致尊敬的顾客

关于产品目录等资料中的旧公司名称

NEC电子公司与株式会社瑞萨科技于2010年4月1日进行业务整合（合并），整合后的新公司暨“瑞萨电子公司”继承两家公司的所有业务。因此，本资料中虽还保留有旧公司名称等标识，但是并不妨碍本资料的有效性，敬请谅解。

瑞萨电子公司网址：http://www.renesas.com

2010年4月1日
瑞萨电子公司

【发行】瑞萨电子公司（http://www.renesas.com）

【业务咨询】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

应用笔记

78K0/Kx2-L

低功耗模式操作的设置

本应用手册描述 78K0/Kx2-L 微处理器的低功耗设置，使读者了解如何对这些微处理器实现低功耗。
提供低功耗操作所需的主要背景要求和对 78K0/Kx2-L 微处理器低功耗功能的概述。

目标设备

78K0/KY2-L 微控制器
78K0/KA2-L 微控制器
78K0/KB2-L 微控制器
78K0/KC2-L 微控制器

目录

文档编号 U19612CA1V0AN00 (第 1 版)
发行日期 2010 年 8 月

EEPROM 是 NEC 电子的注册商标。

- **本文档所登载的内容有效期截止至 2009 年 3 月，信息先于产品的生产周期发布。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。**
- 并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。
- 未经本公司事先书面许可，禁止复制或转载本文件中的内容。否则因本文档所登载内容引发的错误，本公司概不负责。
- 本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。
- 本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。
- 虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。
- 本公司产品质量分为：

  “标准等级”、“专业等级”以及“特殊等级”三种质量等级。

  “特殊等级”仅适用于为特定用途而根据用户指定的质量保证程序所开发的日电电子产品。另外，各种日电电子产品的推荐用途取决于其质量等级，详见如下。用户在选用本公司的产品时，请事先确认产品的质量等级。

  “标准等级”： 计算机，办公自动化设备，通信设备，测试和测量设备，音频·视频设备，家电，加工机械以及产业用机器人。

  “专业等级”： 运输设备（汽车、火车、船舶等），交通用信号控制设备，防灾装置，防止犯罪装置，各种安全装置以及医疗设备（不包括专门为维持生命而设计的设备）。

  “特殊等级： 航空器械，宇航设备，海底中继设备，原子能控制系统，为了维持生命的医疗设备、用于维持生命的装置或系统等。

  除在本公司半导体产品的数据表或数据手册等资料中另有特别规定以外，本公司半导体产品的质量等级均为“标准等级”。如果用户希望在本公司设计意图以外使用本公司半导体产品，务必事先与本公司销售代表联系以确认本公司是否同意为该项应用提供支持。

（注）

（1） 本声明中的“本公司”是指日本电气电子株式会社（NEC Electronics Corporation）及其控股公司。

（2） 本声明中的“本公司产品”是指所有由日本电气电子株式会社开发或制造的产品或为日本电气电子株式会社（定义如上）开发或制造的产品。

M5 02.11-1

第一章　概述

本应用手册描述 78K0/Kx2-L 微处理器的低功耗设置，使读者了解如何对这些微处理器实现低功耗。第 1 章提供低功耗操作所需的主要背景要求和对 78K0/Kx2-L 微处理器低功耗功能的概述。

目标设备：

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 低功耗背景要求和 78K0/Kx2-L 微处理器的特点

随着手持设备的广泛应用和安全设备市场的扩张，电池的长期寿命越来越成为主要技术指标。此外，随着环境意识的增加，对半导体和其他种类设备的低功耗要求日益提高。

78K0/Kx2-L 微处理器的开发填补了需要电池操作的领域、免维修操作领域和节能产品的需求。

下面介绍 78K0/Kx2-L 微处理器的低功耗的各个功能。

（1）待机功能

通过输入一个时钟信号操作微处理器的内部电路，操作期间电流在微处理器中通过。当时钟供应停止，内部的电路的操作和电流也停止。

通过程序执行待机指令，使时钟信号停止或者使时钟振荡器本身停止操作的功能叫做待机功能。

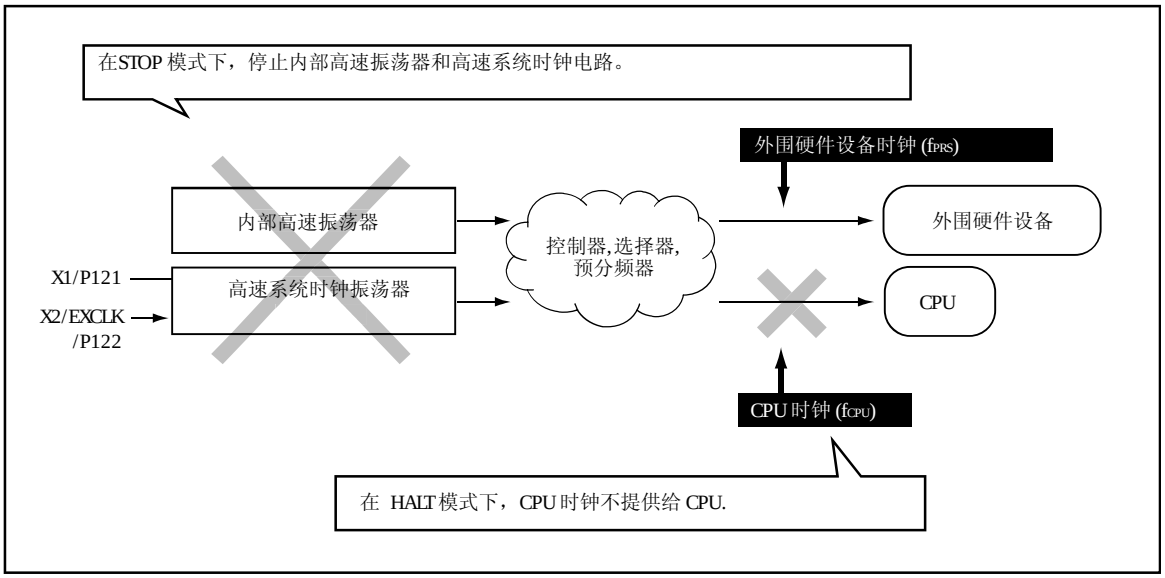
在不需要执行处理时，通过执行待机指令可以有效地降低功耗。例如在外部事件等待期间，可以将微处理器设置为待机状态（这种情况下，时钟信号流停止，或者时钟振荡器本身停止振荡，微处理器处于停止状态）。78K0/Kx2-L 微处理器有 2 个待机功能，HALT 模式和 STOP 模式。

STOP 模式： 通过执行 STOP 指令使微处理器进入这个模式。这个模式下时钟振荡器本身停止振荡。

HALT 模式： 通过执行 HALT 指令使微处理器进去这个模式。
在这个模式下时钟振荡器继续操作，但是提供给 CPU 的时钟是停止的。

功耗的降低水平顺序如下：正常操作模式 > HALT 模式 > STOP 模式。

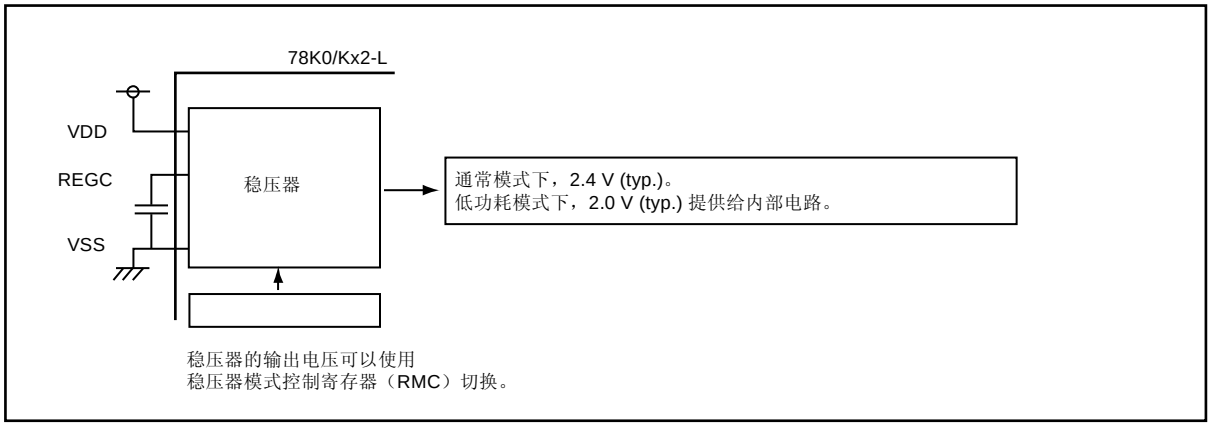
图 1-1. 待机功能和时钟发生器



（2）通过稳压器设置低功耗模式

每个 78K0/Kx2-L 微处理器集成了一个稳压电路，以使微处理器的外部电压为微处理器内部的电路提供稳定的电源。通过设置控制稳压器电路的稳压器模式寄存器（RMC），这个稳压器能够控制微处理器内部的电压输出。稳压器在正常模式下提供 2.4 V （typ.）的电压，在低功耗模式下提供 2.0 V （typ.）的电压 。即使当供电电压高于通过稳压器提供的电压时，功耗也是相同的。

图 1-2. 稳压器



[专栏]为进一步降低功耗，未使用引脚的处理

设置未使用的 I/O 端口为输入状态和断开这些端口的连接可能产生电流，进而产生额外的电能消耗。通过设置端口模式寄存器为输出和不连接这些端口，可以避免这个问题。对于仅输入端口的情况，通过上拉和下拉使通过这些端口的电流降低到最小。

第二章　时钟发生器和待机功能

待机功能控制时钟发生器。微处理器的功耗很大程度上受频率影响，此外还受到时钟是否运转或者停止的影响。

对时钟发生器时钟信号流的良好认识将有助于减少功耗,诸如在执行处理时选择最合适的时钟频率，而在不需要的时候停止这个时钟。

本章介绍 78K0/Kx2-L 处理器的时钟发生器和待机功能。

2.1　时钟发生器

（1） 复位释放后的时钟发生器

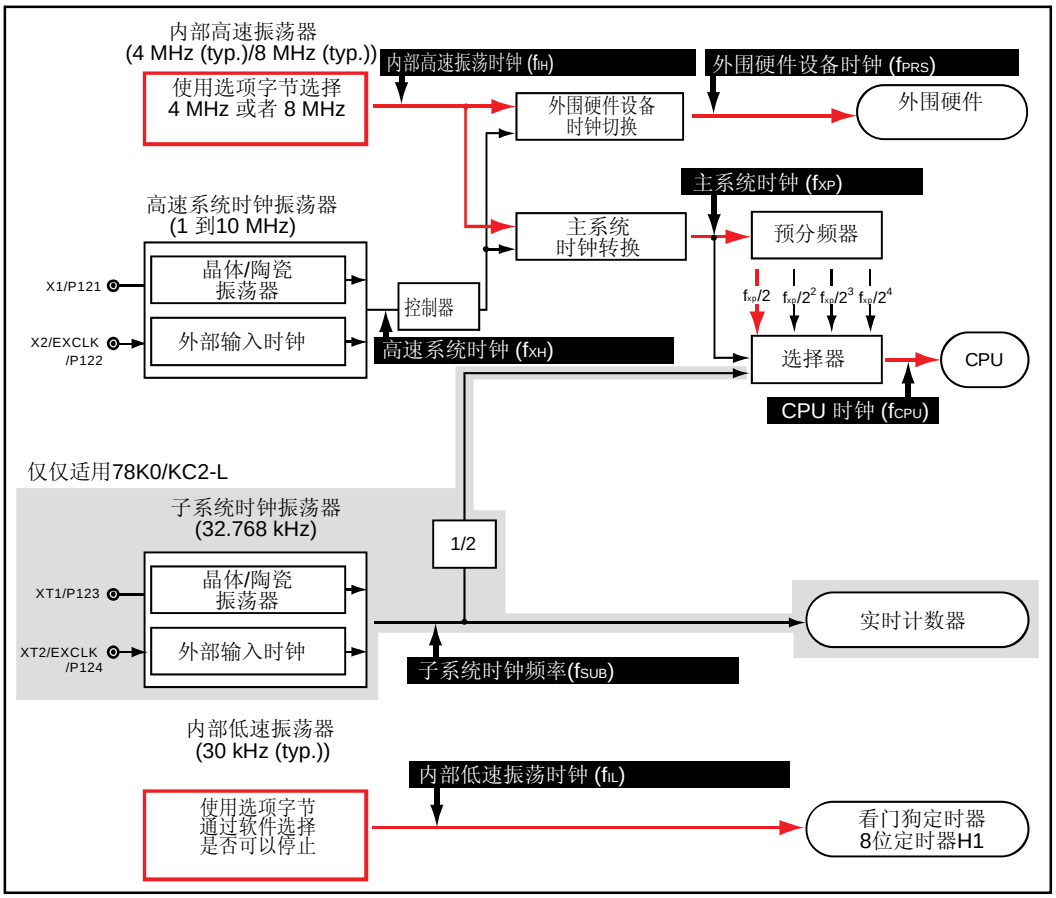
复位释放后，内部高速振荡器（8 MHz（典型值）或者 4 MHz（典型值）通过选项字节可以选择）和内部低速振荡器 （30 kHz（typ.））开始操作。

内部高速振荡器信号提供给主系统时钟，通过预分频器将其一分为二，主系统时钟（$f_{XP}/2$）供给 CPU 时钟（f_{CPU}）。

内部高速振荡器用作外部硬件设备的时钟源。

内部低速振荡器（30 kHz（typ.））用作看门狗定时器和定时器 H1 的时钟源。

图 2-1. 复位释放后时钟发生器 （红色框和线条指示操作）



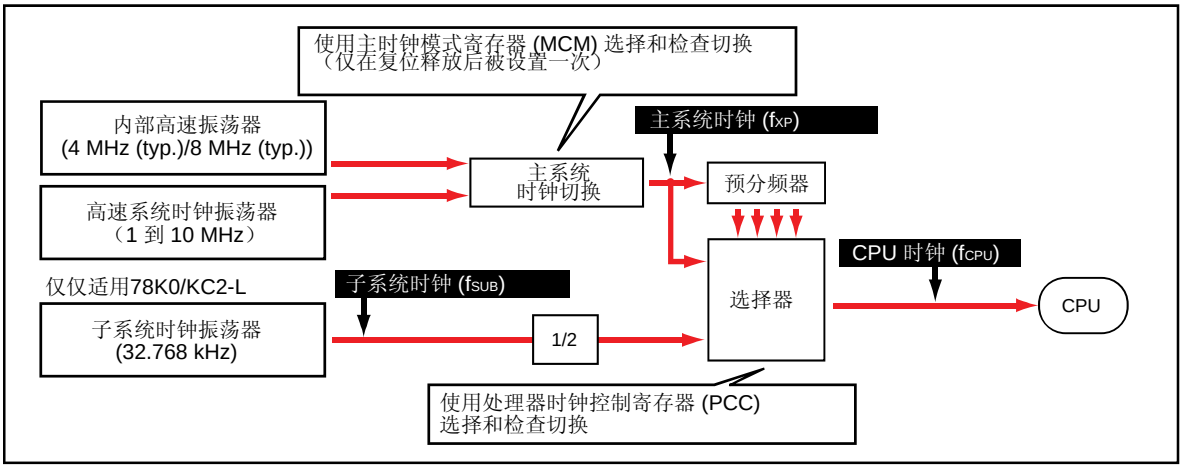
（2） CPU 时钟（f_{CPU}）流和时钟源的切换

CPU 时钟（f_{CPU}）是一个时钟信号，这个时钟信号提供给 CPU 用于读/执行指令，以及读/写内部寄存器和存储器。

供给 CPU 的 时钟（f_{CPU}）源可以从主系统时钟（不分频，或者通过预分频器进行 2, 4, 8, 或者 16 分频）和经过 2 分频后的副系统时钟（fSUB/2）之间选择。使用处理器时钟控制寄存器（PCC）切换 CPU 的时钟源。

主系统时钟（f_{XP}）源可以从内部高速振荡器切换为高速系统时钟振荡器。仅在复位释放后才可以执行切换。通过设置主时钟模式寄存器（MCM）完成切换。

图 2-2. 来自于每个时钟发生器 （红色框）的 CPU（f_{CPU}）时钟流和用于切换的寄存器



主时钟模式寄存器(MCM)　复位后: 0000 0000B

0	0	0	0	0	XSEL	MCS	MCM0

XSEL	MCM0	主系统时钟 (f_{XP}) 时钟源	外围硬件时钟 (f_{PRS})的时钟源
0	0	内部高速振荡器时钟 (f_{IH})（默认）	内部高速振荡时钟 (f_{IH})（默认）
0	1		
1	0		高速系统时钟 (f_{XH})
1	1	高速系统时钟(f_{XH})	

MCS	主系统时钟 (f_{XP}) 状态 （时钟源）
0	内部高速振荡时钟 (f_{IH}) （默认）
1	高速系统时钟 (f_{XH})

处理器时钟控制寄存器(PCC)　复位后: 0000 0001B

0	★ XSTART	★ CLS	★ CSS	0	PCC2	PCC1	PCC0

★ 仅仅适用78K0/KC2-L

CSS	PCC2	PCC1	PCC0	CPU 时钟 (f_{CPU}) 时钟源
1	0	0	0	主系统时钟, 不分频(f_{XP})
	0	0	1	主系统时钟经过2分频 ($f_{XP}/2$)（默认）
	0	1	0	主系统时钟经过2^2 分频($f_{XP}/2^2$)
	0	1	1	主系统时钟经过2^3 分频($f_{XP}/2^3$)
	1	0	0	主系统时钟经过2^4 分频($f_{XP}/2^4$)

除了以上: 设置禁止

CSS	PCC2	PCC1	PCC0	CPU时钟 (f_{CPU}) 时钟源
0	0	0	0	子系统时钟经过2分频 ($f_{SUB}/2$)
	0	0	1	
	0	1	0	
	0	1	1	
	1	0	0	

除了以上: 设置禁止

CLS	CPU 时钟 (f_{CPU}) 状态 （时钟源）
0	主系统时钟 (f_{XP}) （默认）
1	子系统时钟经过2分频($f_{SUB}/2$)

当切换主系统时钟（f_{XP}）或者 CPU 时钟（f_{CPU}）源时，切换之前的时钟将持续提供几个时钟周期的振荡。

在主系统时钟的情况下，切换是否完成可以使用主时钟模式寄存器（MCM）的 MCS 位检测。

如果 CPU 时钟（f_{CPU}）已经从主系统时钟（f_{XP}）切换到经过 2 分频的副系统时钟（$f_{SUB}/2$），或者反之亦然，这个切换是否完成可以使用处理器控制寄存器（PCC）的 CLS 位来检测。

如果通过改变处理器时钟控制寄存器（PCC）的设置已经切换了主系统时钟的分频比例，所需要的一个等待时间需满足如下所示的时间要求。

表 2-1. 当已经切换主系统时钟的分频比例时转换 CPU 时钟所需的最大时间

切换前的分频比例	切换后的分频比例				
	未分频	1/2	$1/2^2$	$1/2^3$	$1/2^4$
没有划分		16 个时钟	16 个时钟	16 个时钟	16 个时钟
1/2	8 个时钟		8 个时钟	8 个时钟	8 个时钟
$1/2^2$	4 个时钟	4 个时钟		4 个时钟	4 个时钟
$1/2^3$	2 个时钟	2 个时钟	2 个时钟		2 个时钟
$1/2^4$	1 个时钟	1 个时钟	1 个时钟	1 个时钟	

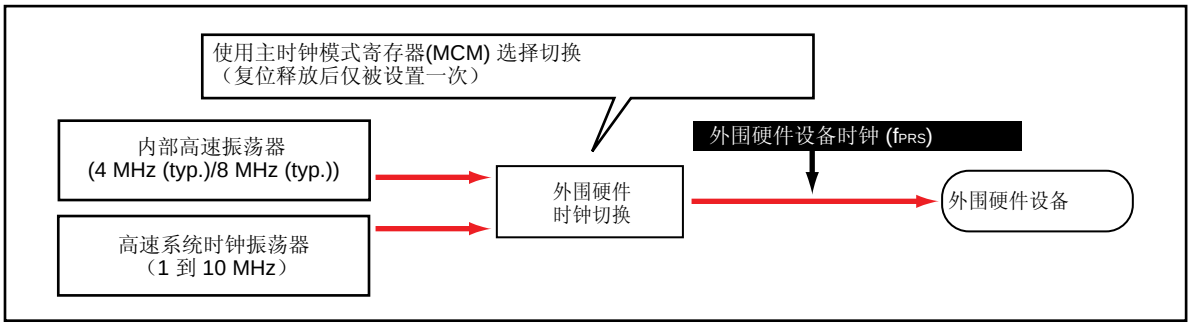
备注　时钟的个数是切换之前的 CPU 时钟 （f_{CPU}）的个数。

（3） 外围硬件设备时钟流和时钟源的切换

外部硬件设备时钟提供给外部硬件设备，使用它作为操作时钟。

外围硬件设备的时钟源可以从内部高速振荡器切换到高速系统时钟振荡器。复位释放后仅能执行一次切换。通过设置主时钟模式寄存器（MCM）可以完成切换。（关于主系统时钟模式寄存器（MCM）的详细情况请参考图 **2-2**.）

图 2-3. 来自于每个时钟发生器 （红线）的外围硬件设备时钟（f_{PRS}）流



（4） 停止和启动时钟发生器

在时钟发生器的信号没有提供到任何地方的情况下，或者目标功能没有被使用，时钟发生器将被停止（如果通过一个选项字节设置了“可以通过软件停止”，则可以停止内部低速振荡器）。一个时钟发生器被停止之前，必须检测它的信号是否提供给当前的操作功能（CPU，外围硬件设备）。

当被停止的时钟发生器的操作再一次启动时，必须经过等待振荡稳定时间之后，再将它的信号提供给 CPU 时钟，外围硬件设备时钟，实时计数器时钟。

<1> 内部高速振荡器和内部低速振荡器

使用内部振荡模式寄存器（RCM）启动或者停止内部振荡器的振荡。可以使用 RSTS 位检测内部高速振荡器的振荡稳定。

图 2-4. 内部振荡模式寄存器

内部振荡模式寄存器 (RCM) 复位后: 1000 0000B

RSTS	0	0	0	0	0	LSRSTOP	RSTOP

LSRSTOP	内部低速振荡器振荡/停止
0	内部低速振荡器振荡（默认）
1	内部低速振荡器停止

RSTOP	内部高速振荡器振荡/停止
0	内部高速振荡器振荡 （默认）
1	内部高速振荡器停止

RSTS	内部高速振荡器状态
0	内部高速振荡器的精确的稳定等待
1	内部高速振荡器的稳定操作（默认）

注意事项 复位之后 RSTS 位为 0，内部高速振荡器已经稳定之后变为 1。

<2> 高速系统时钟振荡器

使用主 OSC 控制寄存器（MOC）的 MSTOP 位来启动或者停止高速系统时钟振荡器的振荡。

在 X1 振荡模式的情况下，振荡稳定时间的等待可以使用 X1 振荡稳定时间计数器测量。稳定振荡时间设置在振荡稳定时间选择寄存器（OSTS）中，是否经过了振荡稳定时间可以使用振荡稳定时间计数状态寄存器（OSTC）进行检测。根据所使用的振荡器类型设置一个有效的时间长度。（X1 振荡稳定时间计数器仅能测量已经设置的时间，不能检测 X1 振荡器是否稳定。）

振荡稳定时间选择寄存器（OSTS）在复位释放后设置为最长的测量时间（$2^{16}/f_X$）。

图 2-5. 主 OSC 控制寄存器

主 OSC 控制寄存器 (MOC) 复位后: 1000 0000B

MSTOP	0	0	0	0	0	0	0

MSTOP	高速系统时钟操作控制	
	X1 振荡模式	外部时钟输入模式
0	X1 振荡操作	使能EXCLK 引脚的外部时钟
1	X1 振荡器停止（默认）	禁止EXCLK 引脚的外部时钟（默认）

图 2-6. X1 振荡稳定时间计数器的寄存器

振荡稳定时间寄存器 (OSTS) 复位后: 0000 0101B

0	0	0	0	0	OSTS2	OSTS1	OSTS0

OSTS2	OSTS1	OSTS0	振荡稳定时间选择	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	$2^{16}/f_X$ （默认）	6.55 ms

除了以上: 设置禁止

振荡稳定时间计数状态寄存器 (OSTC) 复位后: 0000 0000B

0	0	0	MOST11	MOST13	MOST14	MOST15	MOST16

MOST11	MOST13	MOST14	MOST15	MOST16	振荡稳定时间选择	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.

→ 经过了设定的时间后变为1

<3> 副系统时钟振荡器 （仅适用 78K0/KC2-L ）

使用处理器时钟控制寄存器（PCC）的每一位和时钟操作模式选择寄存器（OSCCTL）启动或者停止副系统时钟振荡器的振荡。

在 XT1 振荡模式下的启动，必须根据振荡器的使用情况来等待稳定时间。

图 2-7. 副系统时钟振荡器的寄存器

处理器时钟控制寄存器 (PCC) 复位后: 0000 0001B

0	★ XSTART	★ CLS	★ CSS	0	PCC2	PCC1	PCC0

★ 仅仅适用78K0/KC2-L

时钟操作模式选择寄存器 (OSCCTL) 复位后: 0000 0000B

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

★ 仅仅适用78K0/KC2-L

XSTART	EXCLKS	OSCSELS	子系统时钟操作模式
0	0	1	XT1 振荡模式 (晶体或者陶瓷振荡器必须连接到 P123/XT1 和 P124/XT2)
1	×	×	
0	1	1	外部时钟输入模式（外部时钟必须输入到 P124/EXCLKS）

要停止副系统时钟，设置 OSCSELS 为 0（默认=停止） x：无关

RSWOSC	AMPHXT	XT1 振荡器振荡模式选择
0	0	低功耗振荡（默认）
0	1	正常振荡
1	×	超低功耗振荡

×：无关

CLS	CPU 时钟 (f_{CPU}) 状态 （时钟源）
0	使用主系统时钟 (f_{XP})操作
1	使用子系统时钟 (f_{SUB})操作

2.2 待机功能

使用待机功能，以程序的任意时序来控制时钟以减少功耗，通过执行 HALT 指令或者 STOP 指令进入各自的 HALT 模式或者 STOP 模式。通过发出一个非屏蔽中断请求信号或者复位信号可以从 HALT 模式或者 STOP 模式返回到正常的操作模式。

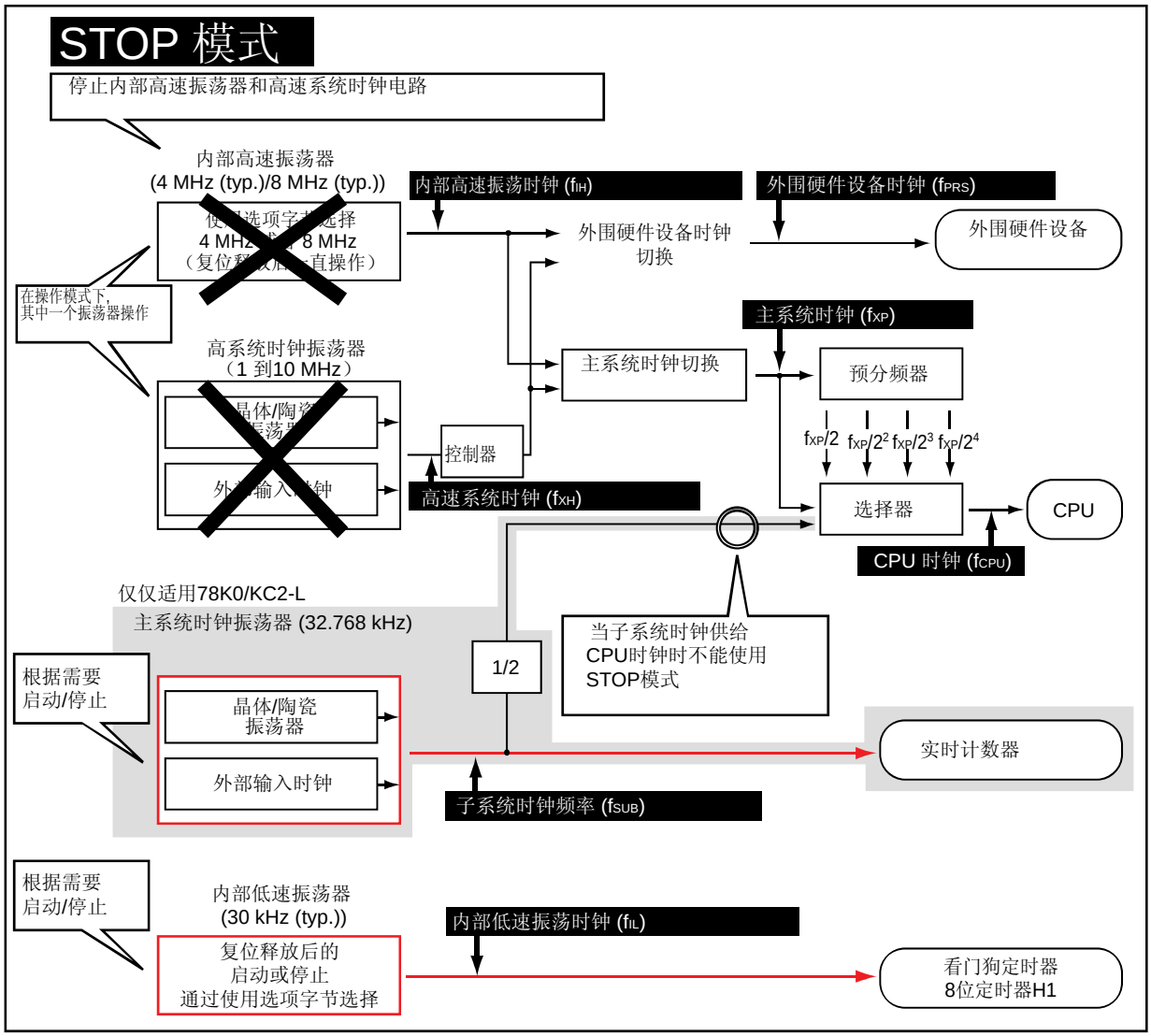
HALT 模式不能像 STOP 模式那样减少那么多的操作电流，但是，因为时钟正在振荡，因而返回到正常操作模式的时间很短。与 HALT 模式相比，STOP 模式可以使操作电流很大程度的减少，但是时钟停止振荡，返回到正常操作模式花费的时间比较长。

待机使用哪一种模式，要根据间歇式的程序操作频率以及功能的消耗要求来决定。有关工作电流的比较，请参考 **2.3 待机功能的总电流的对比**，有关返回时间的比较，**请**参考 **2.4 待机功能的中断返回时间的对比**。

（1） STOP 模式

当执行 STOP 指令时，将停止内部高速振荡器和高速系统时钟振荡器。如果副系统时钟振荡器和内部低速振荡器正在操作，则这个操作继续。

图 2-8. 在 STOP 模式下的时钟发生器



<1> CPU 时钟（f_{CPU}）条件

仅当主系统时钟（f_{XP}）选择作为 CPU 时钟（f_{CPU}）时，才能够进入 STOP 模式。因为即时执行了 STOP 指令，也不能停止副系统时钟 （f_{SUB}: 仅 78K0/KC2-L），所以当 CPU 时钟选择副系统时钟时，不能进入 STOP 模式。

<2> CPU 和外围硬件设备状态

因为内部高速振荡器（f_{IH}）和高速系统时钟振荡器（f_{XH}）被停止，因而 CPU 停止。

使用外围硬件设备时钟（f_{PRS}）作为时钟源的外围硬件设备的功能被停止。然而，使用从一个引脚输入的外部时钟，内部低速振荡器时钟（f_{IL}），或者副系统时钟（f_{SUB}）作为在外围硬件设备的时钟源的情况下，将保持 STOP 指令执行之前的状态（如果他们正在操作，则继续操作）。在执行 STOP 指令之前，如果使用外围硬件设备时钟（f_{PRS}）的外围硬件设备正在操作，则必须被停止。

CPU 的各种寄存器，RAM 和外围硬件设备将保持执行 STOP 指令之前的状态。

如果通过选项字节的设定选择了“内部低速振荡器可以通过软件停止”，从内部低速振荡器提供给看门狗定时器的时钟停止。

表 2-2. STOP 模式下的外围硬件设备

外围硬件设备		STOP 模式下的状态
16 位定时器/事件计数器 00		操作停止
8 位定时器/事件计数器	50	仅当 TI50 选择作为计数时钟可操作
	51	仅当 TI51 选择作为计数时钟可操作
8 位定时器	H0	仅当 TM50 的输出选择为 8 位定时器/事件计数器 50 操作期间的计数时钟可操作
	H1	仅当 f_{IL}, $f_{IL}/2^6$, 或 $f_{IL}/2^{15}$ 选择作为计数时钟时可操作
实时计数器		可操作
看门狗定时器		可操作。通过选项字节设置“内部低速振荡器可以通过软件停止”时，提供给看门狗定时器的时钟停止。
时钟输出		仅当副系统时钟选择作为计数时钟时可操作
A/D 转换器		操作停止
运算放大器 0，1		操作停止
串行接口	UART6	仅当 TM50 的输出选择为 8 位定时器/事件计数器 50 操作期间的计数时钟时可操作
	CSI10, CSI11	仅当外部时钟选择作为串口时钟时可操作
	IICA	通过地址匹配唤醒（中断信号产生）时可操作
按键中断		可操作
上电清零功能		
低电压检测功能		
外部中断		

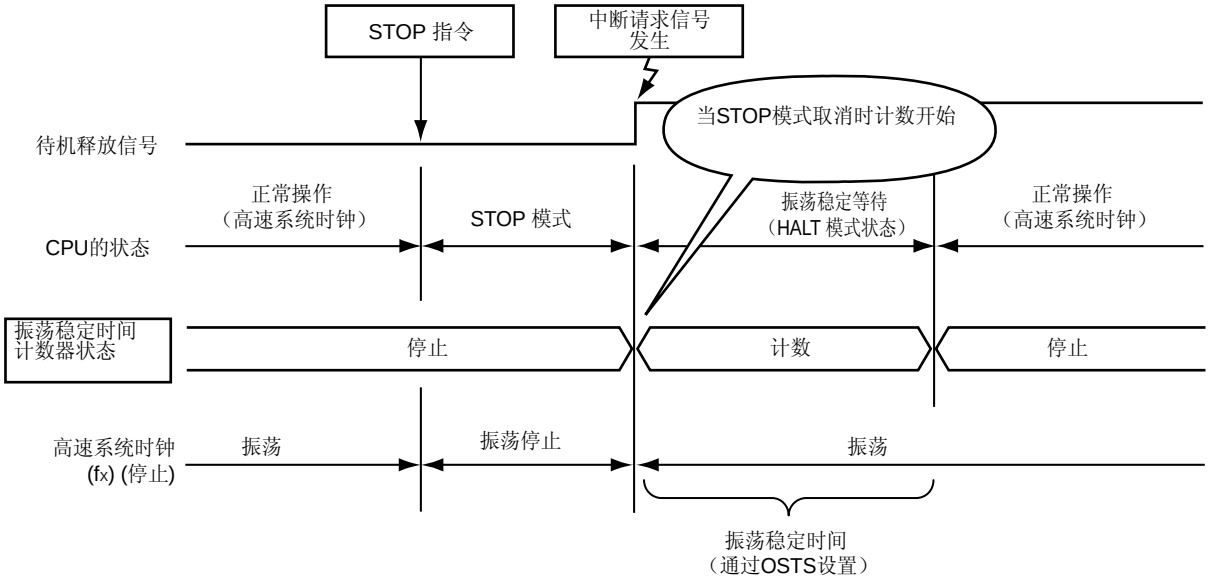
<3> 从 STOP 模式返回振荡稳定时间计数器的操作

时钟发生器包括一个用于测量 X1 振荡器的稳定时间的计数器。如果高速系统时钟的 X1 振荡器处于振荡状态，通过设置振荡稳定时间选择寄存器（OSTS）和读取振荡稳定时间计数器状态寄存器（OSTC）可以测量振荡稳定时间。

当 X1 振荡器开始振荡（MSTOP = 0）时 OSTC 寄存器开始计数，此时，从内部高速振荡器或者副系统时钟振荡器为 CPU 提供时钟（f_{CPU}）。

然而，仅在下列情况下计数立即开始：当执行了 STOP 指令，并且由于一个非屏蔽中断请求信号有效发生，从 STOP 模式返回后，而此时从高速系统时钟的 X1 振荡器为 CPU 时钟（f_{CPU}）提供时钟源。执行计数期间，此时的执行状态是在时钟源没有提供给 CPU 时钟（f_{CPU}）期间的 HALT 状态，在测量到 OSTS 寄存器设置的时间之后，返回到操作模式（正常状态）。因此，执行 STOP 指令之前为振荡稳定时间选择寄存器（OSTS）设置一个估值。

图 2-9. 当 STOP 指令被取消时 X1 振荡稳定定时计算器的操作



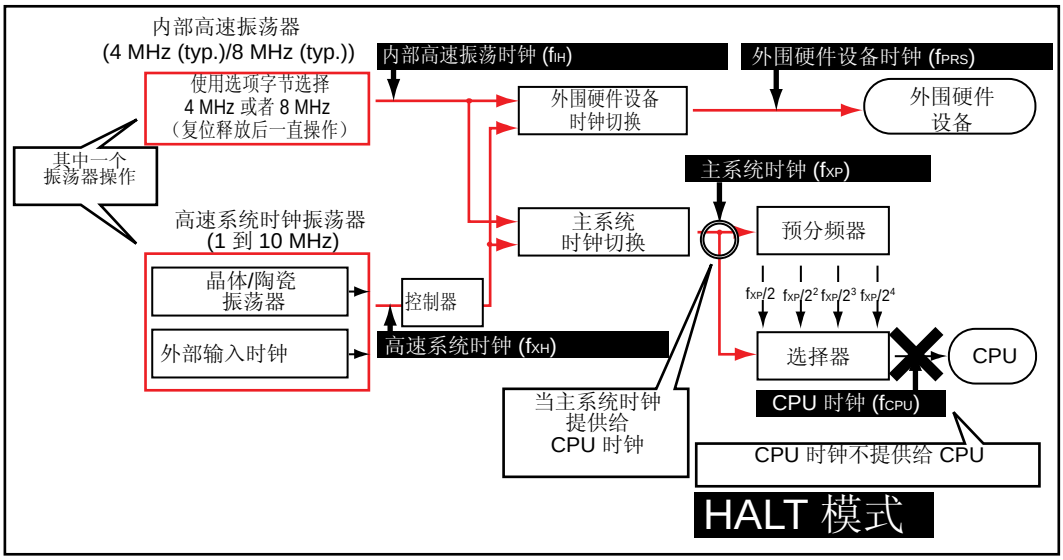
（2） HALT 模式

当 HALT 指令执行时，如果各种振荡器正在操作，则操作将继续，而供给 CPU 的时钟（f_{CPU}）停止 。

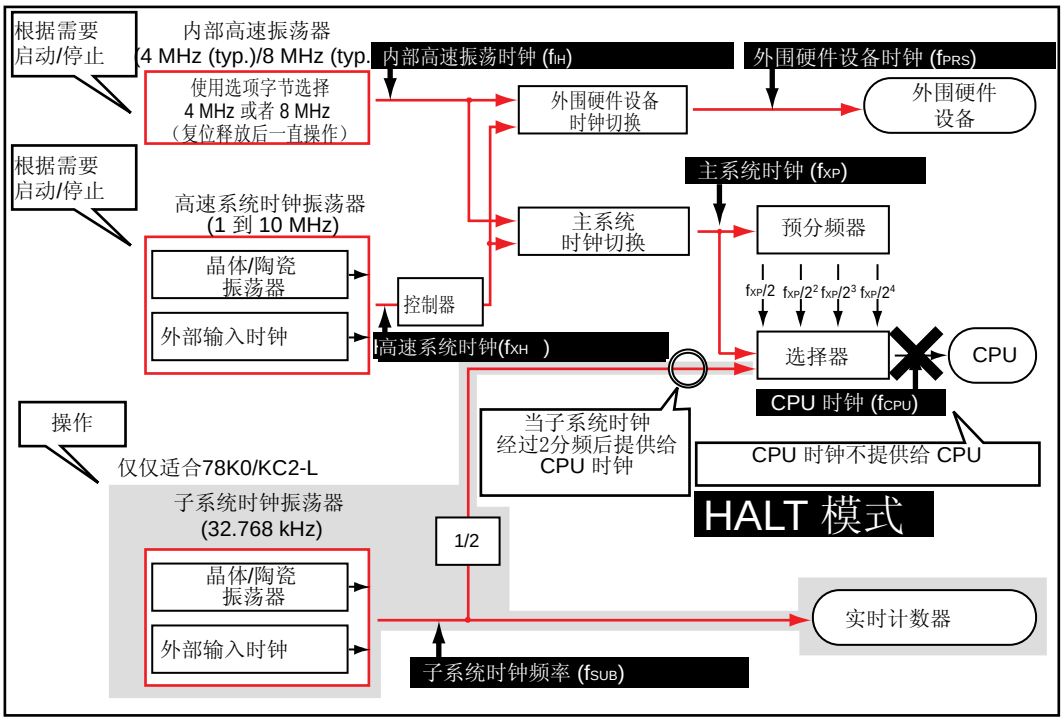
当主系统时钟（f_{XP}）或者 2 分频后的副系统时钟（$f_{SUB}/2$）作为 CPU 时钟（f_{CPU}）时，能够进入 HALT 模式。

图 2-10. 在 HALT 模式下的时钟发生器 （内部低速振荡器可以忽略）

当主系统时钟选择作为 CPU 时钟



当经过2分频后的子系统时钟选择作为 CPU 时钟 （只适合78K0/KC2-L）



<1> CPU 时钟条件

当主系统时钟（f_{XP}）或者副系统时钟经过 2 分频（$f_{SUB}/2$）作为 CPU 时钟（f_{CPU}）时，能够进入 HALT 模式。

当副系统时钟经过 2 分频（$f_{SUB}/2$）作为 CPU 时钟（f_{CPU}）时， 在 HALT 模式下通过停止内部高速振荡器和高速系统振荡器（参考 表 **2-5**）可以在很大程度上降低功耗。

<2> CPU 和外围硬件设备状态

CPU 停止，因为 CPU 时钟（f_{CPU}）不提供给 CPU。

保持 HALT 指令执行之前的状态，因为外围硬件设备时钟（f_{PRS}）提供给外围硬件设备。（如果外围硬件设备正在操作，则外围硬件设备将继续操作。）

保持 HALT 指令执行之前的 CPU 的各种寄存器， RAM 和外围硬件设备的状态。

如果通过选项字节的设置选择了“内部低速振荡器能够通过软件停止”，则从内部低速振荡器提供给看门狗定时器的时钟停止。

（3） 通过非屏蔽中断返回 （STOP 模式和 HALT 模式）

如果通过非屏蔽中断请求信号返回，中断使能信号（IE）有效（IE = 1），则执行中断服务程序，如果无效（IE = 0），则不执行中断服务程序，而是执行下一条指令。在这种情况下，中断请求信号不能自动清除而是保持挂起状态，所以当中断使能有效时（IE = 1），将执行中断服务程序。（如果在中断向量对应的地址上没有程序，则会产生一个意外的程序循环。）

如果通过一个非屏蔽中断从 STOP 模式或者 HALT 模式返回而不需要执行中断服务，则在返回之后中断有效（IE = 1）之前一定要清除中断请求标志。

表 2-3. 响应中断请求返回操作 （STOP 和 HALT 模式）

释放资源	MKXX（屏蔽标志）	PRXXX（优先级指定标志）	IE	ISP	操作
可屏蔽中断请求	0（允许）	0（高）	0	×	执行下一个地址（保持中断请求标志）
			1	×	执行中断服务程序
		1（低）	0	1	执行下一个地址（保持中断请求标志）
			×	0	
			1	1	执行中断服务程序
	1（禁止）	×	×	×	保持 STOP 和 HALT 模式
复位	–	–	×	×	复位处理

[专栏] 外部硬件设备的事件等待期间 HALT 模式的使用

如果外围硬件设备事件发生时，例如通过软件轮询检测到 A/D 转换器的转换等待，应该考虑使用 HALT 模式。通过中断从 HALT 模式返回的时间有时候比轮询速度快。此外，因为微处理器在轮流检测期间持续执行指令，因此使微处理器处于 HALT 模式的待机模式中将减少功率的消耗。

2.3 待机功能的总电流比较

下面表格中列出了在各种情况下，内部供电（V_{DD}, AV_{REF}）电路分别在正常操作，HALT 模式，和 STOP 模式下流过的总电流（通过稳压器设置）。

表 2-4. 从内部高速振荡器时钟 （f_{IH}） 或者高速系统时钟 （f_{XH}） 提供给 CPU 时钟 （f_{CPU}）

操作条件	操作模式 （正常操作）	HALT 模式	STOP 模式
f_{IH} = 4 MHz （内部高速振荡器时钟）， V_{DD} = 3.0 V RMC = 56H （低功耗模式）	0.5 mA（TYP.）， 1.4 mA（MAX.）	0.2 mA（TYP.）， 0.5 mA（MAX.）	0.3 μA（TYP.）， 5.5 μA（MAX.） （时钟发生器停止，所以频率不包括在条件中）
f_{IH} = 8 MHz （内部高速振荡器时钟）， 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 （外部输入时钟）， 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 （晶体/陶瓷振荡器）， V_{DD} = 5.0 V RMC = 00H	2.3 mA（TYP.）， 3.9 mA（MAX.）	1.0 mA（TYP.）， 2.4 mA（MAX.）	–

表 2-5. 从副系统时钟经过 2 分频 （f_{SUB}/2） （仅 78K0/KC2-L ）提供给 CPU 时钟 （f_{CPU}）

操作条件	操作模式 （正常模式）	HALT 模式	STOP 模式
f_{SUB} = 32.768 MHz （晶体/陶瓷振荡器）， V_{DD} = 3.0 V RMC = 56H （低功耗模式）	3 μA（TYP.）， 9.7 μA（MAX.）	0.8 μA（TYP.）， 6.7 μA（MAX.）	0.3 μA（TYP.）， 5.5 μA（MAX.） （时钟发生器停止，所以频率不包括在条件内）

注意事项

1. **表 2-4 和 2-5 摘录于 78K0/Kx2-L 用户手册的第 28 章的电气规格。**
2. **表 2-4 和 2-5 的电流值包括当输入引脚的电平为 V_{DD} 或 V_{SS} 时的输入漏电流。然而，不包括流入到上拉电阻，下拉电阻，和端口的输出电流。**
3. **正常模式和 HALT 模式的电流值不包括流入振荡器的电流，除产生提供给 CPU 的时钟电路之外。不包括流入到 LVI 电路，A/D 转换器，运算放大器，看门狗定时器，实时计数器和 8 位 H1 定时器（当使用 30 kHz 内部低速振荡时钟作为计数时钟）的电流。**
4. **STOP 模式的电流值不包括流入 LVI 电路，A/D 转换器，运算放大器，看门狗定时器，实时计数器和 8 位 H1 定时器（当使用 30 kHz 内部低速振荡时钟作为计数时钟）的电流。**
5. **STOP 模式的电流值条件为 V_{DD} = 3.0 V，RMC = 56H。时钟发生器停止，所以频率不包括在这些条件中。**

2.4 待机功能通过中断返回的时间比较

通过一个非屏蔽中断请求信号返回的情况下，从 HALT 模式返回和从 STOP 模式返回的返回时间不同。这是因为在 HALT 状态下，内部高速振荡器和高速系统时钟振荡器操作，而在 STOP 状态下，它们停止操作。

当发生中断请求信号时，释放待机释放信号（内部信号：H = 释放状态），从每个模式有效的返回。

从 STOP 模式返回的情况下，返回时间的不同依赖于使用的时钟（内部高速振荡器（f_{IH}），高速系统时钟振荡器（ f_X 和 f_{EXCLK}），副系统时钟振荡器 2 分频（$f_{SUB}/2$））提供给 CPU 时钟（f_{CPU}）。（当由高速系统时钟振荡器的 X1 振荡器提供时返回时间最长。）从 HALT 模式返回的情况下，返回的时间是相同的而与提供给 CPU 时钟（f_{CPU}）的时钟源无关

除另有规定外，在下面表格中的值为 RMC = 00H （固定为 2.4 V 供电）。

（1）当 CPU 时钟 （f_{CPU}）由内部高速振荡时钟 （f_{IH}） 提供时

图 2-11. 通过中断请求信号返回（$f_{CPU} \leftarrow f_{IH}$）

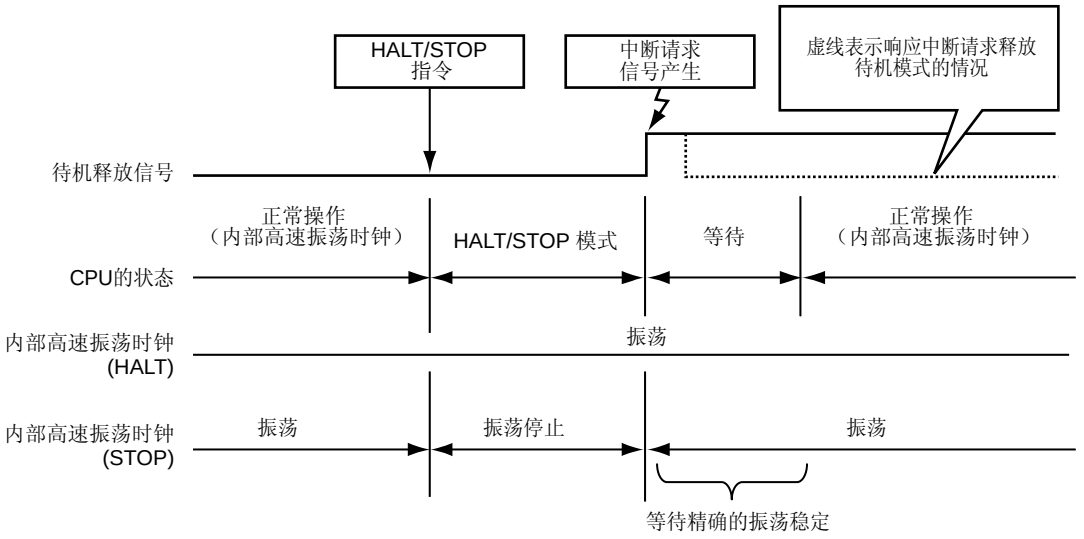


表 2-6. 从 HALT 和 STOP 模式返回的返回时间（$f_{CPU} \leftarrow f_{IH}$）

资源		HALT 模式 （f_{CPU} = 8 MHz）	STOP 模式 （f_{CPU} = 8 MHz）
等待	当执行向量中断服务时	11 或 12 个时钟 （1.375 到 1.5 μs）	17 或 18 个时钟 （2.125 到 2.25 μs）
	当不执行向量中断服务时	4 或 5 个时钟 （1.375 到 1.5 μs）	11 或 12 个时钟 （1.375 到 1.5 μs）
等待精确的振荡稳定		–	102 到 407 μs（RMC = 00H） 120 到 481 μs（RMC = 56H）

（2） 当 CPU 时钟（f_{CPU}）由高速系统时钟的 X1 的振荡器提供时

如果 CPU 时钟（f_{CPU}）由 X1 晶体/陶瓷振荡器提供，并有效地从 STOP 模式返回时，系统将等待用户根据振荡器设置在 OSTS 寄存器中的振荡稳定时间。在这个期间，微处理器在 HALT 模式状态，没有时钟提供给 CPU 时钟（f_{CPU}）。

图 2-12. 通过中断请求信号返回（$f_{CPU} \leftarrow f_{XH} \leftarrow f_X$）

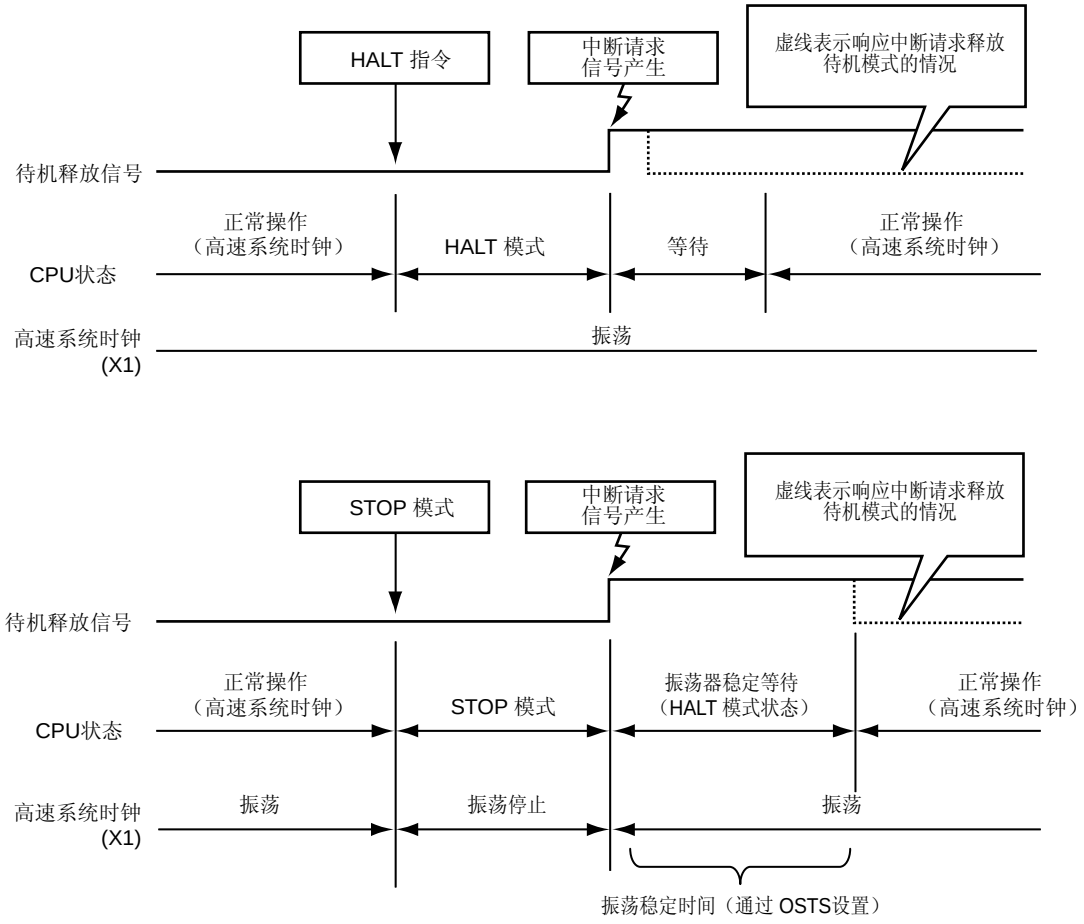


表 2-7. 从 HALT 和 STOP 模式返回的返回时间（$f_{CPU} \leftarrow f_{XH} \leftarrow f_X$）

资源		HALT 模式 （f_{CPU} = 10 MHz）	STOP 模式 （f_{CPU} = 10 MHz）
等待	当执行向量中断服务时	11 或 12 个时钟 （1.1 到 1.2 μs）	–
	当不执行向量中断服务时	4 或 5 个时钟 （0.4 到 0.5 μs）	–
振荡稳定时间（通过 OSTS 设置）		–	等待时间设置到 OSTS 寄存器 $2^{11}/f_X$（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） 当 CPU 时钟（f_{CPU}）由高速系统时钟的的 X2 外部时钟（f_{EXCLK}）提供时

图 2-13. 通过中断请求信号返回（$f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$）

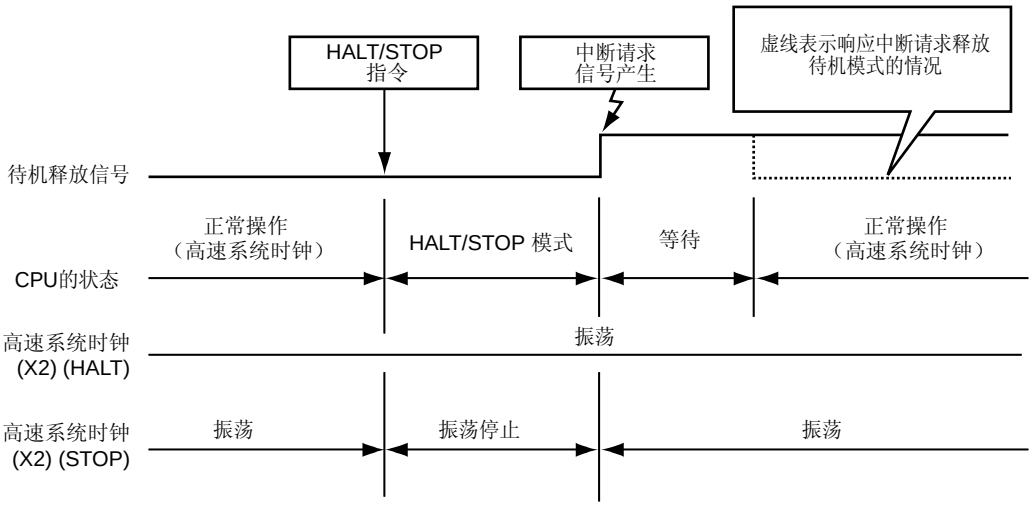


表 2-8. 从 HALT 模式和 STOP 模式返回的返回时间（$f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$）

资源		HALT 模式 （f_{CPU} = 8 MHz）	STOP 模式 （f_{CPU} = 8 MHz）
等待	当执行向量中断服务时	11 或 12 个时钟 （1.375 到 1.5 μs）	17 或 18 个时钟 （2.125 到 2.25 μs）
	当不执行向量中断服务时	4 或 5 个时钟 （1.375 到 1.5 μs）	11 或 12 个时钟 （1.375 到 1.5 μs）

（4） 当 CPU 时钟（f_{CPU}）由副系统时钟提供时 （$f_{SUB}/2 \leftarrow f_{XT}$, f_{EXCLKS}）

只有 XT1 输入（f_{XT}: 晶体振荡器/陶瓷振荡器）和 XT2 输入（f_{EXCLKS}：外部时钟）允许 HALT 指令。（STOP 指令禁止执行。）返回时序图和返回时间与从 HALT 指令返回的情况是一样的，（1）到（3）。

2.5 待机功能通过复位返回

如果通过复位信号的产生返回，在复位期间所有的时钟发生器停止，所以返回时间与在 STOP 状态和 HALT 状态下是一样的。

（1） 通过复位信号返回

如果通过复位信号的产生返回，在复位期间所有的时钟发生器停止，所以返回时间与在 STOP 状态和 HALT 状态下是一样的。

<1> 当 CPU 时钟 clock（f_{CPU}）由内部高速振荡时钟（f_{IH}）提供时

图 2-14. 通过复位信号返回（$f_{CPU} \leftarrow f_{IH}$）

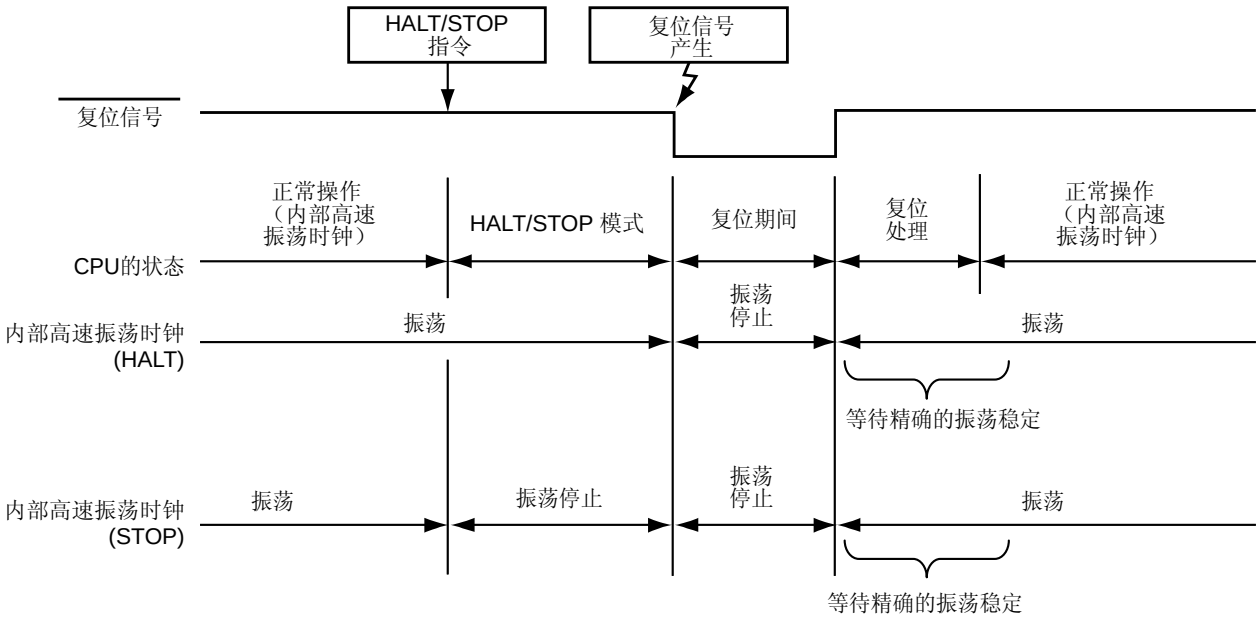


表 2-9. 从 HALT 模式和 STOP 模式返回的返回时间（$f_{CPU} \leftarrow f_{IH}$）

资源	HALT 模式/STOP 模式
复位处理	12 到 51 μs
等待精确稳定的振荡	102 到 407 μs

<2> 当 CPU 时钟（f_{CPU}）由高速系统时钟的 X1 振荡器提供时

图 2-15. 通过复位信号返回（$f_{CPU} \leftarrow f_{XH} \leftarrow f_X$）

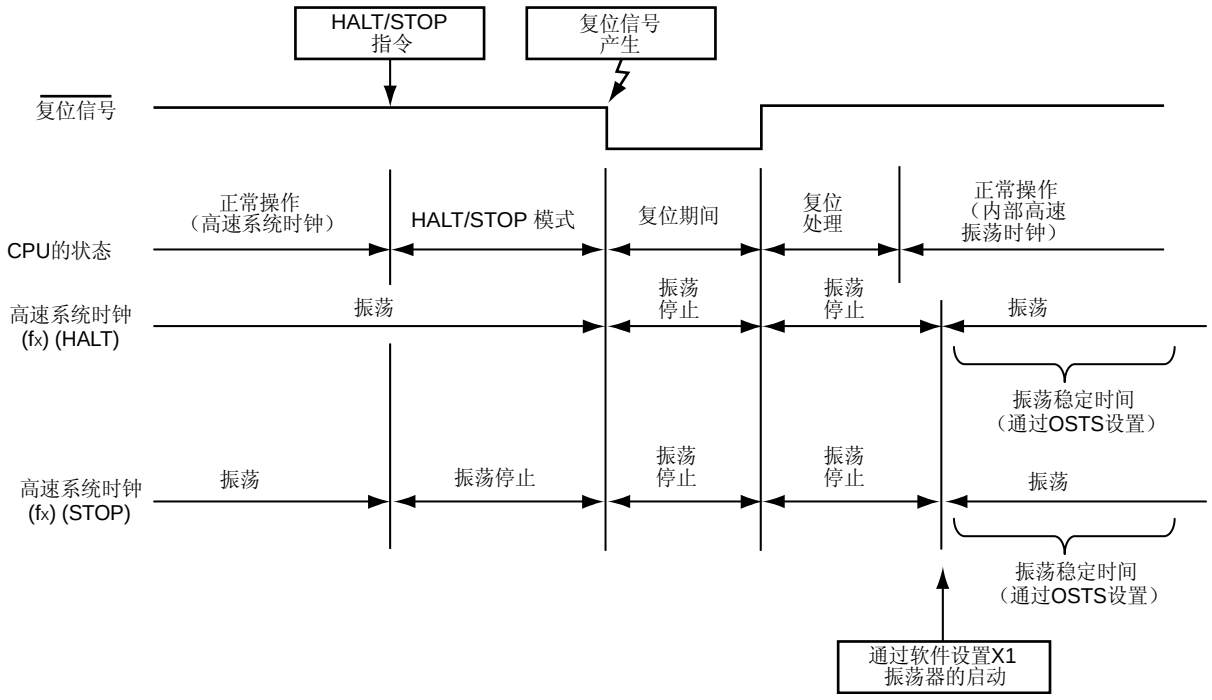


表 2-10. 从 HALT 模式和 STOP 模式返回的返回时间（$f_{CPU} \leftarrow f_{XH} \leftarrow f_X$）

资源	HALT 模式/STOP 模式
复位处理	12 到 51 μs
振荡稳定时间（通过 OSTS）	等待时间设置到 OSTS 寄存器（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> 当 CPU 时钟（f_{CPU}）由高速系统时钟的 X2 外部时钟（f_{EXCLK}）提供

图 2-16. 通过复位信号返回（$f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$）

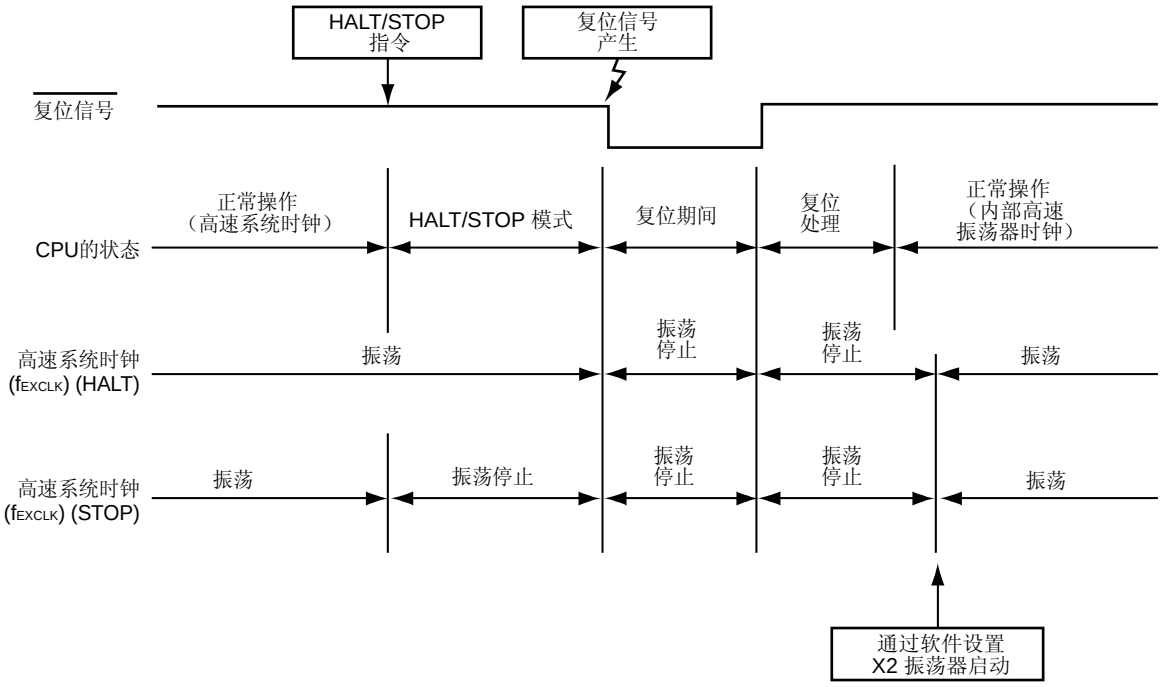


表 2-11. 从 HALT 模式和 STOP 模式返回的返回时间（$f_{CPU} \leftarrow f_{XH} \leftarrow f_{EXCLK}$）

资源	HALT 模式/STOP 模式
复位处理	12 到 51 μs

<4> 当 CPU 时钟（f_{CPU}）由副系统时钟的 2 分频（$f_{SUB}/2$）提供

只有 XT1 输入（f_{XT}：晶体振荡器/陶瓷振荡器）和 XT2 输入（f_{EXCLKS}：外部时钟）**允许** HALT 指令 （禁止 STOP 指令的执行）。

该 XT1 输入振荡稳定时间的等待时间由用户根据所使用的振荡器使用一个定时器等测量。

图 2-17. 通过复位信号返回（$f_{CPU} \leftarrow f_{SUB}/2$）

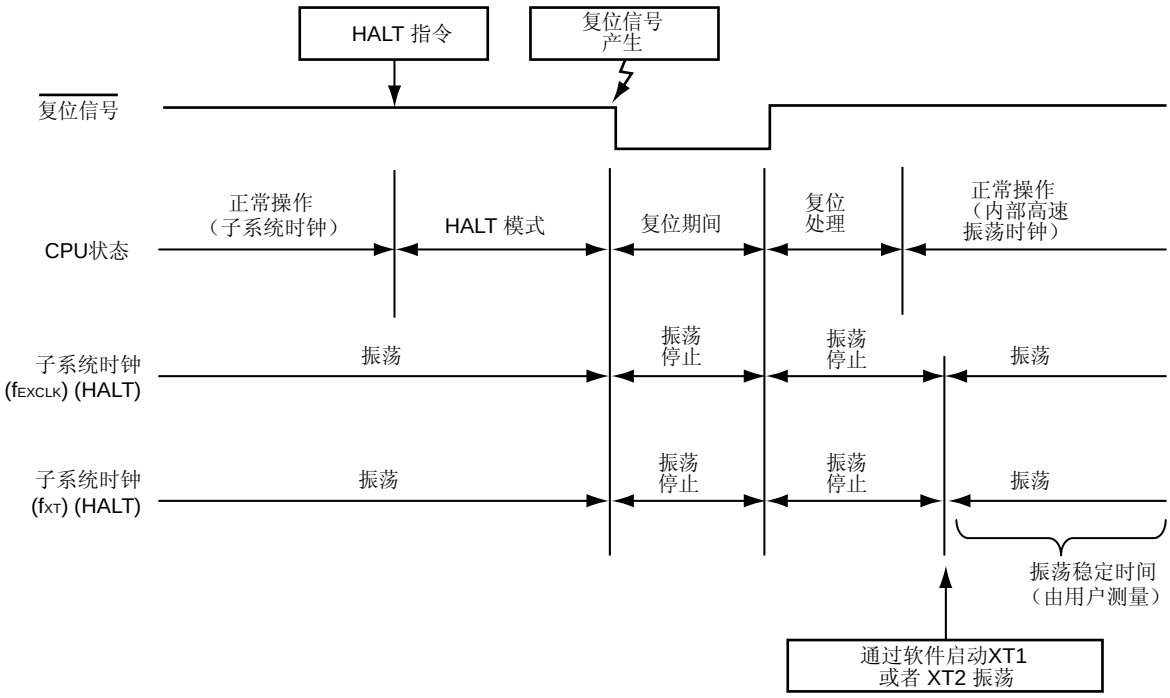


表 2-12. 从 HALT 模式返回的返回时间（$f_{CPU} \leftarrow f_{SUB}/2 \leftarrow f_{XT}$, f_{EXCLKS}）

资源	HALT 模式/STOP 模式
复位处理	12 到 51 μs
振荡器稳定时间（由用户测量）（$f_{CPU} \leftarrow f_{SUB}/2 \leftarrow f_{XT}$ 仅）	系统等待适合于所用振荡器的稳定时间，可以使用例如一个定时器等进行测量。

2.6 时钟发生器和待机功能的注意事项

如果副系统时钟经过 2 分频（$f_{SUB}/2$）提供给 CPU 时钟（f_{CPU}）， 不能使用 flash 存储器的自编程功能。如果内部高速振荡时钟或者高速系统时钟提供给 CPU 时钟（f_{CPU}）自编程功能可以使用。

如果通过 STOP 指令或者程序停止外围硬件时钟（f_{PRS}），确保停止使用外围硬件时钟作为时钟资源的外围硬件设备。

[专栏] 自编程功能

78K0/Kx2-L 微处理器具有自编程功能，它通过一个用户程序允许重写 flash 存储器。当使用自编程功能时，请使用 NEC Electronics 提供的自编程库。有关自编程库的详细情况，请参考第四章中的 4.2 初始化设置和工作区中的专栏关于 EEPROM™ 仿真库和自编程库（1）。

第三章　稳压器

3.1　稳压器概述

78K0/Kx2-L 微处理器包含了一个恒定电压操作的电路，这时为了稳定稳压器的输出电压，使用一个电容（0.47 到 1 μF）把 REGC 引脚连接到 Vss 。然而，当使用 STOP 模式时，以内部高速振荡时钟和外部主系统时钟操作进入这个模式时，推荐使用 0.47 μF 电容。此外，使用具有良好的特性的电容，因为它是用来稳定内部电压的。

稳压器的正常输出电压是 2.4 V（typ.），低功耗模式是 2.0 V（typ.）。

3.2　控制稳压器的寄存器

（1） 稳压器模式控制寄存器（RMC）

这个寄存器设置稳压器的输出电压。

用一个 8 位存储器操作指令设置 RMC。

复位信号的输入设置这个寄存器为 00H。

图 3-1. 稳压器模式控制寄存器（RMC）的格式

地址： FF3DH　复位后：00H　R/W

符号	7	6	5	4	3	2	1	0
RMC								

RMC[7:0]	稳压器的输出电压控制
56H	固定低功耗模式为（2.0 V）
00H	根据条件（参考 **表 3-1**）切换正常功耗模式（2.4 V）与低功耗模式（2.0 V）
其它	禁止设置

注意事项 1. 改变 RMC 寄存器的设置值从 56H 到 00H，使用 5 MHz 或者更高的 CPU 操作频率，RMC 寄存器设置之后经过至少 10μs 或者更长时间再改变 PCC 和 RCM 寄存器的值。

2. 当使用设置固定为低功耗模式时， RMC 寄存器在以下情况下使用。

<当 CPU 时钟选择为 X1 时钟时>

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

<当 CPU 时钟选择为内部高速振荡时钟，外部输入时钟，或副系统时钟时>

$f_{CPU} \leq 5$ MHz

表 3-1. 稳压器输出电压条件

模式	输出电压	条件
低功耗模式	2.0 V	在 STOP 模式
		当使用子系统时钟（f_{XT}）的 CPU 操作时，停止高速系统时钟（f_{XH}）和内部高速振荡时钟（f_{IH}）
		在 HALT 模式下设置为 CPU 以子系统时钟操作时，停止高速系统时钟（f_{XH}）和内部高速振荡时钟（f_{IH}）
正常功耗模式	2.4 V	以上除外

3.3 自编程注意事项

1. 当执行自编程或者 EEPROM 仿真时，固定稳压器输出电压模式。
2. 在电源电压范围内，正常的供电模式是 $V_{DD} \geq 2.5$ V， flash 存储器可以重写。另外，在正常的电源模式下通过使用自编程库可以重写程序区域。
3. 在低功耗模式下重写 flash 存储器时要注意以下几点。
   在低功耗模式下的数据区域可以被重写，但是程序区域不能被重写。
   如果电源等于或者低于稳压器（2.0 V）输出电压时，在低功耗模式下 flash 存储器不能被重写
   在正常的电源模式下不能访问在低功耗模式下重写和擦除的 Flash 存储器，在正常电源模式下为了使用这些数据，需要转换到低功耗模式并将 flash 存储器的内容转移到 RAM 中。
   在 EEPROM 仿真期间执行重写时，可以重写相同的块。然而，当通过使用自编程库来执行重写时，不能重写相同的块，因此在重写数据之前确保首先擦除一个块。
   从正常电源模式切换到低功耗模式后执行自编程之前需要等待 2 ms。

备注 关于自编程功能和自编程库的详细内容，可以参考 **78K0 微处理器自编程库的类型 01 用户手册（U18274E）**和 **78K0 自编程库的类型 01 用户手册 78K0/Kx2-L，78K0/Ix2 规格差异（ZBB-CB-09-0019）**。
关于 EEPROM 仿真的详细内容，可以参考 **78K0 微处理器 EEPROM 仿真库的类型 01 用户手册（U18275E）**和 **78K0 EEPROM 仿真库的类型 01 用户手册 78K0/Kx2-L，78K0/Ix2 规格差异（ZBB-CB-09-0020）**。

第四章　低功耗程序示例

本章介绍使用 78K0/KC2-L 微控制器的各种低功耗功能的示例程序。

本章列出了与待机功能和稳压器有关的主要的程序。有关所有程序，参考 C 源程序代码。

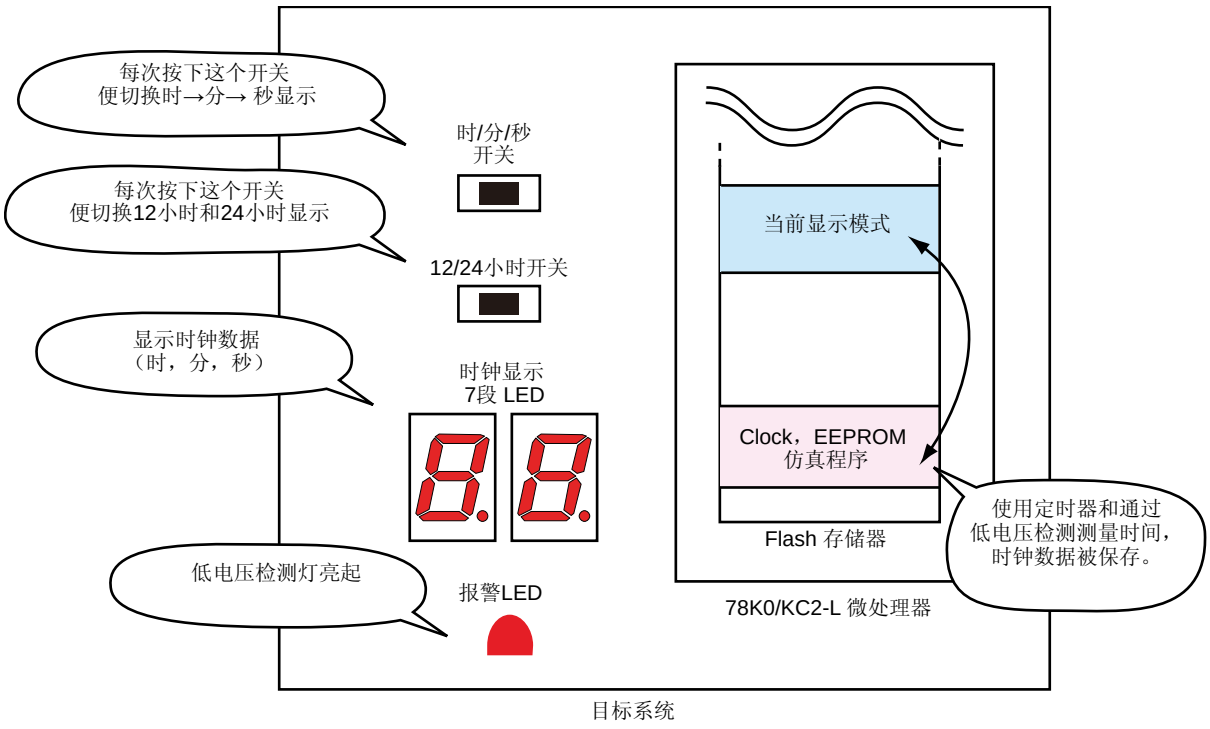
4.1　规格说明和整体流程

通过使用一个实时计数器实现时钟的功能。使用开关可以改变显示模式（时，分， 秒和 12/24 小时的显示）。

通过低电压检测器检测低电压（2.53 V ±0.1 V），报警 LED 灯亮起，通过使用 EEPROM 仿真库当前的显示模式（时，分，秒和 12/24 小时的显示）自编程到 flash 存储器。

通过切断一次电源然后检测到低电压后重复执行，显示模式被重新建立，可以检测写入到 flash 存储器中数据。

图 4-1. 程序操作（图）



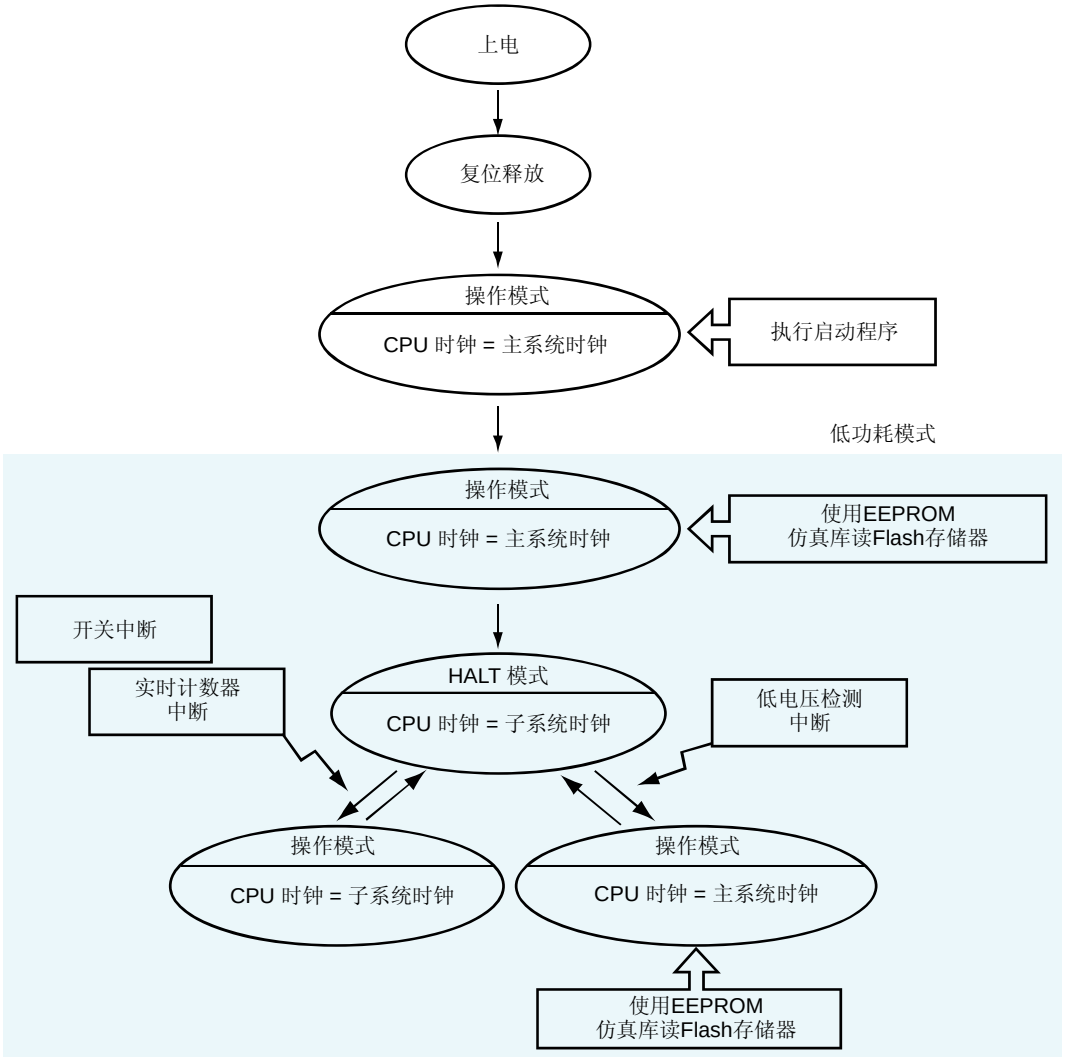
下面显示了示例程序的迁移图的状态。

执行启动程序后，设置低功耗模式和使用 EEPROM 仿真库读取显示模式。CPU 时钟切换到子系统时钟然后进入 HALT 模式。

通过使用实时计数器产生的中断，开关输入产生的中断，或者低电压检测产生的中断，从 HALT 模式返回进入到操作模式（正常操作状态）。

低电压检测中断服务期间，CPU 时钟切换到主系统时钟，使用 EEPROM 仿真库将显示模式写入 flash 存储器。

图 4-2. 程序状态迁移图

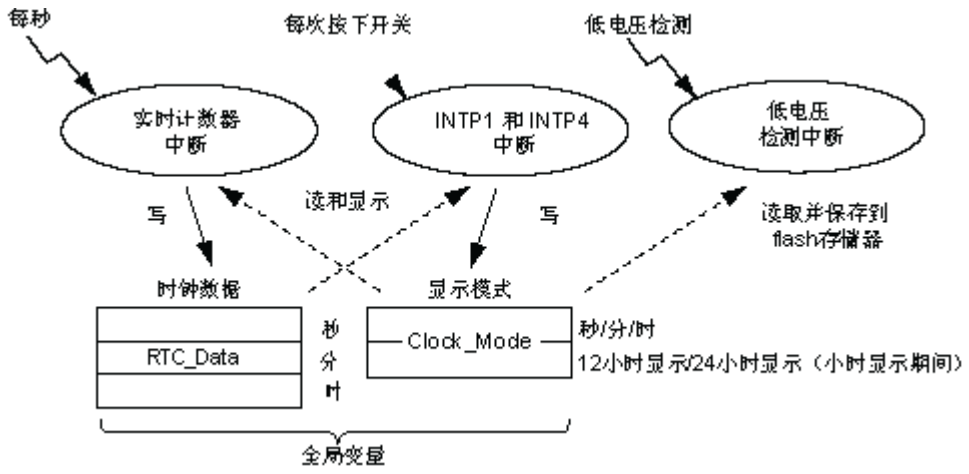


下面是中断服务和示例程序的数据流。

实时计数器中断服务期间，实时计数器的时，分，秒数据保存到 RTC_Data 变量。INTP1 和 INTP4 中断服务期间，显示模式写入到 Clock_Mode 变量。

低电压检测中断服务期间，显示模式被读取并保存到 flash 存储器中。

图 4-3. 中断服务和数据



示例程序中，根据处理切换 CPU 时钟 （f_{CPU}）的时钟源。
程序流和时钟状态显示如下。红色字符表示由于程序设置的结果产生的变化。

图 4-4. 程序处理和时钟状态

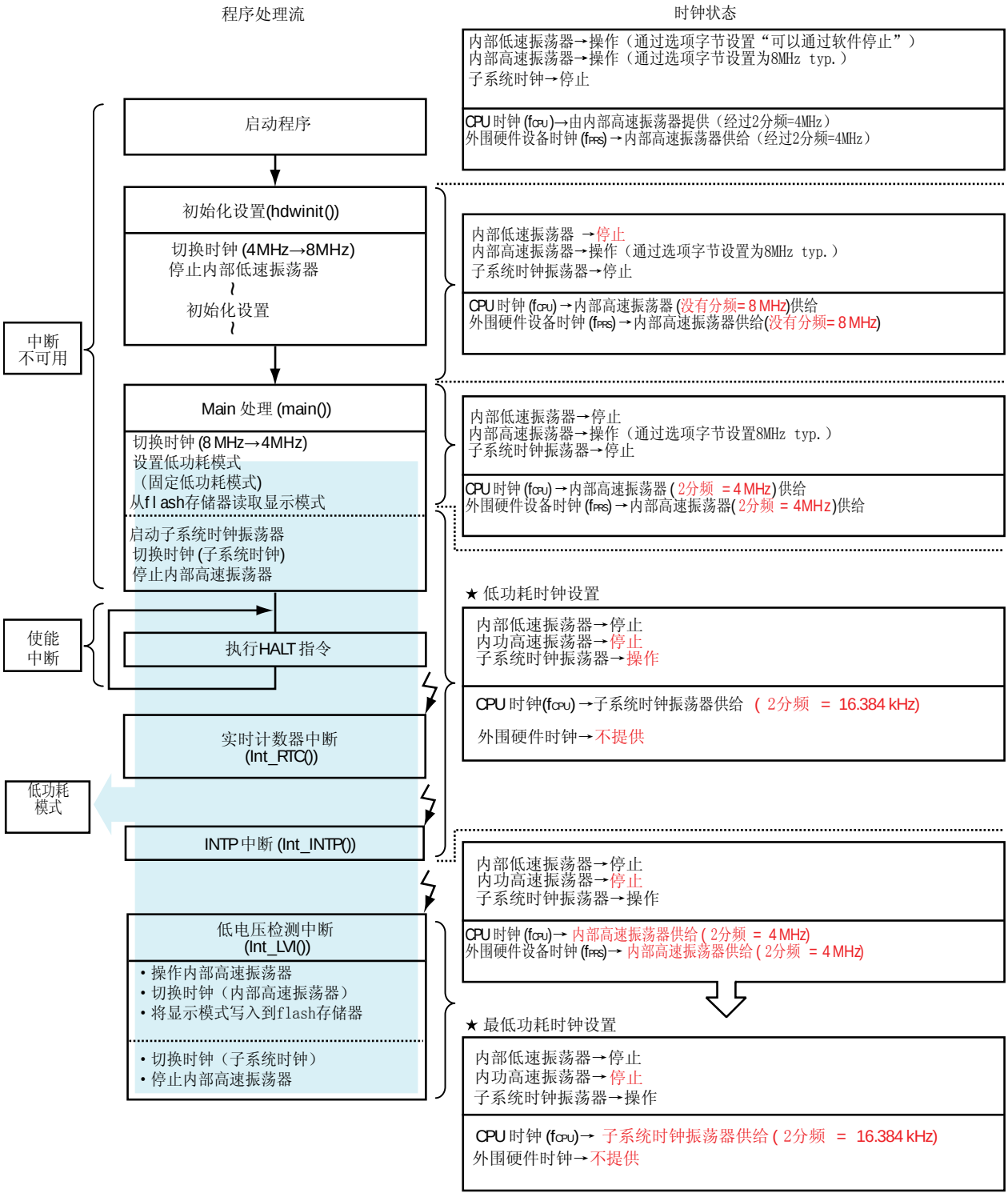


表 4-1. 程序文件的配置 （文件名称: U19612_Kx2L_LowPower_01_AN）

文件名称		文件名称	描述
U19612_Kx2L_LowPower_01_AN		Kx2L_LowPower.lmf	加载模块文件
		Kx2L_LowPower.hex	HEX 格式化对象模块文件
		Kx2L_LowPower.prw	工程文件
		–	其他生成文件和相关工程文件
	/src	initial.c	初始化设置
		main.c	主程序
		Int_RTC.c	实时计数器中断功能
		Int_Intp.c	INTP 边沿检测中断功能
		Int_LVI.c	低电压检测中断功能
		option.asm	选项字节设置文件
		LowPower.h	头文件
		Kx2_eee.h	头文件 （用于 EEPROM 仿真库）

这个示例程序是为 Tessera Technology Inc.的 TK-78K0/KC2L 评估板设计的。（它兼容μPD78F0588GA-GAM-AX）。有关 TK-78K0/KC2L 的操作方法，请参考 TK-78K0/KC2L 文档。

微控制器的输入输出 和特殊功能寄存器（SFR）的使用程序如下所示。

表 4-2. 程序使用的 TK-78K0/KC2L 的输入/输出端口

输入/输出	应用	引脚连接
7 段 LED	上电后显示“-”。 显示时钟数据的时，分，和秒。 按下时/分/秒开关显示每个时间的时 → 分 → 秒。 按下 12/24 小时开关显示每个时间的 12 小时和 24 小时模式。 默认的显示是秒和 12 小时显示，但是低电压中断输入后启动的情况下，直到重新建立才使用这个显示。	<段> 端口 2 P20 到 P27 引脚 <数字切换> 端口 0 P00 和 P01 引脚
时/分/秒开关（SW5）	按下时/分/秒开关显示每个时间的时 → 分 → 秒。	P30/INTP1 引脚
AM/PM 开关（SW6）	按下 12/24 小时开关显示每个时间的 12 小时和 24 小时。	P33/TI51/TO51/INTP4 引脚
报警 LED	低电压检测中断输入的指示灯。（TK-78K0/KC2L 不包括这个 LED。）	端口 0 P02 引脚

表 4-3. 程序使用的 微控制器的特殊功能

特殊功能	应用
实时计数器	实现时钟功能（中断）。
低电压检测器	检测供电电压的低电压（中断）。
中断引脚	INTP1：检测时/分/秒的开关。 INTP4:：检测 12/24 小时的开关。
端口	Port 2： 7 段 LED 的输出。 Ports 0_0，0_1：切换显示 7 段 LED 数字。（该时钟信号输入到每个锁存器。）

4.2　初始化设置和工作区

在示例程序中使用 EEPROM 仿真库写入 flash 存储器，必须包含 EEPROM 仿真库头文件和必须设置库的工作区。其次，头文件的路径必须使用编译器设置，EEPROM 仿真库和自编程库文件必须使用连接器设置。

清单 4-1. 启动程序初始设置函数（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    EEPROM 仿真库头文件
*-----------------------------------------------------------------------*/
#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
*-----------------------------------------------------------------------*/

       省略

                                  用于自编程库
/*-------------------------------------------
*    Global variables
*-------------------------------  用于EEPROM 仿真库  ---------*/

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];           显示模式写入到 flash存储器
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};
   实时时钟的时钟数据
/************************************************************************************
                                                  7段 LED的点亮段码
           Initialization after RESET

************************************************************************************/
void hdwinit(void)  从启动程序开始执行
{
          /* Disable interrupts */
          DI();
          /* Peripheral hardware initial settings */
          Init_sfr();
}
```

[专栏] 关于 EEPROM 仿真库和自编程库（1）

EEPROM 仿真库和自编程库由 NEC Electronics 提供用于从用户程序写 flash 存储器。

自编程库用于写入和擦除 flash 存储器。

EEPROM 仿真库用于实现以任意大小区域操作 flash 存储器，以 4 字节为最小单元写入 flash 存储器，到擦除大小为 1 块（= 1024 字节）。实际上 flash 存储器的写入和擦除是从 EEPROM 仿真库中调用自编程库实现的。

所有没有连接的端口设置为输出端口。子系统时钟启动后，TM00 定时器启动用于测量振荡稳定时间。

清单 4-2. 初始化设置函数（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 = approx. 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 */
```

启动子系统时钟操作

启动振荡稳定时间 （2 s）测量

[专栏] 关于 EEPROM 仿真库和自编程库（2）

如果用户程序使用 EEPROM 仿真库，自编程库也将被使用，所以需要进行一些设置以使 EEPROM 仿真库和自编程库的库文件可以使用连接器连接。

清单 4-2. 初始化设置功能（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  s/(125 * 2 ns) = 40 */
                          /* Mask LVI interrupt (disable interrupt) */            μ
LVIMK = 1;
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 */
}
```

通过宏定义
延迟4个NOP指令

提供控制时钟

停止提供
控制时钟

[专栏] 有关实时计数器的寄存器设置

使用子系统时钟（f_{SUB}）操作实时计数器，只要提供 CPU 时钟（f_{CPU}），即使子系统时钟（f_{SUB}）停止，也可以设置相关的寄存器。写入相关寄存器需要提供控制时钟。（读这些寄存器无需提供控制时钟。）

通过设置外围使能寄存器 0（PER0）的 RTCEN 位为 1，提供控制时钟。当不执行相关寄存器的写操作时，停止控制时钟的供给（RTCEN = 0）以降低功耗。

4.3　主处理

在启动程序中执行 hdwinit()函数后，调用 main()函数，执行存储器初始化设置等操作。

启动实时计数器之前，从 flash 存储器中读取显示模式。如果显示模式通过一个低电压检测中断写入，将再现之前的显示状态。

因为在 hdwinit()函数中子系统时钟的操作被启动，启动定时器 TM00 用于 2 秒时间的测量，在子系统时钟提供给 CPU 时钟之前测量 2 秒定时是否已到。如果已经到达 2 秒。允许定时器 TM00 中断，系统进入在 HALT 模式下的待机模式。如果中断使能标志无效（IE = 0），则操作不会分支到一个中断向量。因此，当产生一个中断请求标志时，在已经产生中断请求之后必须手动清除中断请求标志。

清单 4-3. 主函数（main.c）

```
/************************************
从启动程序执行
************************************/
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();

        }
}
```

CPU 时钟切换到主时钟的2分频，并设置低功耗模式。
（由于主时钟是内部高速振荡器 8 MHz (typ.)，因此分频后的结果是4 MHz (typ.)。）
（仅当CPU时钟为5 MHz或者更低时，固定设置低功耗模式。）

如果期望在包括EEPROM仿真库的执行期间设置禁止中断， 则必须遵循下面的顺序： 首先设置中断屏蔽标志，然后执行EEPROM 仿真库，接下来执行 DI()，之后再清除中断屏蔽标志。这里，设置了中断屏蔽标志，所以仅执行 DI() 。
（更详细的内容，参考4.6 低电压中断服务。）

执行 HALT 指令

如果没有执行中断服务，
确保清除中断请求标志

执行 HALT 指令

（1）设置时钟发生器操作/停止

当启动各种时钟的发生器时，从启动时钟发生器操作开始必须等待时振荡稳定时间。

内部高速振荡器是否振荡稳定可以通过参考状态（RSTS = 1：稳定操作）检测。

子系统时钟使用 XT1 振荡器的情况下，必须使用一个定时器等功能等待振荡稳定时间。在程序中使用定时器 TM00 测量 2 秒。（关于振荡稳定时间，请联系振荡器的生产厂家。)

在此之前停止内部高速振荡器，检查另一个振荡时钟（高速系统时钟或者子系统时钟）供给 CPU 时钟（f_{CPU}）。

清单 4-4. 操作或停止时钟发生器（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;
}
```

在此之前停止内部高速振荡器，检查另一个振荡时钟提供给CPU 时钟。

[专栏] 启动程序

在标准的 CC78K0 编译器启动程序中，首先调用 hdwinit()函数，完成变量（有初始值和没有初始值）的初始化设置，然后调用 main() 函数。在 hdwinit()函数中的变量的值被设置后，已经执行了初始化，因此需要谨慎。

（2）切换 CPU 时钟（f_{CPU}）

当切换了 CPU 时钟的时钟源或它的分频比时，需要等待一些时间直到切换全部有效。

当时钟源从主系统时钟切换到子系统时，或者反之亦然，使用 CLS 位可以检测，但是当时钟源为主系统时钟，切换预分频器的分频比时，系统需要等待一些时间直到时钟生效。

清单 4-5. 切换 CPU 时钟（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();
                        }
                }
        }
}
```

通过宏定义
延迟4个NOP的时间

当切换主系统时钟的分频比时等待所需要的时间。

通过宏定义
延迟8个NOP的时间

在主系统时钟 <-> 子系统时钟之间切换的情况下，检查 CLS。

（3）设置低功耗模式

稳压器模式控制寄存器（RMC）设置为 56H。

清单 4-6. 设置低功耗模式（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）使用 EEPROM 仿真库读 flash 存储器

读取显示模式是否被写入到 flash 存储器。如果显示模式已经写入（如果低电压检测中断已经产生），这些内容被设置到变量中。如果显示模式没有被写入，初始值 （显示秒，12 小时模式）被设置到变量中。

清单 4-7. 读 Flash 存储器（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 设置

[专栏] FLMDPUP 位

在 78K0/Kx2-L 系列微控制器中，FLMDPUP 位必须设置为 1 以执行自编程。为使用 EEPROM 仿真库，需在执行该库之前设置 FLMDPUP 位为 1，接下来执行库函数时，设置 FLMDPUP 位为 0。

（5）设置实时计数器的 AMPM 位

在 main() 函数中，从 flash 存储器已经读取显示模式后设置实时计数器的 AMPM 位。

在示例程序中，12 小时显示的值在初始化设置中被设置到时计数寄存器 HOUR 中，如果显示模式从 12 小时显示变为 24 小时显示，则需要调整这个设置值。其次，当 AMPM 位从 12 小时变为 24 小时时，时计数寄存器的值变为 0，所以一定要事先保存此值。

关于如何修改设置值的详细情况请参考 **4.5 INTP1，INTP4 中断服务。**

清单 4-8. 设置 AMPM 位（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;                                                          ← 供应控制时钟

         /* 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;                                                          ← 停止控制时钟供应

}
```

4.4　实时计数中断服务

在示例程序中，通过使用实时计数器的报警中断更新每一秒的显示。

当读取时，分，和秒计数寄存器时，实时计数控制寄存器 1（RTCC1）的 RWAIT 位必须设置为 1（计数停止设置，读/写模式）停止计数，必须使用 RWST 标志检查是否计数停止。然而，如果在实时计数被停止 （RTCE = 0）时设置各种寄存器，则 RWAIT 位不需要设置。当实时计数停止，即使设置了 RWAIT 位，RWST 标志也保持不变。

List 4-9. 实时计数中断（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;

}
```

显示模式

时钟数据

设置计数器值的读/写模式

读寄存器

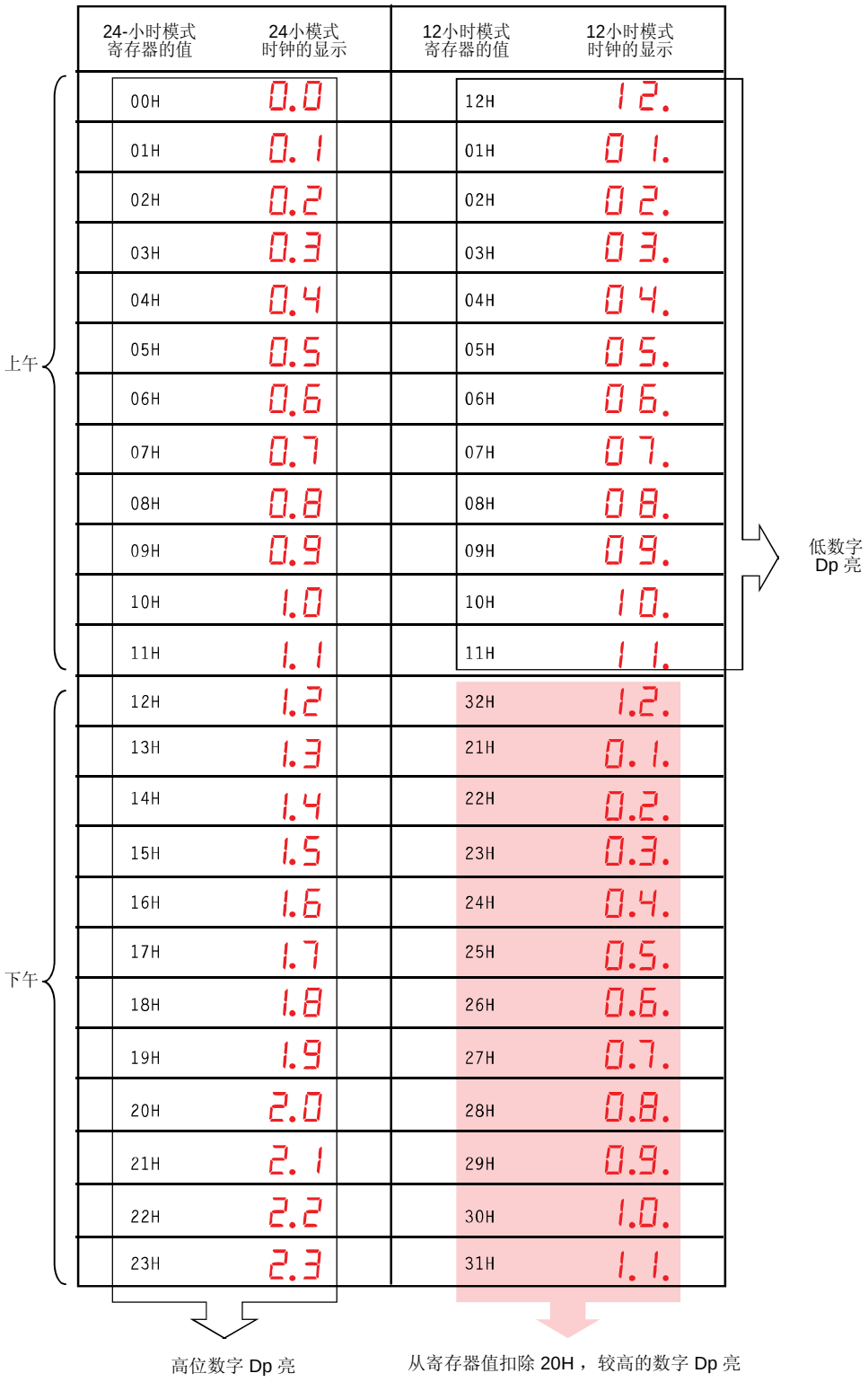
返回计数器操作模式

时计数寄存器的值和 7 段 LED 的显示形式的对应关系如下所示。

24 小时模式的情况下，较高数字的 Dp 亮，和 12 小时模式的情况下，较低数字的 Dp 亮。其次，在 12 小时模式下下午的时间，较高的 Dp 也亮。

12 小时模式的情况下，21H 和较高的值与实际时钟显示不同，所以必须在显示之前处理。

图 4-5. 时计数寄存器和时钟显示形式的对应关系



清单 4-9. 实时计数中断（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)];
            }
    }
}
```

清单 4-9. 实时计数中断（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;
    }
}
```

如果寄存器值是 21H 或者更高，除去 20H，更高数字的 Dp 亮

如果是12小时显示，显示较低数字的 Dp

如果24小时显示，显示较低数字的 Dp

[专栏]停止提供外围硬件设备时钟时重写端口寄存器

实时计数，INTP1 和 INTP4 中断期间，停止内部高速振荡器，所以没有时钟供给外围硬件，但是。如果提供 CPU 时钟（f_{CPU}），则端口寄存器可以被重写。

4.5　INTP1, INTP4 中断服务

使用 INTP1 中断通过开关完成显示时/分/秒的切换，使用 INTP4 中断通过开关完成 12/24 小时显示切换。

（1）INTP1 中断服务

每次进入中断则切换显示模式。根据显示模式 7 段 LED 的显示也被切换。

清单 4-10. INTP1 中断服务（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];
extern UCHAR     RTC_Data[3];

/*****************************************************************************************************
;       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;
        }
}
```

显示模式

时钟数据

（2）INTP4 中断

当 12 小时模式切换为 24 小时显示时（AMPM 位的值从 0 改变为 1），时计数寄存器 HOUR 的值变为 0，所以必须保存寄存器内容，而且，12 小时显示值和 24 小时值的格式不同，他们必须经过转换后返回到时计数寄存器 HOUR。

清单 4-11. INTP4 中断服务（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;
                }
```

供应控制时钟

设置计数器值读/写模式

保存寄存器的值

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	

清单 4-11. INTP4 中断服务（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;          <- 返回计数器操作模式

        /* Stop supplying control clock for write access to real-time counter */
        RTCEN    = 0;       <- 停止控制时钟供应

        /* 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　低电压检测中断服务

检测到 2.53 V ±0.1 V 低电压时，将产生一个低电压检测中断。

中断服务期间，为了使用 EEPROM 仿真库将显示模式写入到 flash 存储器中，必须使内部高速振荡器操作，并且 CPU 时钟（f_{CPU}）必须由内部高速振荡时钟（4 MHz （typ.））提供。（要执行 flash 存储器的自编程，CPU 时钟（f_{CPU}）不能使用子系统时钟作为时钟源。）。接下来的写操作，CPU 时钟（f_{CPU}）返回到子系统时钟（f_{SUB}）作为时钟源，停止内部高速振荡器。

EEPROM 仿真库的执行期间禁止中断，必须设置其他的中断屏蔽标志。

清单 4-12. LVI 中断服务（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 flags
                    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;
}
```

当使用 EEPROM 仿真库执行写时如果要写入的块已满，全部的数据作为返回值返回，在这种情况下改变要写入的块。

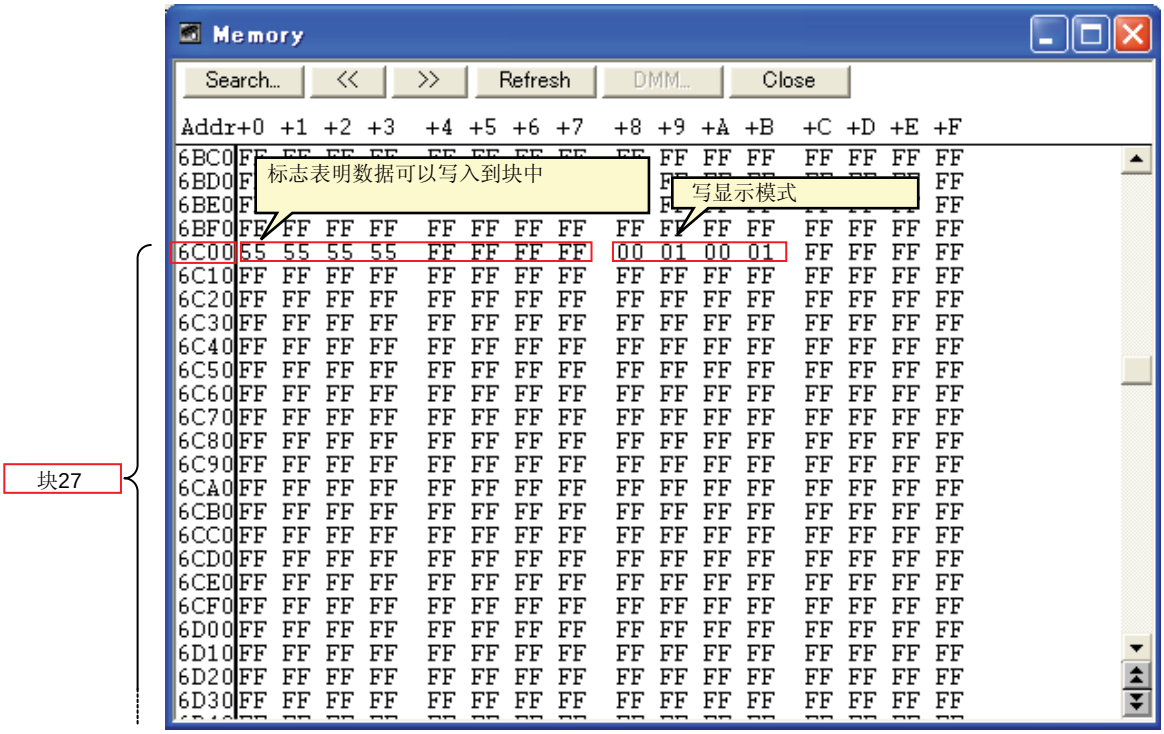
清单 4-12. LVI 中断服务（Int_LVI.c）（2）

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

          FLMDPUP = 1;                                          FLMDPUP 设置
          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)                                如果块满，改变这个块
          {

                    /* 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 );
}
```

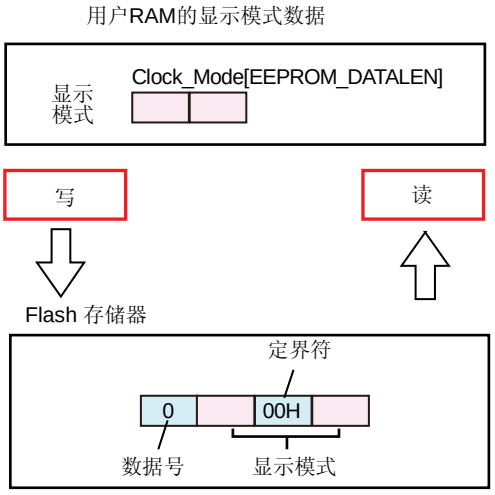
使用 EEPROM 仿真库写 flash 存储器的操作显示如下（用 MINICUBE2 检查）。因为写块号通过 EEPROM 仿真库的头文件 Kx2_eee.h 指定为 27（1BH），因此，从地址 6C00H 开始执行写操作。（在使用 EEPROM 仿真库的情况下，从指定的块号开始的 4 个块作为数据写入区。）

在使用 EEPROM 仿真库的情况下，添加数据号和定界符后开始写入数据，并且写入的数据需要调整为 4 个字节的倍数，这是 flash 存储器写入的最小单元。如果只写入 2 个字节，则需要添加数据并使定界符为 4 个字节，所以长度不需要调整为 4 的倍数。在前面添加数据号，定界符（00H）添加在倒数第二个字节的位置上。

图 4-6. 显示模式写入 Flash 存储器



<使用EEPROM 仿真库写入数据的格式>



第五章 相关文档

文档名称		PDF/文档编号
RA78K0 Ver. 3.80 用户手册[注1]	操作	PDF
	语言	PDF
	结构化汇编语言	PDF
RA78K0 Ver. 4.01 操作注意事项 （通知文档）[注1]		ZUD-CD-07-0181-E
CC78K0 Ver. 3.70 用户手册[注2]	操作	PDF
	语言	PDF
CC78K0 Ver. 4.00 操作注意事项 （通知文档）[注2]		ZUD-CD-07-0103-E
SM+ 用户手册	操作	PDF
	用户开放接口	PDF
ID78K0-QB Ver. 2.94 用户手册	操作	PDF
ID78K0-QB Ver. 3.00 用户手册	操作	PDF
PM plus Ver. 5.20[注3] 用户手册		PDF
PM+ Ver. 6.30[注4] 用户手册		PDF

注
1. 当安装 RA78K0 Ver. 4.01 时这个文档随着开发工具一起安装在 PC 中。这些描述不包括在“RA78K0 Ver. 4.01 操作注意事项中 （通知文档）”，参考用户手册 RA78K0 Ver. 3.80.
2. 当安装 CC78K0 Ver. 4.00 时这个文档随着开发工具一起安装在 PC 中。这些描述不包括在“CC78K0 Ver. 4.00 操作注意事项中 （通知文档）”， 参考用户手册 CC78K0 Ver. 3.70。
3. PM + Ver. 5.20 是包括 RA78K0 Ver. 3.80 的集成开发环境。
4. PM+ Ver. 6.30 是包括 RA78K0 Ver. 4.01 的集成开发环境。可以管理不同版本的软件工具（汇编器，C 编译器，调试器和仿真器）。

注意事项 以上所列的有关文档如有变动，不另行通知。设计开发时确保使用每个文档的最新版本。

详细信息请联系：

中国区

MCU 技术支持热线：
电话：+86-400-700-0606 (普通话)
服务时间：9:00-12:00，13:00-17:00 （不含法定节假日）

网址：
http://www.cn.renesas.com/ （中文）
http://www.renesas.com/ （英文）

[北京]
瑞萨电子（中国）有限公司
中国北京市海淀区知春路 27 号量子芯座
7，8，9，15 层
电话：（+86）10-8235-1155
传真：（+86）10-8235-7679

[上海]
瑞萨电子（中国）有限公司上海分公司
中国上海市浦东新区陆家嘴环路 1233 号
汇亚大厦 205 室
电话：（+86）21-5877-1818
传真：（+86）21-6887-6100

瑞萨电子（上海）有限公司
中国上海市浦东新区陆家嘴环路 1233 号
汇亚大厦 205 室
电话：（+86）21-5877-1818
传真：（+86）21-6887-6100

[长春]
瑞萨电子（中国）有限公司长春分公司
吉林省长春市朝阳区
西安大路 727 号中银大厦 A 座 1609 室
电话：(+86)431-8859-7533 / 8859-8533
传真：(+86)431-8680-2944

[青岛]
瑞萨电子（中国）有限公司青岛分公司
中国山东青岛市宁夏路 288 号 G3 楼 607 室
电话： (+86)532-8872-7900/8872-7901
传真：(+86)532-8872-7902

[深圳]
瑞萨电子（中国）有限公司深圳分公司
深圳市福田区益田路卓越时代广场大厦 39 楼
3901，3902，3909 室
电话：（+86）755-8282-9800
传真：（+86）755-8282-9899

[香港]
香港瑞萨电子有限公司
香港九龙旺角太子道西 193 号新世纪广场
第 2 座 16 楼 1601-1613 室
电话： （+852）2886-9318
传真： （+852）2886-9022
2886-9044

[成都]
瑞萨电子（中国）有限公司成都分公司
四川省成都市二环路南三段 15 号
天华大厦 608 室
电话：(+86)28-8512-5224
传真：(+86)28-8512-5334

[大连]
瑞萨电子（中国）有限公司大连分公司
大连市中山路 88 号天安国际大厦 2701 室
电话：(+86)411-8230-8815 / 8230-8825
传真：(+86)411-8230-8835