according to display mode (hour/minute/second) */
          Set_DisplayData_by_ssmmddMode( display_data );

          /* Display to 7-segment LED */
          Display_LED( display_data);

          /* Clear interrupt request flag */
          RTCIF = 0;

}
```

Display mode

Clock data

Set counter value read/write mode

Read register

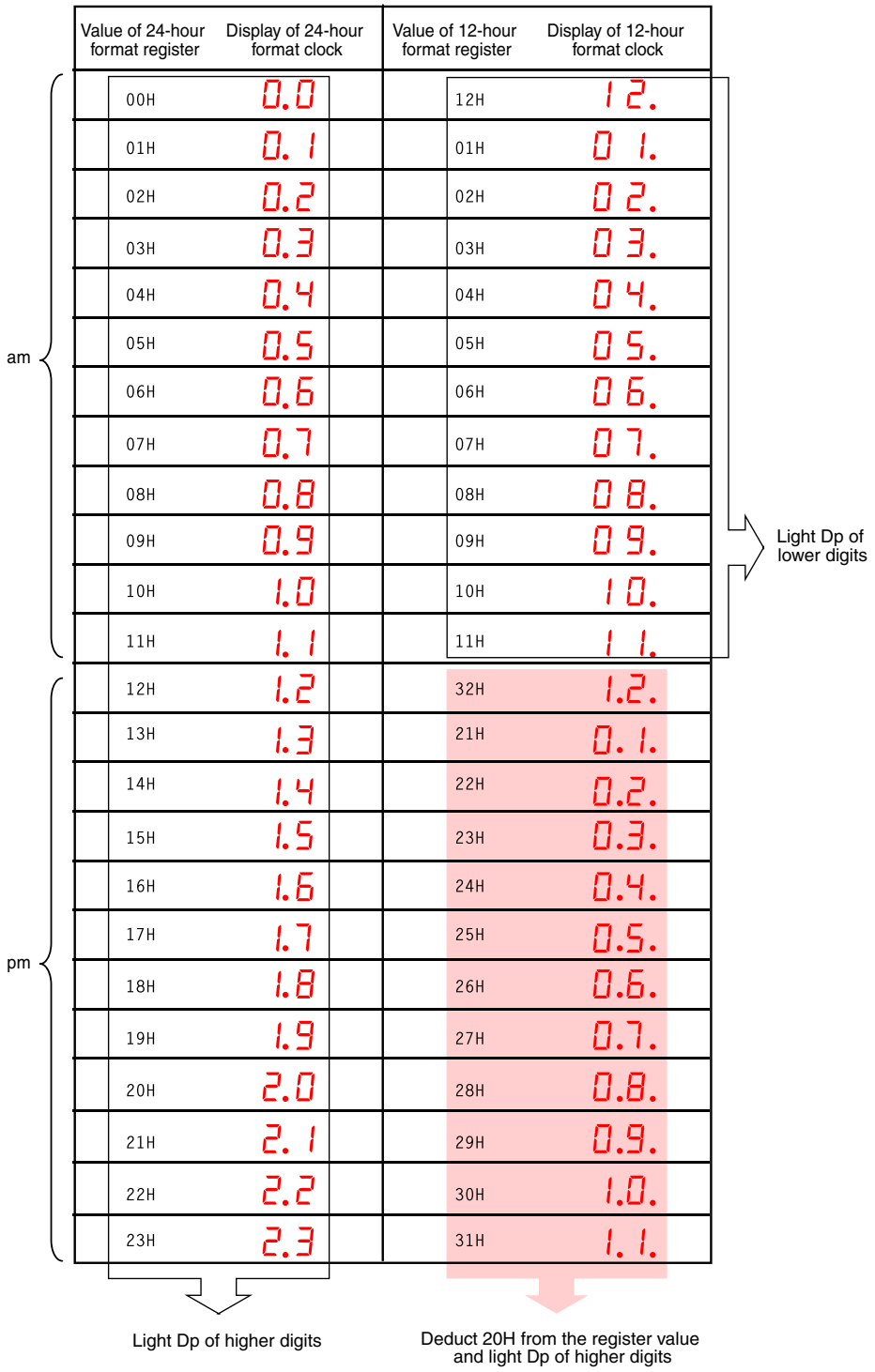
Return to counter operating mode

The correspondence between the value of the hour count register and the display pattern of the 7-segment LED is shown below.

In the case of the 24-hour format, the Dp of higher digits is lit, and in the case of the 12-hour format, the Dp of lower digits is lit. Moreover, for pm hours in the 12-hour format, the Dp of higher digits is also lit.

In the case of the 12-hour format, 21H and higher values differ from the actual clock display, so they must be processed prior to being displayed.

Figure 4-5. Correspondence Between Hour Count Register and Clock Display Pattern



List 4-9. Real-Time Counter Interrupt (Int_RTC.c) (2)

```
void Set_DisplayData_by_ssmmddMode( UCHAR *display_data)
{
        UCHAR tmp;

      switch (Clock_Mode[MODE_HHMMSS])
      {
                /* Display minute */
                case MM:
                {
                        tmp = RTC_Data[MM];
                        display_data[LOW] = LED_Pattern[(tmp & 0x0f)];
                        display_data[HIGH] = LED_Pattern[((tmp & 0xf0)>>4)];
                        break;
                }

                /* Display hour */
                case HH:
                {
                        /* Set hour display data according to display mode (12-/24-hour display) */
                        Set_DisplayData_by_12_24Mode( display_data );
                        break;
                }

                /* Display second if 0 or another value */
                default:
                {
                        tmp = RTC_Data[SS];
                        display_data[LOW] = LED_Pattern[(tmp & 0x0f)];
                        display_data[HIGH] = LED_Pattern[((tmp & 0xf0)>>4)];
                }
      }
}
```

List 4-9. Real-Time Counter Interrupt (Int_RTC.c) (3)

```
void Set_DisplayData_by_12_24Mode( UCHAR *display_data)
{

        UCHAR tmp;

        tmp = RTC_Data[HH];

        /* Distinction of 12-/24-hour display */
      if (Clock_Mode[MODE_AMPM] == 0)
      {
                /* Modify display data for PM12 to PM11 and light DP of higher digits */

              if (tmp >= 0x21)
              {
                        tmp = tmp - 0x20;
                        display_data[HIGH] = (LED_Pattern[((tmp & 0xf0)>>4)])& 0b01111111;
              }
              else
              {
                        display_data[HIGH] = LED_Pattern[((tmp & 0xf0)>>4)];
              }
              /* If 12-hour display, light DP of lower digits */
              display_data[LOW] = (LED_Pattern[(tmp & 0x0f)])& 0b01111111;
      }
      else
      {
                /* If 24-hour display, light DP of higher digits */
                display_data[LOW] = LED_Pattern[(tmp & 0x0f)];
                display_data[HIGH] = (LED_Pattern[((tmp & 0xf0)>>4)]) & 0b01111111;
      }
}
```

If the register value is 21H or higher, deduct 20H and light Dp of higher digits

If 12-hour display, display Dp of lower digits

If 24-hour display, display Dp of higher digits

[Column] Rewriting port registers while peripheral hardware clock supply is stopped

During real-time counter, INTP1, and INTP4 interrupts, the internal high-speed oscillator is stopped so no clock is supplied to the peripheral hardware, but port registers can be rewritten if the CPU clock (f_{CPU}) is supplied.

4.5 INTP1, INTP4 Interrupt Servicing

Switching of the hour/minute/second display by switch is done with the INTP1 interrupt, and 12-/24-hour display switching by switch is done with the INTP4 interrupt.

(1) INTP1 interrupt servicing

The display mode is switched each time an interrupt is input. The display of the 7-segment LED is also switched according to the display mode.

List 4-10. INTP1 Interrupt Servicing (Int_Intp.c)

```
/*-----------------------------------------------------------------------
*       Preprocessing directive (#pragma)
*-----------------------------------------------------------------------*/
#pragma  sfr
#pragma  nop
#pragma interrupt INTP1 SW_hhmmss
#pragma interrupt INTP4 SW_12_24

/*-----------------------------------------------------------------------
*       Include header files
*-----------------------------------------------------------------------*/
#include "Kx2_eee.h"         /* Sample program header */
#include "LowPower.h"        /* Sample program header */

/*-----------------------------------------------------------------------
*       External reference (functions)
*-----------------------------------------------------------------------*/
extern void Set_DisplayData_by_ssmmddMode( UCHAR *);
extern void Set_DisplayData_by_12_24Mode( UCHAR *);
extern void Display_LED( UCHAR *);
/*-----------------------------------------------------------------------
*       External reference (variables)
*-----------------------------------------------------------------------*/
extern UCHAR     Clock_Mode[EEPROM_DATALEN];   <- Display mode
extern UCHAR     RTC_Data[3];                  <- Clock data

/*****************************************************************************************************
;       SW_ttmmss()  INTP1 interrupt function
;-----------------------------------------------------------------------------------------------------
;       Called each time switch SW5 is pressed
;-----------------------------------------------------------------------------------------------------
;       [I N]    -
;       [OUT]    -
;*****************************************************************************************************/
void SW_hhmmss(void)
{
        UCHAR display_data[2];     /* Display data */

        /* See port status (SW chattering countermeasure) */
        if(P3.0 == 0)
        {

                switch (Clock_Mode[MODE_HHMMSS])
                {
                        /* If second, then display minute */
                        case SS:
                        {
                                Clock_Mode[MODE_HHMMSS] = MM;
                                break;
                        }

                         /* If minute, then display hour */
                        case MM:
                        {
                                Clock_Mode[MODE_HHMMSS] = HH;
                                break;
                        }

                        /* If hour or other, then display second */
                        default:
                        {
                                Clock_Mode[MODE_HHMMSS] = SS;
                        }
                }

                /* Set display data according to display mode (hour/minute/second) */
                Set_DisplayData_by_ssmmddMode( display_data );

                /* Display to 7-segment LED */
                Display_LED( display_data);

                /* Clear interrupt request flag */
                PIF1 = 0;
        }
}
```

(2) INTP4 interrupt

When the 12-hour format is switched to 24-hour display (the value of the AMPM bit is changed from 0 to 1), the value of the hour count register HOUR becomes 0, so the register contents must be saved. Also, since 12-hour display values and 24-hour format values differ, they must be converted and then return to the hour counter register HOUR.

List 4-11. INTP4 Interrupt Servicing (Int_Intp.c) (1)

```
void SW_12_24(void)
{
        UCHAR tmp;              /* Temporary saving */
        UCHAR display_data[2];  /* Display data */

        /* See port status (SW chattering countermeasure) */
        if(P3.3 == 0)
        {
                /* Supply control clock for write access to real-time counter */
                RTCEN    = 1;

                /* Stop counter, counter value read/write mode */
                RWAIT = 1;

                /* Wait until counter value read/write mode is set */
                while(RWST == 0)
                {
                        NOP();
                }

                /* Save hour count register */
                tmp = HOUR;

                /* If 24, display using 12-hour display */
                if(Clock_Mode[MODE_AMPM] == MODE_12)
                {
                        /* Switch 12 ← 24 display */
                        Clock_Mode[MODE_AMPM] = MODE_24;

                        /* Set AMPM (3rd bit) to 1
                           (24-hour display) */
                        RTCC0 = RTCC0 |  0b00001000;

                        /* Change 12-hour format to
                           24-hour display and return */
                        if(tmp == 0x12)
                        {
                                HOUR = 0x00;
                        }
                        else if(tmp == 0x32)
                        {
                                HOUR = 0x12;
                        }
                        else if((0x21 <= tmp && tmp <= 0x27)
                                                ||(tmp==0x
                                                ||(tmp==0x
                        {
                                HOUR = tmp - 0x0e;
                        }
                        else if((tmp==0x28)||(tmp==0x29))
                        {
                                HOUR = tmp - 0x08;
                        }
                        else
                        {
                                HOUR = tmp;
                        }
                        RTC_Data[HH] = HOUR;
                }
```

Supply control clock

Set counter value read/write mode

Save register

Value of 24-hour format register	Value of 12-hour format register	
00H	12H	Convert 00H ↔ 12H
01H	01H	
02H	02H	
03H	03H	
04H	04H	
05H	05H	
06H	06H	Not processed
07H	07H	
08H	08H	
09H	09H	
10H	10H	
11H	11H	
12H	32H	Convert 12H ↔ 32H
13H	21H	
14H	22H	
15H	23H	
16H	24H	Deduct 0EH
17H	25H	
18H	26H	
19H	27H	
20H	28H	Deduct 08H
21H	29H	
22H	30H	Deduct 0EH
23H	31H	

List 4-11. INTP4 Interrupt Servicing (Int_Intp.c) (2)

```
        /* If 24 or other, display using 12-hour display */
        else
        {
                /* Switch 12→ 24 display */
                Clock_Mode[MODE_AMPM] = MODE_12;

                /* Set AMPM (3rd bit) to 0 (12-hour display) */
                RTCC0 = RTCC0 &  0b11110111;

                /* Change 24-hour format to 12-hour display and return */
                if(tmp == 0x00)
                {
                        HOUR = 0x12;
                }
                else if(tmp == 0x12)
                {
                        HOUR = 0x32;
                }
                else if((0x13 <= tmp && tmp <= 0x19)||(tmp==0x22)||(tmp==0x23))
                {
                        HOUR = tmp + 0x0e;
                }
                else if((tmp==0x20)||(tmp==0x21))
                {
                        HOUR = tmp + 0x08;
                }
                else
                {
                        HOUR = tmp;
                }
                RTC_Data[HH] = HOUR;
        }

        /* Counter operating mode */
        RWAIT = 0;          ← Return to counter operating mode

        /* Stop supplying control clock for write access to real-time counter */
        RTCEN     = 0;      ← Stop control clock supply

        /* Set hour display data according to display mode (hour/minute/second) */
        Set_DisplayData_by_ssmmddMode( display_data );

        /* Display to 7-segment LED */
        Display_LED( display_data);
    }

    /* Clear interrupt request flag */
    PIF4 = 0;

}
```

4.6 Low Voltage Detection Interrupt Servicing

Upon detection of a low voltage of 2.53 V ±0.1 V, a low voltage detection interrupt is generated.

During interrupt servicing, in order to write the display mode to the flash memory with the EEPROM emulation library, the internal high-speed oscillator must be operated and the CPU clock (f_{CPU}) must be switched momentarily to clock supply from the internal high-speed oscillation clock (4 MHz (typ.)). (To execute self programming of the flash memory, a clock other than the subsystem clock must be supplied to the CPU clock (f_{CPU}).) Following the write operation, return the CPU clock (f_{CPU}) to supply from the subsystem clock (f_{SUB}) and stop the internal high-speed oscillator.

To disable interrupts during EEPROM emulation library execution, the other interrupt mask flags must be set.

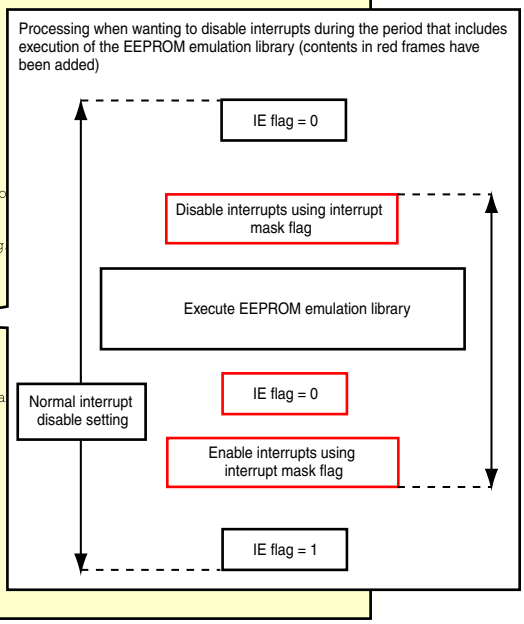
List 4-12. LVI Interrupt Servicing (Int_LVI.c) (1)

```
/*-----------------------------------------------------------------------
*       Preprocessing directive (#pragma)
*-----------------------------------------------------------------------*/
#pragma sfr
#pragma interrupt INTLVI LVI_vdd

/*-----------------------------------------------------------------------
*       Include header files
*-----------------------------------------------------------------------*/
#include  "eeelib.h"          /* EEPROM emulation library header */
#include  "Kx2_eee.h"         /* Sample program header */
#include  "LowPower_01.h"     /* Sample program header */

/*-----------------------------------------------------------------------
*       Declare prototypes
*-----------------------------------------------------------------------*/
UCHAR EEPROM_Write(const UCHAR, const UCHAR *, UCHAR *);

/*-----------------------------------------------------------------------
*       External reference (functions)
*-----------------------------------------------------------------------*/
extern void Write_ClockMode(void);
extern void Change_CPUClk(UCHAR clk);
extern void Set_HispeedIntClk(void);
extern void Stop_HispeedIntClk(void);

/*-----------------------------------------------------------------------
*       External reference (variable)
*-----------------------------------------------------------------------*/
extern UCHAR       Clock_Mode[EEPROM_DATALEN];

/*****************************************************************************************************
;         LVI_vdd()
;-----------------------------------------------------------------------------------------------------
;         Interrupt occurs at 2.53 V ±0.1 V
;-----------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;*****************************************************************************************************/
void LVI_vdd(void)
{
        /* Power supply voltage < LVI detection voltage */
        if(LVIF == 1)
        {
                /* Light warning LED */
                LED_LVI = 0;

                /* Operate internal high-speed oscillator */
                Start_HispeedIntClk();

                /* Switch CPU clock to internal high-speed oscillato
                Change_CPUClk(SUBCLK_MAINSYSCLK);

                /* Set other interrupt (RTC, INTP0, INTP4) mask flag
                Set_IntMsk_RTC_INTP();

                /* Write display mode to flash memory */
                Write_ClockMode();

                /* Disable interrupt */
                DI();

                /* Clear other interrupt (RTC, INTP0, INTP4) ma
                Clear_IntMsk_RTC_INTP();

                /* Change CPU clock to subsystem clock (32.768
                Change_CPUClk(MAINSYSCLK_SUBCLK);

                /* Stop internal high-speed oscillator */
                Stop_HispeedIntClk();
        }

        /* Clear interrupt request flag */
        LVIIF = 0;
}
```

EEPROM emulation library header file

Display mode

If power supply voltage < LVI detection voltage, the voltage is considered to have dropped.



If the write block is full when write is executed using the EEPROM emulation library, data full is returned as the return value. In this case, change the write block.

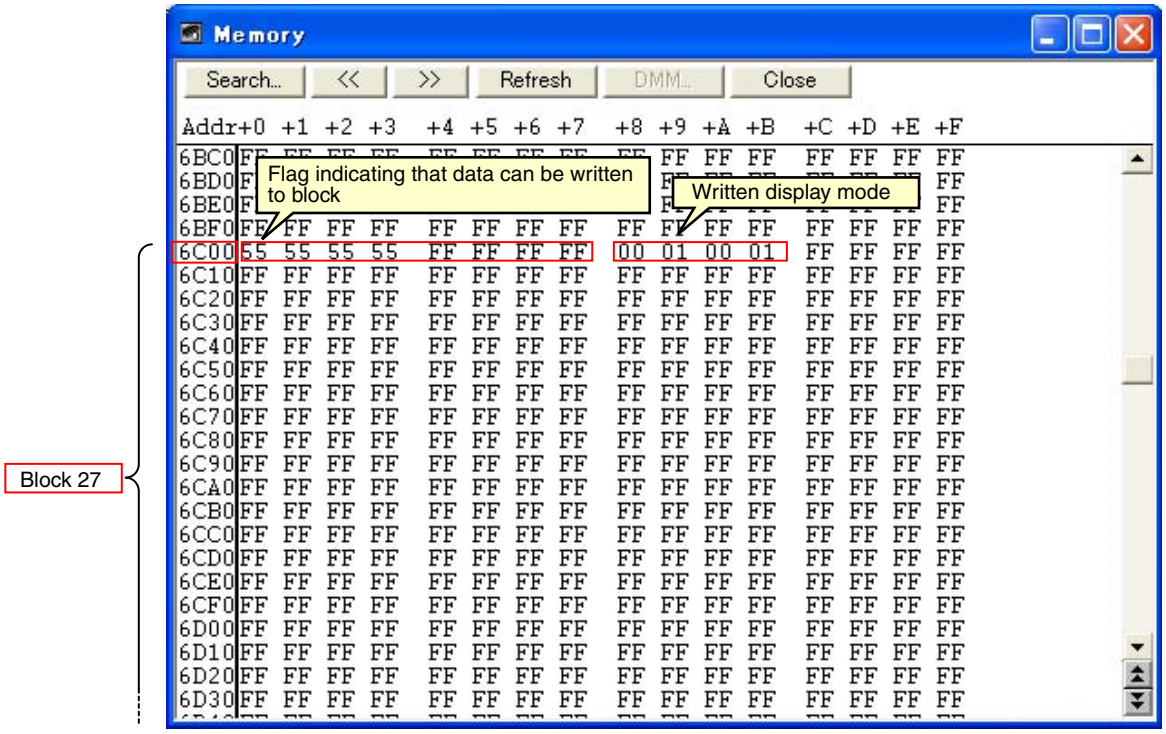
List 4-12. LVI Interrupt Servicing (Int_LVI.c) (2)

```
/************************************************************************************************
;         Write_ClockMode()
;------------------------------------------------------------------------------------------------
;         Write current display mode to flash memory
;------------------------------------------------------------------------------------------------
;         [I N]     -
;         [OUT]     -
;************************************************************************************************/
void Write_ClockMode(void)
{
         UCHAR temp;

         FLMDPUP = 1;                                          FLMDPUP settings
         EEPROM_Write(EEPROMData_No,Clock_Mode,&temp);
         FLMDPUP = 0;

}

/************************************************************************************************
;         EEPROM_Write()
;------------------------------------------------------------------------------------------------
;         Save data with EEPROM emulation library
;------------------------------------------------------------------------------------------------
;         [I N]     const UCHAR ucEEPROMNo, const UCHAR *ucEEPROMBuf, UCHAR *ChangeBlock_Count
;         [OUT]     UCHAR Result
;************************************************************************************************/
UCHAR EEPROM_Write( const UCHAR ucEEPROMNo, const UCHAR *ucEEPROMBuf, UCHAR *ChangeBlock_Count )
{
         /* Local variables */
         UCHAR     i;                  /* Loop counter */
         UCHAR     Result;             /* Return value of EEPROM emulation library */

         /* Write to flash memory with EEPROM emulation library */
         Result = ucEEPROMWriteEx_A( ucEEPROMNo, ucEEPROMBuf );

         /* If data full */
         if (Result == ERRFULL)                                If the block is full, change the block.
         {

                  /* Change block with EEPROM emulation library */
                  Result = ucEEPROMChangeEx_A();
                  if (( Result == EEE_NORMAL ) || ( Result == NMLBLK ))
                  {
                           /* Increment block change counter (counting from 0 to FF is possible) */
                           (*ChangeBlock_Count) = (*ChangeBlock_Count)  + 1;

                           /* Once block has been changed, write to new effective block with EEPROM
                              emulation library */
                           Result = ucEEPROMWriteEx_A( ucEEPROMNo, ucEEPROMBuf );

                           /* End in case of write error */
                           if ( Result != EEE_NORMAL )
                           {
                                    return( Result );
                           }

                           /* Erase previous effective block with EEPROM emulation library */
                           Result = ucEEPROMEraseEx_A();
                  }
         }
         return( Result );
}
```

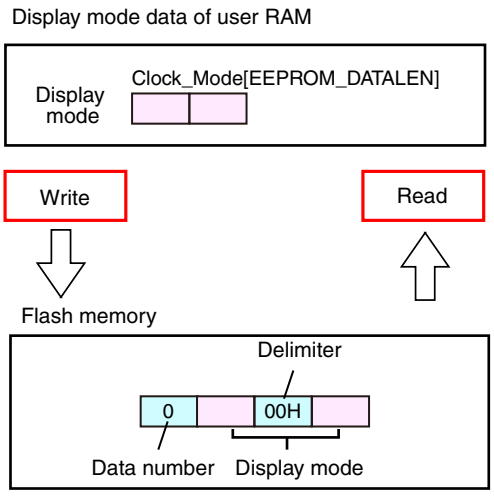
The flash memory written with the EEPROM emulation library is shown below (checked with MINICUBE2). Since the write block number is specified as 27 (1BH) by the Kx2_eee.h header file for the EEPROM emulation library, writing is done from address 6C00H. (In the case of the EEPROM emulation library, the 4 blocks from the specified block number are used as the data write area.)

In the case of the EEPROM emulation library, the write data is written after data numbers and delimiters have been added and the data has been adjusted to a length that is a multiple of 4 bytes, which is the smallest unit for flash memory write. If only 2 bytes are to be written, adding a data number and delimiter results in four bytes, so the length need not be adjusted to a multiple of 4. The data number is added at the front, and the delimiter (00H) is added one position at the penultimate byte.

Figure 4-6. Display Mode Written to Flash Memory



<Format of write data using EEPROM emulation>



CHAPTER 5 RELATED DOCUMENTS

Document Name		PDF/Document No.
RA78K0 Ver. 3.80 User's Manual[Note 1]	Operation	PDF
	Language	PDF
	Structured Assembly Language	PDF
RA78K0 Ver. 4.01 Operating Precautions (Notification Document)[Note 1]		ZUD-CD-07-0181-E
CC78K0 Ver. 3.70 User's Manual[Note 2]	Operation	PDF
	Language	PDF
CC78K0 Ver. 4.00 Operating Precautions (Notification Document)[Note 2]		ZUD-CD-07-0103-E
SM+ User's Manual	Operation	PDF
	User Open Interface	PDF
ID78K0-QB Ver. 2.94 User's Manual	Operation	PDF
ID78K0-QB Ver. 3.00 User's Manual	Operation	PDF
PM plus Ver. 5.20[Note 3] User's Manual		PDF
PM+ Ver. 6.30[Note 4] User's Manual		PDF

Notes 1. This document is installed into the PC together with the tool when installing RA78K0 Ver. 4.01. For descriptions not included in "RA78K0 Ver. 4.01 Operating Precautions (Notification Document)", refer to the user's manual of RA78K0 Ver. 3.80.

2. This document is installed into the PC together with the tool when installing CC78K0 Ver. 4.00. For descriptions not included in "CC78K0 Ver. 4.00 Operating Precautions (Notification Document)", refer to the user's manual of CC78K0 Ver. 3.70.

3. PM plus Ver. 5.20 is the integrated development environment included with RA78K0 Ver. 3.80.

4. PM+ Ver. 6.30 is the integrated development environment included with RA78K0 Ver. 4.01. Software tool (assembler, C compiler, debugger, and simulator) products of different versions can be managed.

Caution The related documents listed above are subject to change without notice. Be sure to use the latest version of each document for designing.

Published by: NEC Electronics Corporation (http://www.necel.com/)
Contact: http://www.necel.com/support/