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

Power-Down Mode Demonstration

For NEC Electronics Microcontrollers

Document no. U18288EU1V0AN00

The information in this document is current as of July 2006. 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 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 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 NEC Electronics 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 customer's equipment shall be done under the full responsibility of customer. NEC Electronics 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 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 NEC Electronics sales representative in advance to determine NEC Electronics 's willingness to support a given application.

Notes:

1. "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
2. "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02.10

Revision History

Date	Revision	Section	Description
July 2006	—	—	First release

Contents

1. Introduction 9

1.1 An Overview of Power-Down Features 9

2. Clock Control and Standby 10

2.1 Features of Clock Control and Standby Functions 10
2.1.1 Clock Control Features 10
2.1.2 Standby Features 13
2.2 Program Description and Specification 15
2.3 Software Flow Charts 18
2.3.1 Program Startup and Initialization 19
2.3.2 Clock_Init( ) – CPU Clock Initialization 20
2.3.3 INT_Init( ) – Key-Return Interrupt Initialization 21
2.3.4 WT_Init( ) – Watch-Timer Initialization for Square-Wave Generation 22
2.3.5 Main( ) – The Main Program – Power-Down Functions 23
2.3.6 SetClk( ) – Select CPU Clock Source 25
2.3.7 SetClkHSR( ) – Set CPU Clock to Internal High-Speed Oscillator 27
2.3.8 SetClkEx( ) – Set CPU Clock to X1/X2 Crystal Oscillator 29
2.3.9 SetClkEx( ) – Alternate – Set CPU Clock to EXCLK Input 32
2.3.10 SetClkSub( ) – Set CPU Clock to Subclock 34
2.3.11 SetPCC( ) – Set PCC Register for Main Clock Division 35
2.3.12 DispOff( ) – Turn LCD and IIC0 Peripheral Off 38
2.3.13 TurnDIspOff() – Turn Display Off 39
2.3.14 TurnDIspOn() – Turn Display On 40
2.3.15 Standby( ) – Select Standby Mode 41
2.3.16 StandbyHalt1( ) – HALT With Periodic Wake-up Interrupt 43
2.3.17 StandbyHalt2() – HALT with No Periodic Interrupt 44
2.3.18 StandbyStop1( ) – Stop with Periodic Wake-up Interrupt 46
2.3.19 StandbyStop2( ) – Stop with No Periodic Interrupt, Subclock Running 48
2.3.20 StandbyStop3( ) – Stop with No Periodic Interrupt, Subclock Stopped 50
2.3.21 MD_INTKR( ) – Key-Return Interrupt-Service Routine 51
2.3.22 MD_INTWT( ) – Watch-Timer Interrupt-Service Routine 53
2.4 Applilet's Reference Driver 54
2.4.1 Configuring Applilet for Clock Initialization 54
2.4.2 Configuring Applilet for Key-Return Interrupt 56
2.4.3 Configuring Applilet for Watch Timer 57
2.4.4 Configuring Applilet for IIC0 Communication 58
2.4.5 Generating Code with Applilet 60
2.4.6 Applilet-Generated Files and Functions for Clock Initialization 61
2.4.7 Applilet-Generated Files and Functions for Key-Return Interrupt 61
2.4.8 Applilet-Generated Files and Functions for Watch Timer 62
2.4.9 Applilet-Generated Files and Functions for IIC0 Communication 63
2.4.10 Other Applilet-Generated Files 66
2.4.11 Demonstration-Program Files Not Generated by Applilet 67
2.5 Demonstration Platform 67
2.5.1 Resources 67
2.5.2 Demonstration of Program 68
2.6 Hardware Block Diagram 70
2.6.1 Power Measurement Results 71
2.7 Software Modules 74

3. Appendix A - Development Tools 75
3.1 Software Tools 75
3.2 Hardware Tools 75
4. Appendix B – Software Listings 76
4.1 Main.c 76
4.2 Pwr_dn.h 88
4.3 Macrodriver.h 89
4.4 System.h 90
4.5 Systeminit.c 91
4.6 System.c 93
4.7 Int.h 94
4.8 Int.c 95
4.9 Int_user.h 96
4.10 Serial.h 98
4.11 Serial.c 99
4.12 Serial_user.c 106
4.13 Watchtimer.h 110
4.14 Watchtimer.c 111
4.15 Watchtimer_user.c 112
4.16 Option.inc 114
4.17 Option.asm 115
4.18 defines.h 116
4.19 Lcd.h 116
4.20 Lcd.c 117
4.21 LcdDrvApp.h 121
4.22 LcdDrvApp.c 125

1. Introduction

This application note illustrates the use of the peripherals in NEC Electronics microcontrollers. It will help you better understand the peripherals and provide you with basic routines you can use in more complex applications.

The information provided includes:

- Description of peripheral features
- Example program descriptions and specifications
- Software flow charts
- Applilet reference drivers
- Descriptions of the demonstration platforms
- Hardware block diagram
- Software modules

Each of the techniques described in this application note use the Applilet—an NEC Electronics software tool that generates driver code for the peripherals. This tool provides a quick and convenient way to generate code.

For details on using the Applilet and NEC Electronics microcontrollers, please consult the appropriate user manuals and related documents.

1.1 An Overview of Power-Down Features

The power dissipation of CMOS devices is:

P = C x (V**2) x (frequency of operation)

Power dissipation is a direct function of the operating frequency.

NEC Microcontrollers implement many advanced features to select and control CPU and peripheral clocks. Using these features, you can reduce the CPU clock frequency when you do not need full processing power and turn off on-chip peripherals when they are not needed. Standby features, implemented for most NEC Electronics microcontrollers, reduce power consumption to a minimum. These features make NEC Electronics microcontrollers an ideal choice for battery-operated portable systems. This document demonstrates various methods of controlling the clocks and using the standby features.

2. Clock Control and Standby

Whether using an on-board oscillator or external clock, the clock generator is a major power consumer. NEC Electronics microcontrollers have two major functions that reduce this power consumption:

- Comprehensive methods of controlling clocks
- Standby features

2.1 Features of Clock Control and Standby Functions

This section provides a brief description of the features for clock control and standby.

2.1.1 Clock Control Features

The clocking options for NEC Electronics microcontrollers include:

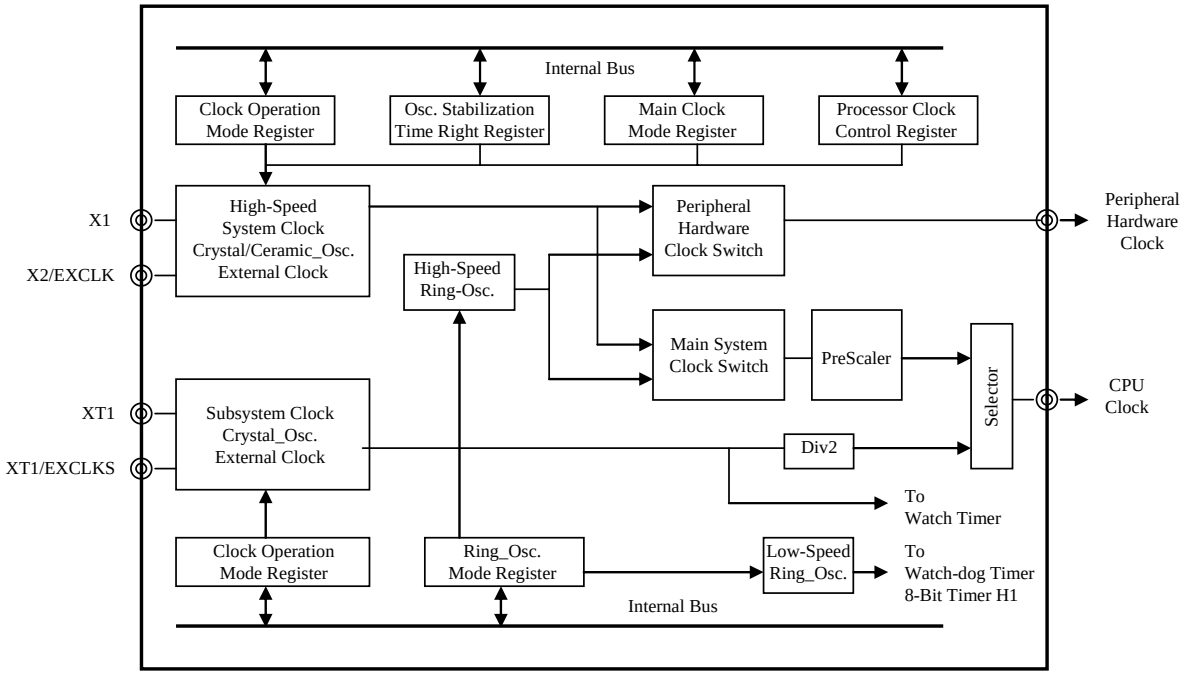
- Main clock operation from external crystal oscillator for high speed
- Operation from a driven external clock
- Operation from an internal high-speed oscillator (available in some microcontrollers)
- Ability to operate the CPU at the main clock frequency or at a fraction of it (for power saving)
- Ability to operate CPU and peripherals from different clocks
- Support for a low-speed external subclock (32.768 kHz) for timekeeping
- Ability to operate the CPU from the subclock for reduced power consumption
- Ability to operate some peripherals on the subclock
- Low-speed internal oscillator for some peripherals (available in some microcontrollers)
- Clock pins not used for clocks available for use as I/O ports

The clock generator generates clocks for the CPU and peripheral hardware. The main clock (X1) can be selected by the main-clock mode register (MCM) and clock-operation mode-select register (OSCCTL). Various sources of the system clock for NEC Electronics microcontrollers are shown in the table below. The subsystem clock is a 32.768 kHz crystal oscillator.

Table 1. System Clock Sources

Clock Category	Clock Select	Descriptions
Main Clock (X1)	Main Oscillator	high-speed main oscillator
		Main osc. can be Stopped by Stop instruction
		Main osc. can also be stopped by setting osc control register
	External Clock	External clock can be disabled by executing stop instruction
		External clock can also be disabled by RCM (internal-osc mode)
	Internal high-speed Osc.	Internal oscillator, typical frequency is 8 MHz
		After reset, CPU always operates with high-speed internal-osc.
		Oscillator can be stopped by stop or internal-osc. mode register
Subsystem Clock (XT1)	Subsystem Clock Osc.	The subsystem clock oscillator frequency is 32.768 kHz
	External subsystem clock	External subsystem clock can be disabled by proc. clock control or
		oscillator-control register
Low-speed Internal oscillator	Internal oscillator oscillates at typical frequency of 240 kHz. After reset, the internal low-speed oscillator always starts operating	
	Oscillation can be stopped by internal-oscillator mode register (depend on mask option or option-byte setting	
	The internal low-speed oscillator cannot be used for CPU clock	
	Internal Low-speed oscillator operates watchdog timer and Timer H1	

Figure 1. Typical Clock System in NEC Electronics Microcontrollers



As main system clock, you can select either the X1 externally supplied high-speed clock (EXCLK) or the internal high-speed oscillator. The peripheral hardware clock derives from the main system clock.

Newer members of the NEC Electronics microcontroller family feature the internal high-speed oscillator. After the release of reset, this oscillator automatically starts—typically at 8 MHz. Once the oscillator stabilizes, you can switch the CPU to this clock source.

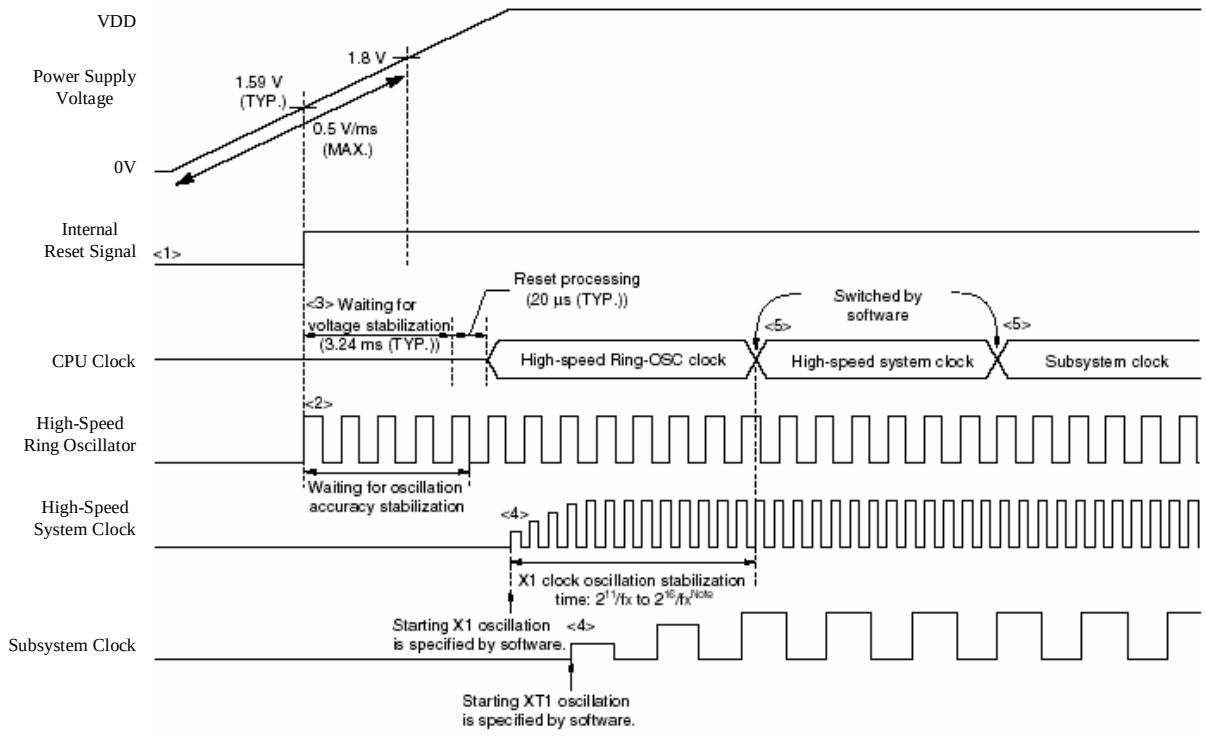
Newer members of NEC Electronics microcontroller family also incorporate an internal low-speed oscillator. This oscillator serves only for the watchdog timer and 8-bit timer H1 and does not work as a CPU clock. After release of reset, the low-speed internal oscillator starts—typically running at 240 kHz.

The microcontroller's prescaler generates various clocks by dividing the main-system clock you select as the CPU clock source.

Table 2. Registers Controlling Clock Generator

Register	Symbol	Description of Functions
Clock-operation mode-select register	OSCCTL	Selects operation modes of high-speed and subsystem clocks
Processor-clock control register	PCC	Selects CPU clock and division ratio
		Sets operating mode for subsystem clock
Internal-oscillator mode register	RCM	Sets operating mode of internal oscillator
Main-oscillator control register	MOC	Selects the operating mode of high-speed system clock
		Stop X1 oscillator or disable external clock (EXCLK)
Main-clock mode register	MCM	Selects main-system clock to CPU and peripheral hardware
Osc. stabilization time-counter status	OSTC	Indicates count status of X1 stabilization time counter
Osc. stabilization time-select register	OSTS	Selects X1 clock oscillation stabilization time

Figure 2. Clock Generation Timing Diagram



Here is a brief description of the timing generation:

- When you turn on the microcontroller's power, the power-on-clear (POC) function generates an internal reset.
- When the power-supply voltage exceeds 1.59V (typical), the reset is released.
- The internal high-speed oscillator clock starts automatically.
- The CPU starts operating on the internal high-speed oscillator.
- When software sets external clock pins for oscillation mode, X1 and XT1 oscillation begins.
- After waiting for clock stabilization, software can switch the CPU clock to X1 or XT1.

To set the system clocks for operation, you typically use this sequence:

- Configure the I/O pins for external high-speed clock (X1 and X2) or I/O mode with the OSCCTL register.
- If using the external clock, enable it with the MOC register.
- Wait for external-oscillator stabilization, using the OSTC register to control the delay.
- Set the OSTS register for the necessary restart stabilization time from STOP mode.
- Switch from the internal high-speed oscillator to the external clock, if desired, using the MCM register.
- Stop the internal oscillator, if desired, with the RCM register.
- Configure the I/O pins for subclock (XT1 and XT2) or I/O mode with the OSCCTL register.
- Select the desired CPU clock divider or subclock with the PCC register.

2.1.2 Standby Features

Standby operation reduces the system operating current, using the halt and stop modes. Features of these standby modes are:

- Either mode retains RAM data and I/O states.
- Halt mode stops CPU execution and provides quick restart for moderate power savings.
- Peripherals can operate in halt mode.
- Main CPU clock stops in stop mode, for large power savings.
- In stop mode, peripherals operate on subclock or internal low-speed oscillator.
- You exit from standby with an unmasked interrupt or reset.
- The microcontroller automatically waits for oscillator stabilization when exiting stop mode.

You set the halt mode by executing a halt instruction. In halt mode, the main CPU clock does not stop oscillating, but the clock is not supplied to the CPU, so the CPU consumes less current. The selected CPU clock resumes operation when you release the halt mode. Although halting does not reduce operating

current as much as stop mode, halt mode is useful if you want to restart operations immediately after an interrupt.

If you operate both the CPU and the peripherals from the same clock, the peripherals' clock is available during halt mode. You can operate the CPU from the internal high-speed oscillator and the peripherals from the external X1 clock, allowing the peripherals to operate during a halt. Alternatively, you can stop the peripherals and their external clock before entering halt mode.

You enter stop mode by executing a stop instruction. This mode stops the main clock oscillators, and thus stops the whole system. You clear stop mode with an interrupt. Because an external crystal oscillator takes time to restart after being stopped, an internal circuit provides a wait for the oscillator to stabilize after release of stop mode. If the microcontroller must start processing immediately after the interrupt, use halt instead.

In either halt or stop mode, all registers, flags, data-memory contents, I/O port-output latches, and output buffers remain unchanged.

You can release both halt and stop mode with either an interrupt or a device reset.

Figure 3. Releasing Halt Mode

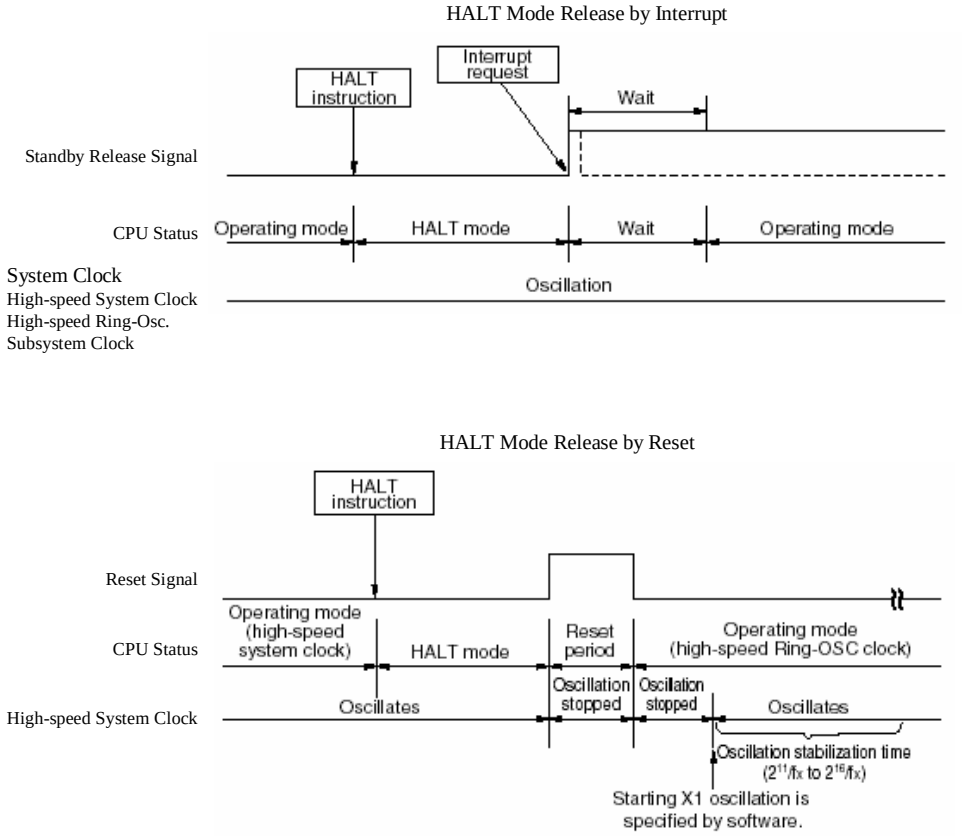
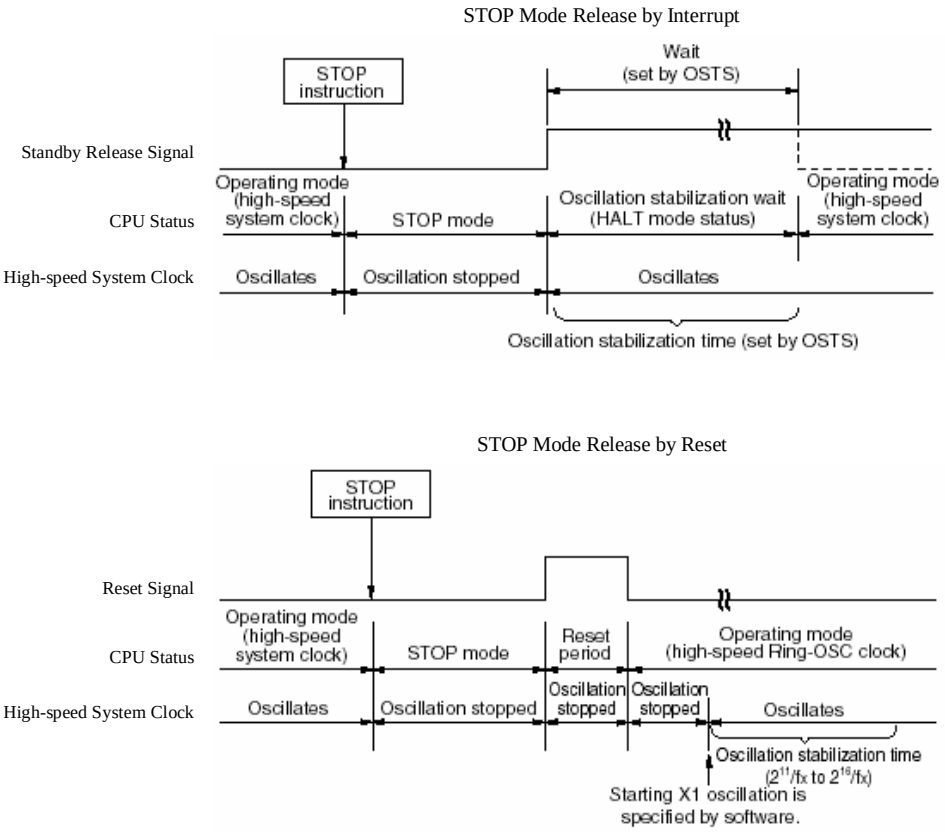


Figure 4. Releasing Stop Mode



2.2 Program Description and Specification

The demonstration program selects various clocks for the system clock and allows you to reduce clock speed and select standby modes. You make choices of clock, clock divider, display control, and standby mode from a menu. An ammeter measures the CPU current to verify that slower clock speeds result in lower power consumption.

A navigation switch allows you to select among program options. An external LCD displays program output. To indicate CPU operation in some power-down modes, an external buzzer or speaker provides beeps.

Figure 5. Demonstration Hardware Configuration

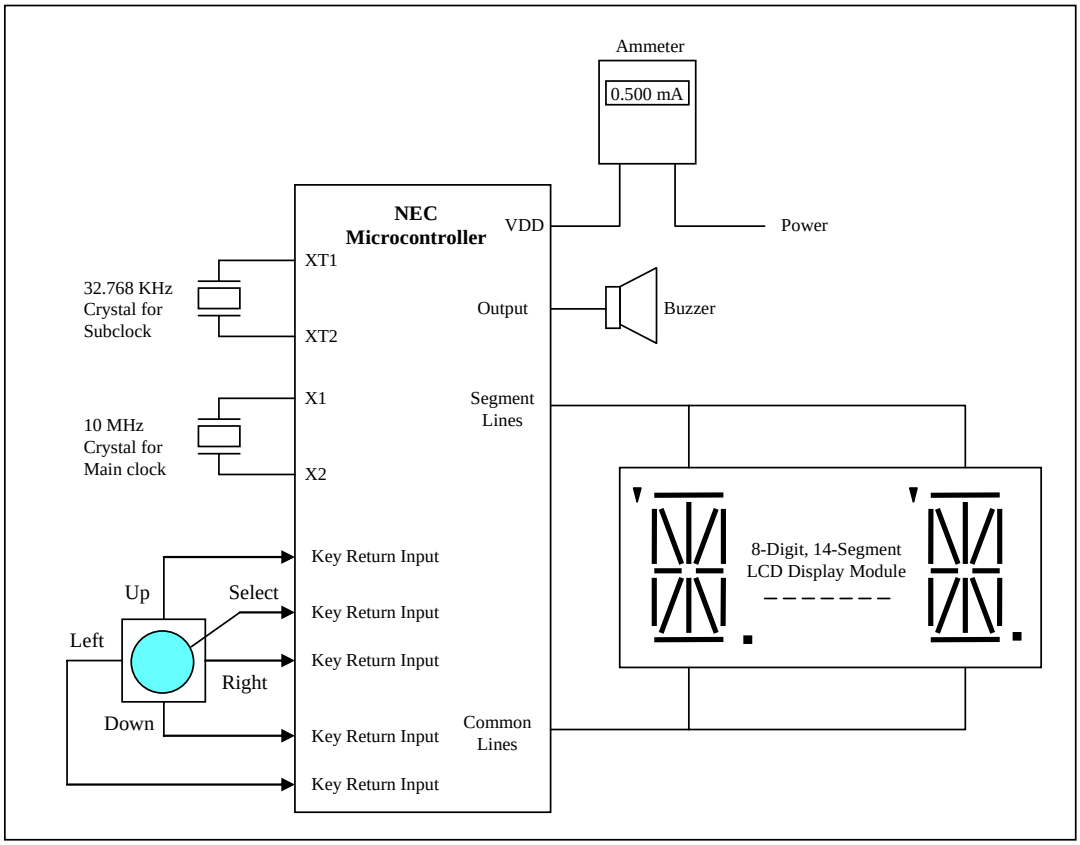
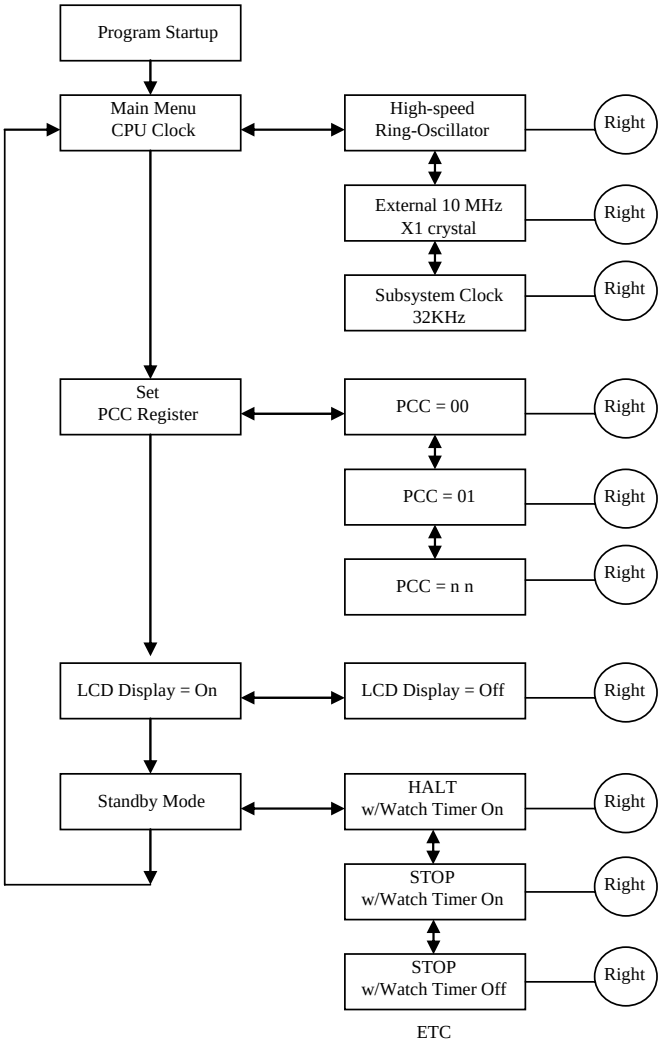


Figure 6. Flowchart for Demonstration Program



Specifications:

- The demonstration system allows use of the internal high-speed oscillator, 10-MHz X1 crystal, or 32.768-kHz subclock.
- Examples are shown for initializing an externally driven EXCLK clock, but this clock is not used in the demonstration program.
- You can set the PCC register for all possible divisions of the main CPU clock.
- A watch timer generates periodic interrupts and measures intervals.
- The system demonstrates halt and stop modes with selected CPU clocks, though stop is not available if running on the subclock.
- Halt and stop modes are shown with and without periodic interrupts.

2.3 Software Flow Charts

The demonstration program consists of the following major sections:

- Program initialization code, called before the main() program starts, which includes clock, key-return, and watch-timer initialization
- The main program loop, which displays menu options and responds to switches
- Submenu routines for CPU CLK, SET PCC, DISPLAY, and STANDBY menu items
- Subroutines invoked from submenus to set clock, PCC, display and standby modes
- Subroutines generated by the Applilet for watch timer and ICC0 operation
- Subroutines with user code for handling key-return and watch-timer interrupts (Applilet-generated stub interrupt-service routines, with user code added)
- Subroutines for LCD operation

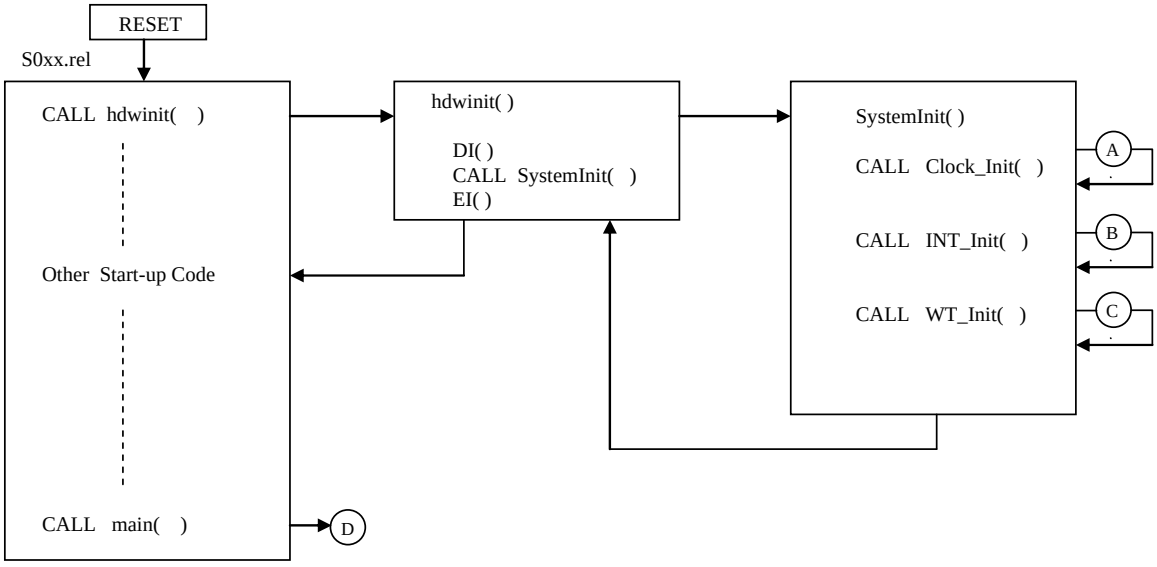
The flowcharts describe the initialization, the main program, clock and standby routines, key-return and watch-timer interrupts. Flowcharts are not included for IIC0 and LCD initialization, IIC0 communication, or LCD-data transfer. The software listings include this code, however.

2.3.1 Program Startup and Initialization

For 78K0 programs written in the C language, an object file such as s01.rel links to the user program and provides the startup code for the C program. The startup code calls a function named hdwinit(), where you can place hardware-initialization code.

When you use the Applilet to generate a C program for the 78K0, the tool automatically adds the hdwinit() function to the user program and calls the function SystemInit(). The SystemInit() function in turn calls initialization routines for each peripheral.

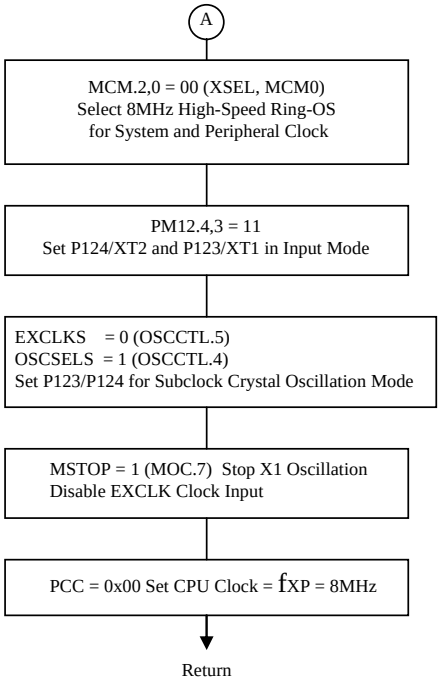
Figure 7. Flowchart for System Initialization



After the hdwinit() function finishes, the startup code calls the main() function of the user program. So when main() starts, peripheral initialization is complete for ports, key-return interrupt, and timers. Thus, main() does not need to call these initialization routines.

2.3.2 Clock_Init( ) – CPU Clock Initialization

Figure 8. Flowchart for CPU-Clock Initialization



SystemInit() first calls Clock_Init() to initialize the main, peripheral and subclocks. The assumption at this point is that the CPU is operating with default values in the clock-control registers. Specifically, the CPU and peripherals are operating from the 8-MHz internal high-speed oscillator, and the external clock and the subclock are not enabled.

The routine first sets the XSEL and MCM0 bits in the MCM register to select the main and peripheral clock. Setting these bits to zero (which should be the current state) selects the 8-MHz internal high-speed oscillator for both clocks.

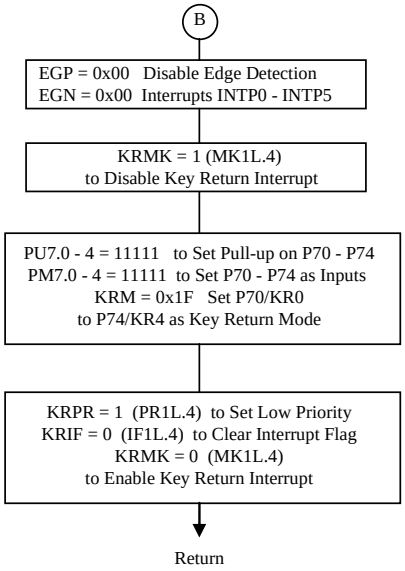
Next, the routine sets PM12 bits 4 and 3 to one, which specifies port pins P124/XT2 and P123/XT1 as inputs. This setting connects the 32.768-kHz external crystal. Clock_Init() sets the EXCLKS and OSCSELS bits in the OSCCTL register to 0,1, selecting XT1 oscillation mode on pins P124/XT2 and P123/XT1. At this point, the subclock begins oscillation.

Clock_Init() sets the MSTOP bit in the MOC register to 1 (the default) to stop X1 oscillation and disable the EXCLK input. This setting stops the external crystal oscillation or disables the driven external clock.

Finally, Clock_Init() sets the PCC register to 0x00, to select the fastest CPU clock by selecting f_{XP} (8 MHz) as the CPU clock.

2.3.3 INT_Init( ) – Key-Return Interrupt Initialization

Figure 9. Flowchart for Initializing Key-Return Interrupt



SystemInit() calls INT_Init() to set up the key-return function. The demonstration uses pins P70/KR0 through P74/KR4 as key-return inputs from the navigation switch. Since no other external interrupts are specified in the Applilet (more on this later), the routine sets the EGP and EGN registers to disable other external interrupts.

INT_Init() sets the key-return interrupt-mask bit, KRMK, to 1 to mask the interrupt while modifying other registers.

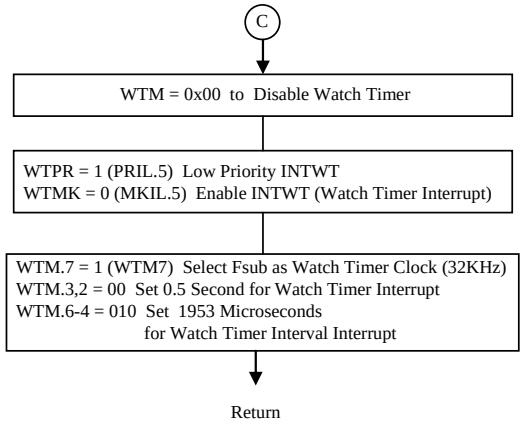
The routine writes the PU7 register with 0x1F to provide the P70-P74 port pins with pull-up resistors. The routine writes the same value to the port-mode register (PM7) to make the pins inputs. The routine writes 0x1F to the key-return mode register (KRM) to specify that key-return interrupt sources KR0 through KR4 cause a key-return interrupt. A negative edge on any of these pins invokes the interrupt.

INT_Init() sets the key-return interrupt priority to low, clears the interrupt flag (KRIF), and sets the mask bit to zero to enable the key-return interrupt (INTKR).

See the section on the MD_INTKR( ) key-return interrupt-service routine to see the actions taken on key-return interrupts.

2.3.4 WT_Init( ) – Watch-Timer Initialization for Square-Wave Generation

Figure 10. Flowchart for Initializing Square-Wave Generator



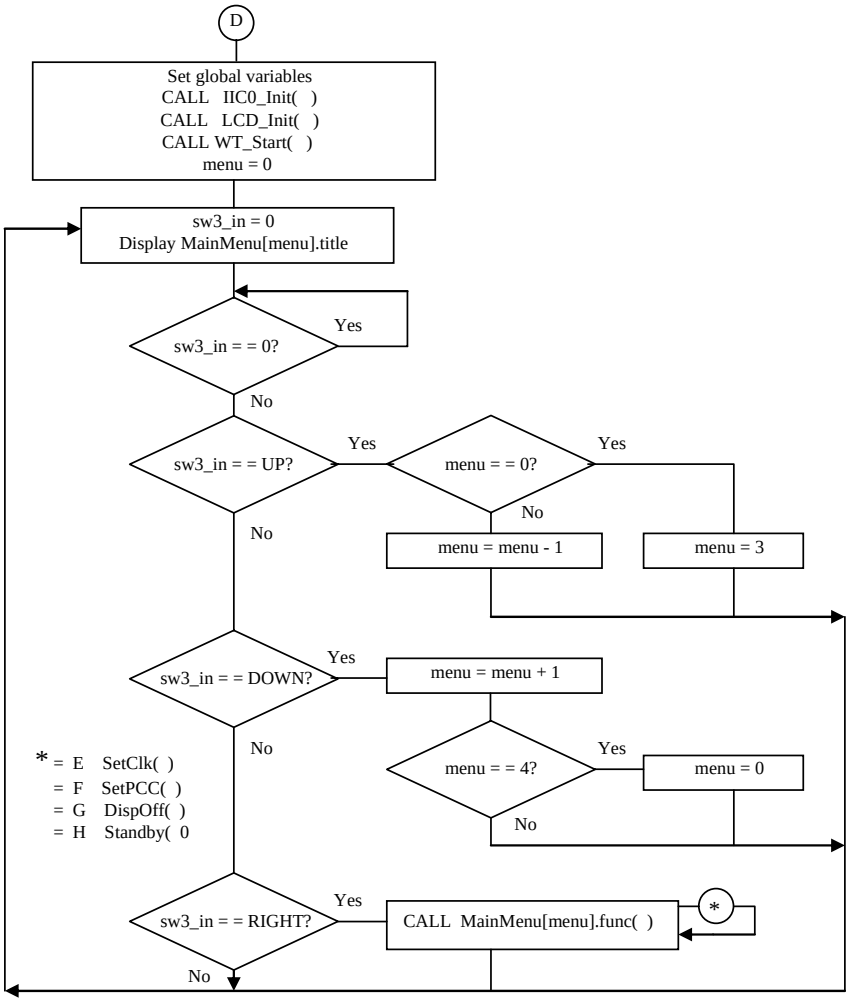
SystemInit() calls WT_Init() to initialize the watch timer for both watch-timer and interval-timer functions. First the routine disables the watch timer by setting WTM.0 to zero.

WT_Init() sets the WTPR (PR1L.5) and WTMK (MK1L.5) bits to give the WTM watch-timer interrupt (INTWT) low priority and to enable the interrupt. The routine leaves the priority and mask bits controlling the watch-timer interval interrupt (INTWTI) in the default state, so the INTWTI interrupt remains disabled.

WT_Init() sets WTM.7 to 1 to set the 32.768-kHz subclock as fw—the time base for WTM. The routine sets WTM bits 3 and 2 to zero, which sets the interval for the watch-timer interrupt as 16384/fw, or every 0.5 seconds. The routine sets WTM bits 6 through 4 to 010, which sets the interval-timer interrupt to 64/fw, or 1953 microseconds (approximately 2 milliseconds). The processor uses this interval to time delays in LCD initialization and switch debouncing.

2.3.5 Main( ) – The Main Program – Power-Down Functions

Figure 11. Flowchart for Power-Down Functions



Calling main() starts the program. Main() sets some global variables used to track the state of the clock:

- **g_clock** = CLK_HSR currently selected clock (CLK_HSR, CLK_EX, or CLK_SUB)
- **g_mainclock** = CLK_HSR main system clock (CLK_HSR or CLK_EX)
- **g_main_on** = ON whether main clock is running or not
- **g_beep** = OFF whether to beep periodically in watch-timer ISR

Main() then calls IIC0_Init() to start the IIC0 peripheral for communication with the LCD controller, calls LCD_Init() to initialize the LCD, and calls WT_Start() to start the watch timer.

The routine sets the **menu** variable to zero. This variable tracks the currently selected item. Then the routine enters the main program loop. The main loop clears the **sw3_in** variable, which reflects the

navigation switch being pressed. The main loop also displays the title of the current menu item from the MainMenu table, which contains the items shown below.

Table 3. Main Menu Table

Index (n)	MainMenu[n].title	MainMenu[n].func	Function Operation
0	“CPU CLK “	SetClk()	Set CPU clock source
1	“SET PCC “	SetPCC()	Set PCC register for main clock divider
2	“DISPLAY “	DispOff()	Turn LCD and ICC0 off
3	“STANDBY “	Standby()	Select standby mode

The loop waits until the **sw3_in** variable is not zero (so the program waits until you press a switch). If you press a switch, the key-return interrupt (INTKR) occurs. The MD_INTKR interrupt-service routine checks the switch input and updates the **sw3_in** variable with the value of the debounced switch input.

Interrupts also occur once every 0.5 seconds from the watch-timer interrupt (INTWT), but the MD_INTWT interrupt-service routine takes no action.

Once a valid switch input is debounced, the main program loop sees a non-zero value in **sw3_in** and proceeds. The routine checks to see which switch you pressed and takes appropriate action.

If you press the UP switch, the program decrements the **menu** variable, wrapping around from 0 to 3.

If you press the DOWN switch, the program increments the **main** variable, wrapping around from 3 to 0.

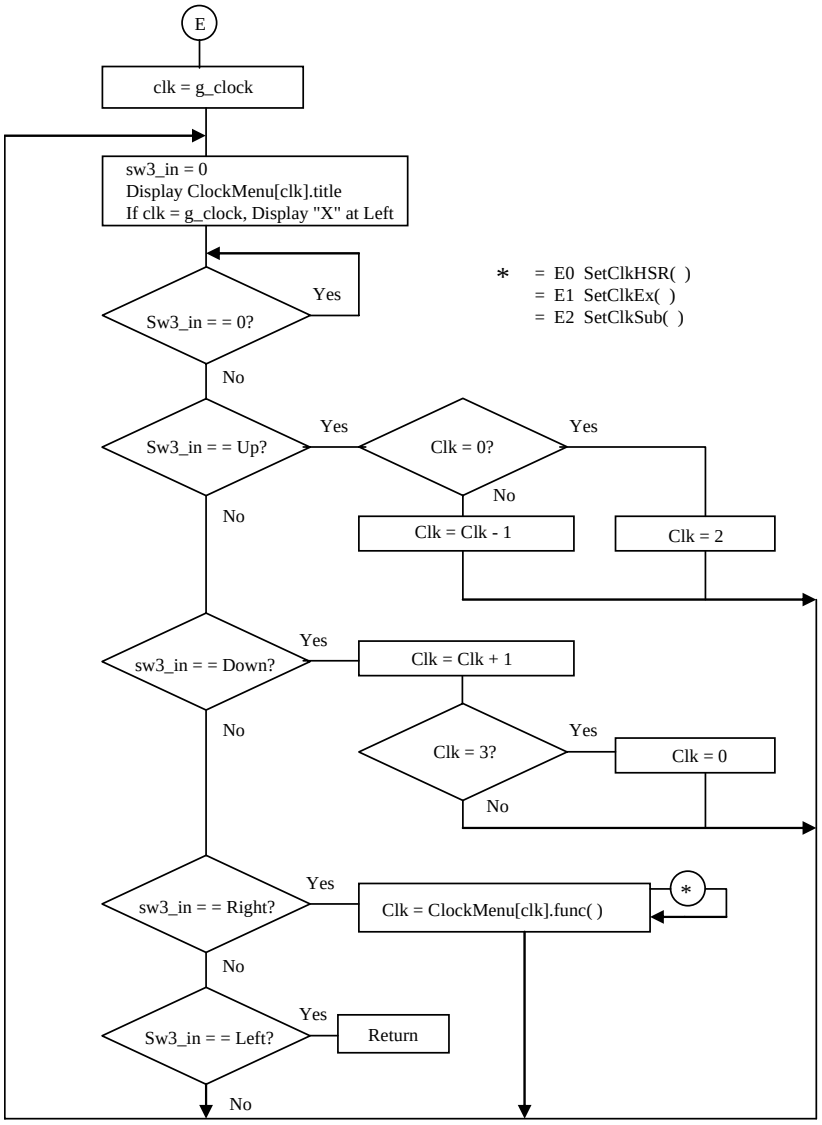
If you press the RIGHT switch, the program calls the function associated with the value of the **menu** variable. The routines called are shown in the table above.

After processing the switch inputs (and after the function called by RIGHT returns), the program goes to the top of the loop, clears the switch input, displays the currently-selected menu item, and waits again for a key press.

In this way, the UP and DOWN switches cycle through the menu choices, and the RIGHT key executes the current choice. Some of the submenu routines called from main() use a similar menu-processing strategy.

2.3.6 SetClk( ) – Select CPU Clock Source

Figure 12. Flowchart for CPU Clock Source Selection



The SetClk() routine, called from main(), sets the source of the CPU clock. SetClk() sets the local variable **clk** to the current state of the global clock variable, **g_clock**, which contains one of the three values shown in the table below.

Table 4. Clock-Source Selection

Symbolic variable	Value	CPU clock source
CLK_HSR	0	Internal high-speed oscillator (8 MHz)
CLK_EX	1	External clock (X1 crystal or driven EXCLK)
CLK_SUB	2	XT1/XT2 subclock crystal (32.768 kHz)

The SetClk() routine then enters a menu processing loop, much like the one used in main(). The menu structure ClockMenu controls operation and the **clk** variable is an index into this structure.

At the top of the loop, the routine sets **sw3_in** to zero to clear the previous switch input. The LCD displays the title of the menu item currently selected. If the current value of **clk** matches **g_clock** (as it does at the start), the routine displays an "X" at the left side of the display.

Table 5. ClockMenu Contents

Index (n)	ClockMenu[n].title	ClockMenu[n].func	Function Operation
0	" C HSR "	SetClkHSR()	Set CPU clock to internal high-speed oscillator
1	" C EX "	SetClkEx()	Set CPU clock to external X1 crystal oscillator
2	" C SUB "	SetClkSub()	Set CPU clock to subclock

The menu processing loop handles switch inputs, changing the value of the **clk** variable to a new index, executing the selected function to change the clock, or returning to the main menu.

The UP switch cycles through the CPU clock choices in reverse order: 0→2→1→0. The DOWN switch cycles **clk** through the CPU clock choices in forward order: 0→1→2→0.

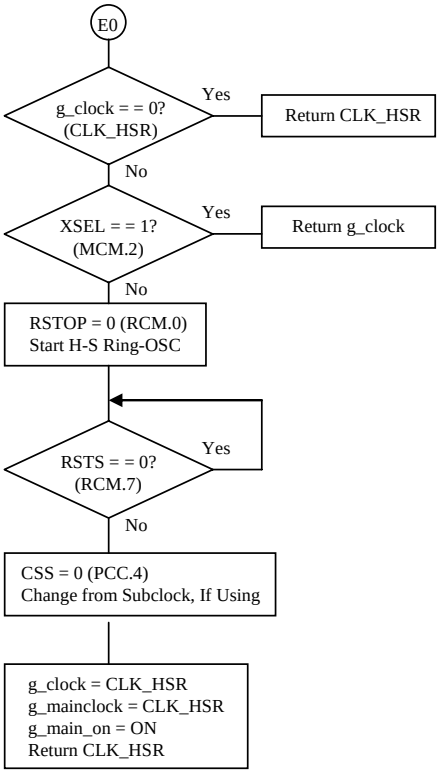
The RIGHT switch calls the selected function. If the function can change the clock (not all changes are possible), the routine updates the global variable **g_clock**. The functions return the value of the current CPU clock and updates the **clk** variable with the (possibly changed) value.

The LEFT switch returns from the SetClk() routine to the main() routine, to return to the main menu.

After processing the switch inputs (and after the function called by RIGHT returns), the program goes to the top of the loop, clears the switch input, displays the currently-selected menu item, and waits again for a key press.

2.3.7 SetClkHSR( ) – Set CPU Clock to Internal High-Speed Oscillator

Figure 13. Flowchart for Setting CPU Clock for Internal High-Speed Oscillator



The SetClkHSR() routine sets the CPU clock to the internal high-speed oscillator (8 MHz) and returns the value of the CPU clock. If the CPU clock is already CLK_HSR, the routine returns with no action.

If the XSEL bit (bit 2 in the MCM register) is one, the program has changed the main CPU clock and peripheral clock from the power-up default (internal high-speed oscillator) to the external EXCLK or X1 crystal oscillator. Since you can change the XSEL only once after a reset, you cannot change it back to the internal high-speed oscillator for the peripherals. Thus, the routine does not change the CPU clock or the peripheral clock. In this case, the routine returns the current state of the clock (CLK_EX or CLK_SUB).

If XSEL is zero, the main clock is still set to the internal high-speed oscillator, but the CPU may be operating on the 32.768-kHz subclock. But because SetClkHSR() would have already returned if **g_clock** was CLK_HSR or CLK_EX, clearly the CPU is running on the subclock. The routine needs to change from operating on the subclock, with the internal high-speed oscillator stopped, to operating on the internal high-speed oscillator.

First the routine starts the internal high-speed oscillator by clearing the RSTOP bit in the RCM register. The routine checks the state of the RSTS bit in the RCM register (which is 1 when the oscillator is stable), and waits until this bit is set.

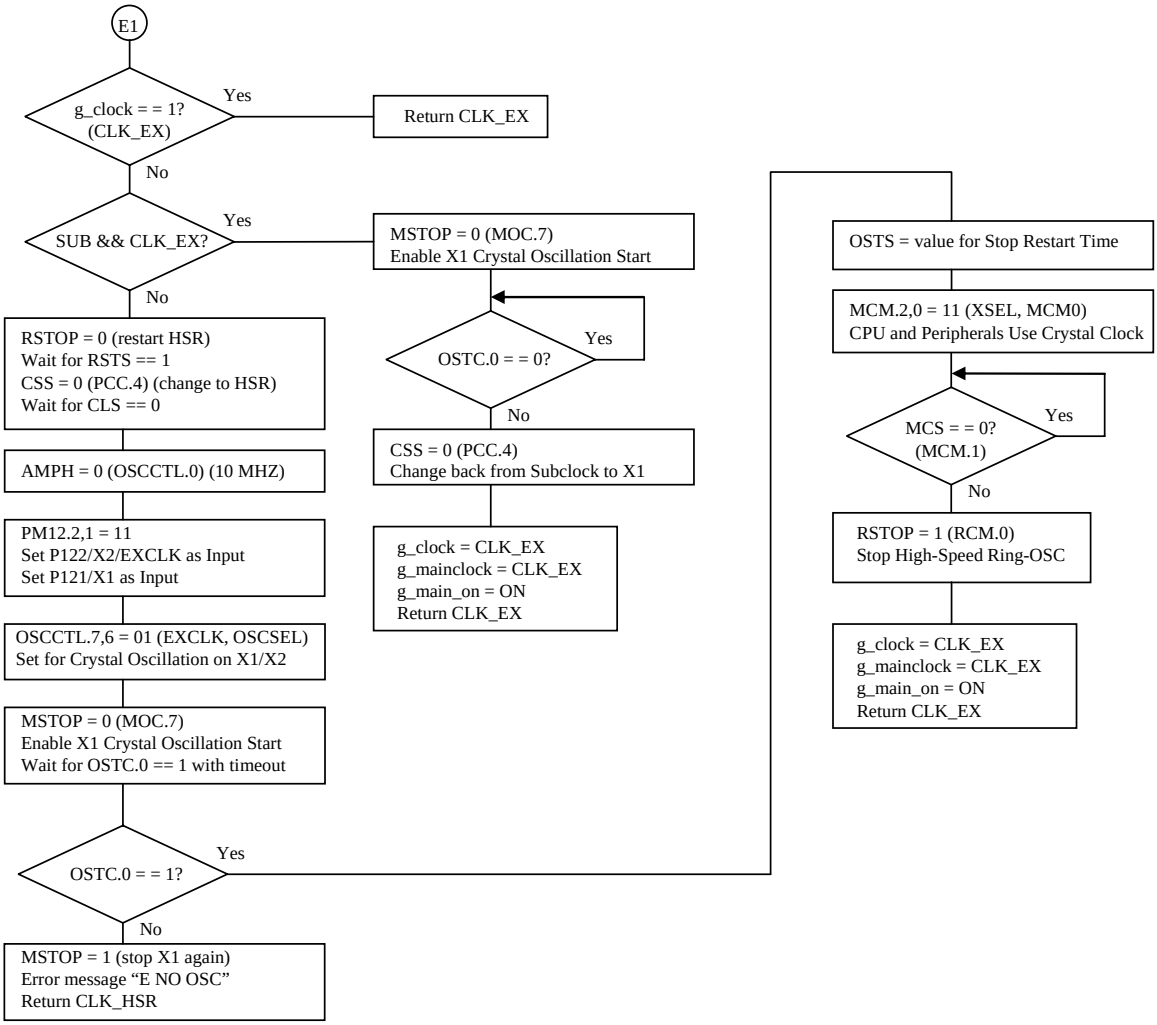
Note that you can skip this wait. The internal high-speed oscillator begins operation immediately after RSTOP is set to zero, at about 5 MHz. It takes some time for the oscillator to reach its stable operating range of 8 MHz. If you do not need an accurate clock, however, the program could proceed without waiting for RSTS to be set.

After the oscillator is stable, the routine sets the CSS bit in the PCC register to zero, which switches the CPU clock source from the subclock to the oscillator. The subclock keeps operating, however.

The routine updates global variables **g_clock** and **g_mainclock**, sets **g_main_on** to ON (to indicate that the main clock is on), and returns the value CLK_HSR to SetClk().

2.3.8 SetClkEx( ) – Set CPU Clock to X1/X2 Crystal Oscillator

Figure 14. Flowchart for Setting CPU Clock to Crystal Oscillator



The SetClkEx() routine sets the CPU clock to the external X1 crystal and returns the value of the CPU clock. In the demonstration program, this menu item selects a 10-MHz external X1 crystal on the P121/X1 and P122/X2/EXCLK input pins. See the next section for a version to use if the external clock is a driven clock signal on the P122/X2/EXCLK pins.

If the CPU clock is already set to CLK_EX (as reflected in the **g_clock** variable), the routine returns CLK_EX and takes no action. If **g_clock** is not CLK_EX, it is either CLK_HSR (internal high-speed oscillator) or CLK_SUB (subclock). SetClkEx() switches the CPU from either of these clock sources to the external X1/X2 crystal oscillator.

If **g_clock** is CLK_SUB and **g_clockmain** is CLK_EX, the crystal has already been selected and the clock switched to the subclock. In this case, the routine sets the MSTOP bit to zero to restart the main clock, and waits for the X1 oscillation to stabilize, indicated when bit zero of the OSTC register becomes one. The routine switches the clock from the subclock to the main clock by setting the CSS bit in the PCC register to zero. The SetClkEx() routine then returns to the caller with the clock set to CLK_EX.

If the routine goes beyond the previous two checks, then either **g_clock** is CLK_HSR (running on the internal high-speed oscillator), or **g_clock** is CLK_SUB (running on the subclock) and **g_clockmain** is CLK_HSR. In either of these cases, the routine now needs to start the X1 crystal oscillator for the first time.

If the program is running on the subclock, the routine needs to switch the clock back to the internal high-speed oscillator. This switch is done by setting the RSTOP bit to zero and waiting for RSTS to be one, and then setting CSS to zero and waiting for CLS to be zero.

To begin setting up the X1 crystal clock, first SetClkEx( ) sets the AMPH bit (OSCCTL register bit 0) to zero for operation from 2 to 10 MHz. If the external crystal oscillates at a higher speed than 10 MHz, set the bit to one. Note that when AMPH is first set to 1, the CPU clock stops for a minimum of 5 microseconds to allow for clock stabilization at the higher frequencies. This delay also occurs when exiting stop mode.

SetClkEx( ) sets the PM12 register bits 2 and 1, which control the mode of the P122/X2/EXCLK and P121/X1 pins. Setting these bits to 1 configures the pins as inputs.

The routine sets the EXCLK and OSCSEL bits in the OSCCTL register for EXCLK = 0, OSCSEL = 1, to select X1/X2 inputs as crystal-oscillator mode (rather than I/O port or driven-clock input mode). At this point, the external crystal is selected, but is not oscillating, and the CPU is operating on the internal high-speed oscillator or subclock.

The routine clears the MSTOP bit in the MOC register to zero, thus starting oscillation of the external crystal. Once oscillation starts, the microcontroller changes bits in the OSTC register (from the reset value of 0x00) to indicate the stability of the oscillation. Specifically, bits 4, 3, 2, 1 and 0 are set in sequence after a certain number of X1 oscillations. By checking for a particular bit to change to a 1, the program can wait for a specific oscillator-stabilization time. For example, by waiting for OSTC bit 0 to change to 1, the program waits a minimum of 6.55 milliseconds at 10 MHz.

The demonstration program times out if OSTC bit zero does not become one after a number of checks designed to be well past the necessary start time. If the timeout occurs, the routine displays the error message “E NO OSC” to indicate that no oscillation of the X1 crystal has been detected, and waits for a key to be pressed. Once you press a key, the program returns without changing the clock source.

Once OSTC.0 is 1, the external crystal clock is enabled and stable, but the CPU is still running on the high-speed oscillator. Before switching to the crystal clock, the routine sets the OSTS register to the value needed to control the X1 oscillator's restart when exiting stop mode.

To switch to the X1 clock, the routine sets the XSEL and MCM0 bits in the MCM register to 1. This setting chooses the external crystal input as the clock for the CPU and peripherals.

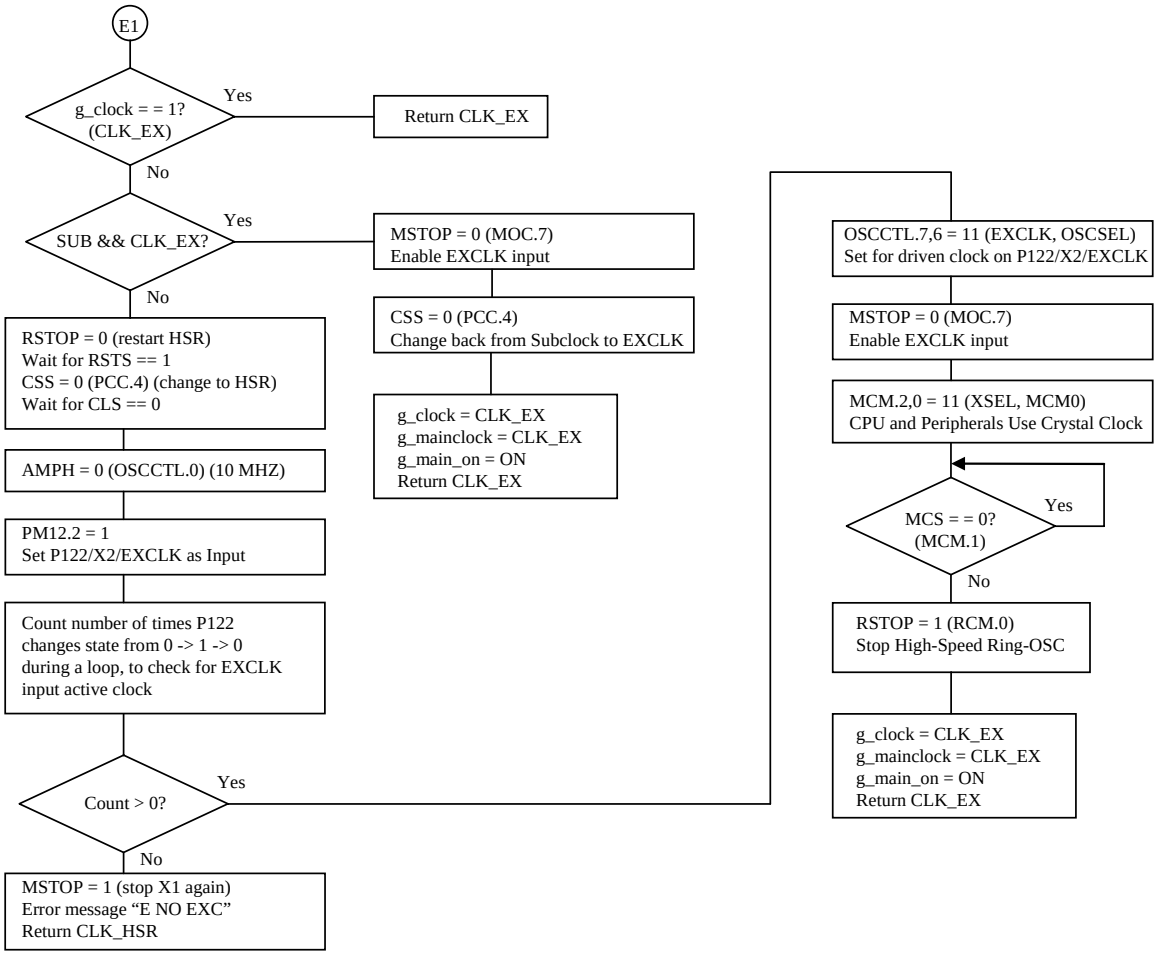
The MCS bit in the MCM register indicates whether the main system clock is operating on the internal high-speed oscillator (zero), or the EXCLK or X1/X2 external clock (one). The SetClkEx() routine waits for this bit to become one to confirm clock switch-over.

Once the MCM bit is one, the CPU is operating on the X1 external crystal clock. Since the internal high-speed oscillator is no longer needed, the routine sets the RSTOP bit in the RCM register to one, stopping the internal high-speed oscillator.

The routine updates the global variables **g_clock** and **g_mainclock** to CLK_EX to reflect the X1/X2 crystal input, sets **g_main_on** to ON to indicate main clock on, and returns the value CLK_EX to SetClk().

2.3.9 SetClkEx( ) – Alternate – Set CPU Clock to EXCLK Input

Figure 15. Flowchart for Setting CPU to EXCLK



The previous SetClkEx() routine set the CPU clock to the external X1 crystal on the P121/X1 and P122/X2/EXCLK pins. You can also use an external driven-clock signal (EXCLK) connected to the P122/X2/EXCLK pin.

This alternate version of SetClkEx() demonstrates how to switch to a driven 6-MHz external clock. This version is not compiled in the demonstration program; the code exists but is commented out. Choose only one version of SetClkEx(), depending on whether the external clock is a crystal or a driven clock.

If the CPU clock is already set to the EXCLK input (as reflected in the **g_clock** variable), the routine returns CLK_EX and takes no action. If **g_clock** is not CLK_EX, the variable is either CLK_HSR (internal high-speed oscillator) or CLK_SUB (subclock). SetClkEx() switches from either of these clocks to the EXCLK input.

If **g_clock** is CLK_SUB and **g_clockmain** is CLK_EX, the external EXCLK has already been checked and selected, and the clock switches to the subclock. In this case, the routine restarts the main clock by setting the MSTOP bit to zero. (It is not necessary to wait for bit zero of the OSTC register to become one, as is done for the crystal input.) The routine switches the clock from subclock to the main clock by setting the CSS bit in the PCC register to zero. The SetClkEx() routine then returns to the caller with the clock set to CLK_EX.

If the routine goes beyond the previous two checks, then either **g_clock** is CLK_HSR (running on the internal high-speed oscillator), or CLK_SUB (running on subclock) and **g_clockmain** is CLK_HSR. In either of these cases, the program has not selected the EXCLK input before and must start EXCLK for the first time.

If the program is running on the subclock, the routine switches the clock back to the internal high-speed oscillator. This switch is done by setting the RSTOP bit to zero and waiting for RSTS to be one, then setting CSS to zero and waiting for CLS to be zero.

The routine then sets up the EXCLK input clock, first setting the AMPH bit (OSCCTL register bit 0) to zero. (This bit needs to be set to one if the external clock frequency is above 10 MHz. The demonstration program uses a 6-MHz driven input.)

SetClkEx() sets the PM12 register bit 2 to 1 to configure the P122/X2/EXCLK pin as an input.

The program checks for oscillation on the P122/X2/EXCLK input. The routine runs a loop, inputting the value of the P122 pin and checking it against the previous value. If the values are different, the pin has changed state, and the routine increments a counter.

After completing the loop, the routine checks the counter. If the count is zero, there is no oscillation on the P122/X2/EXCLK input. The program displays the error message “E NO EXC” and returns, leaving the clock set to the internal high-speed oscillator.

If EXCLK oscillation is detected, the routine sets the EXCLK and OSCSEL bits in the OSCCTL register to 1, to select the P122/X2/EXCLK input as driven-clock input mode (rather than I/O-port or crystal-oscillator mode). The external EXCLK is now selected but not enabled, and the CPU is still operating on the internal high-speed oscillator.

SetClkEx() clears the MSTOP bit in the MOC register, enabling the EXCLK input. The routine sets the XSEL and MCM0 bits in the MCM register to 1, selecting the EXCLK input as the clock to the CPU and the peripherals.

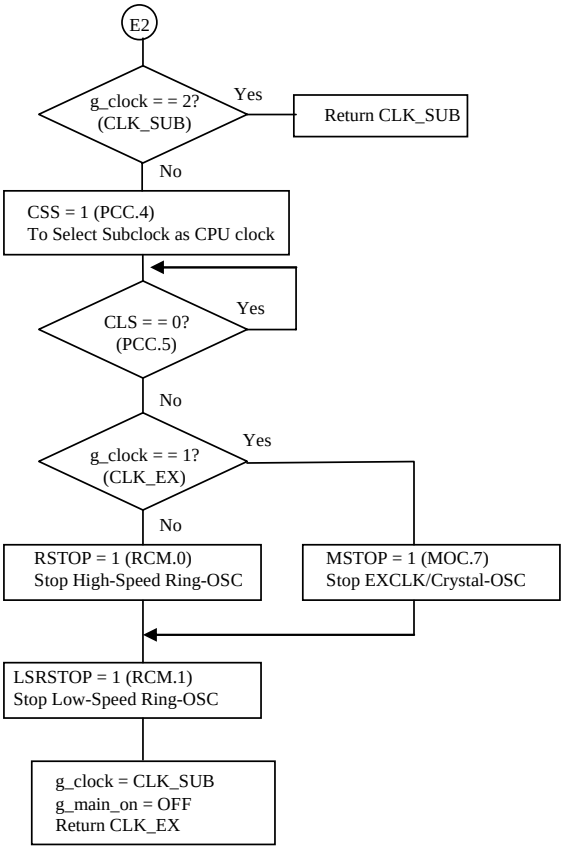
The MCS bit in the MCM register indicates whether the main system clock is operating on the internal high-speed oscillator (if zero), or the EXCLK or X1/X2 external clock (if one). The SetClkEx() routine waits for this bit to become one, to confirm clock switch-over.

Once the MCM bit is one, the CPU is operating on the driven EXCLK external clock, and the internal high-speed oscillator is no longer needed. The routine sets the RSTOP bit in the RCM register to one, to stop the internal high-speed oscillator.

SetClkEx() updates the global variables **g_clock** and **g_mainclock** to CLK_EX to reflect the EXCLK input, sets **g_main_on** to ON to indicate that the main clock is on, and returns the value CLK_EX to SetClk().

2.3.10 SetClkSub( ) – Set CPU Clock to Subclock

Figure 16. Flowchart to Set CPU Clock to Subclock



The SetClkSub() routine sets the CPU clock to the 32.768-kHz subclock and returns the value of the CPU clock.

If the CPU clock is already set to the subclock (as reflected in the **g_clock** variable), the routine returns CLK_SUB and takes no action. If **g_clock** is not CLK_SUB, it is either CLK_HSR (internal high-speed

oscillator) or CLK_EX (external EXCLK or X1/X2 clock), and SetClkSub() switches the CPU to the subclock.

SetClkSub() sets the CSS bit in the PCC register to 1, selecting the subclock as the CPU clock. The CLS bit in the PCC register reflects whether the CPU is operating from the main system clock (CLS=0) or from the subclock (CLS=1). The SetClkSub() routine waits for the CLS bit to become 1, to ensure that the CPU is operating on the subclock before proceeding. At this point, the main system clock(s) can be stopped.

If the system was operating on the external clock, the routine sets the MSTOP bit in the MOC register to one. This setting stops the external X1 crystal (or disables the EXCLK input, if you use this version of the demonstration software).

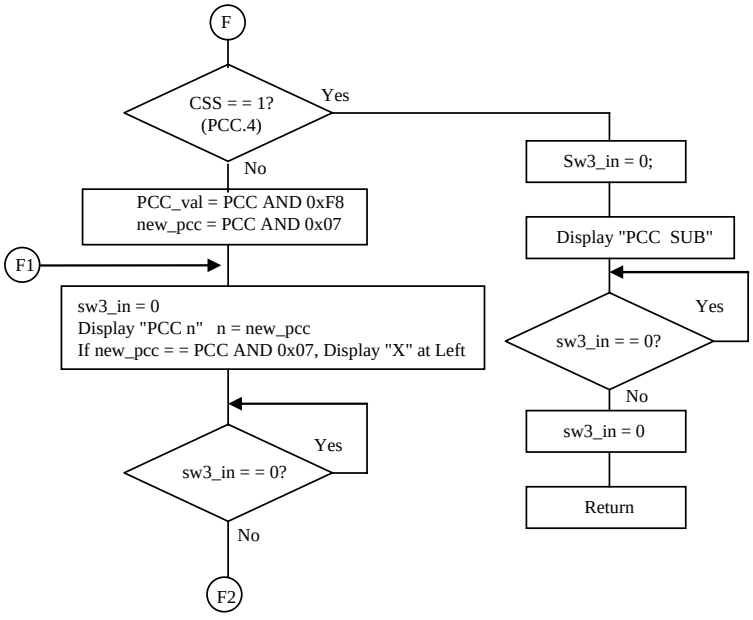
If the system was using the internal high-speed oscillator, the routine sets the RSTOP bit in the RCM register to one to stop the internal high-speed oscillator.

The demonstration program also stops the internal low-speed oscillator by setting the LSRSTOP bit in the RCM register to 1. At this point, the subclock is operating, the CPU is running from the subclock, and all other clocks are stopped.

SetClkSub() sets the global variables **g_clock** to CLK_SUB to reflect the subclock, sets **g_main_on** to OFF to indicate that the main clock is off, and returns the value CLK_SUB to SetClk().

2.3.11 SetPCC( ) – Set PCC Register for Main Clock Division

Figure 17. Flowchart for Setting Main Clock Divisions—Part 1



The SetPCC() routine displays the current setting for the PCC register divider and allows you to change the divider.

When the CPU is operating on the main system clock (internal high-speed oscillator, X1 crystal. or EXCLK clocks), the lower three bits of the PCC register determine the operating frequency the CPU uses in executing instructions. The table below shows the settings for the three low bits of PCC and the selected division of the main system clock.

Table 6. Main Clock Divisions

PCC.2-0	CPU clock (f_{CPU})	f_{CPU} with f_{XP} = 8 MHz	Instruction time ($2/f_{CPU}$)
0	f_{XP}	8 MHz	0.25 µsec
1	$f_{XP}/2$ (default)	4 MHz	0.5 µsec
2	$f_{XP}/4$	2 MHz	1.0 µsec
3	$f_{XP}/8$	1 MHz	2.0 µsec
4	$f_{XP}/16$	500 kHz	4.0 µsec
Other values prohibited			

When the CPU is operating on the subclock, the CPU clock frequency is 32.768 kHz, and the minimum instruction time is 122.1 µs. The subclock is selected by setting the CSS bit in the PCC register (PCC.4) to one. The PCC register's lower three bits have no effect when the CPU is operating on the subclock.

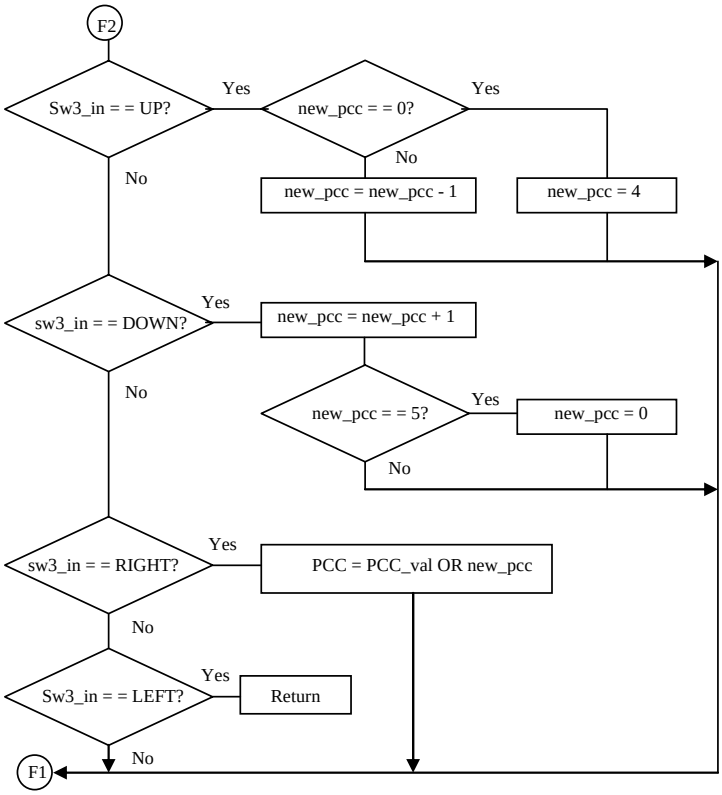
The SetPCC() routine first checks if CSS is 1, and if so, does not change the lower three bits. The routine displays "PCC SUB" and then waits for a switch to be pressed. Once you press a switch, the routine clears the switch value and returns to the main() program loop.

If CSS is 0, then the SetPCC() routine displays and alters the setting. The routine saves the upper five bits of PCC in the variable **pcc_val**, and places the lower three bits in the **new_pcc** variable, which now has a value from 0 to 4.

At the top of the SetPCC() loop, the routine clears the switch input and displays " PCC n ", where n is the value of **new_pcc**. If **new_pcc** is the same as the lower three bits of PCC (as it is at the start of the routine), the routine displays an X at the left of the display to show that this is the current value of PCC.

The SetPCC() routine then waits until you press a switch.

Figure 18. Flowchart for Setting Main Clock Divisions—Part 2



When you press the UP or DOWN switches, SetPCC() decrements or increments the **new_pcc** variable, wrapping around from 0→4 or 4→0. The routine then returns to the top of the loop and displays " PCC n" where n is the changed value of **new_pcc**. You can cycle through all possible values of the lower three bits of PCC. Whenever the current value of PCC matches **new_pcc**, the routine displays the X.

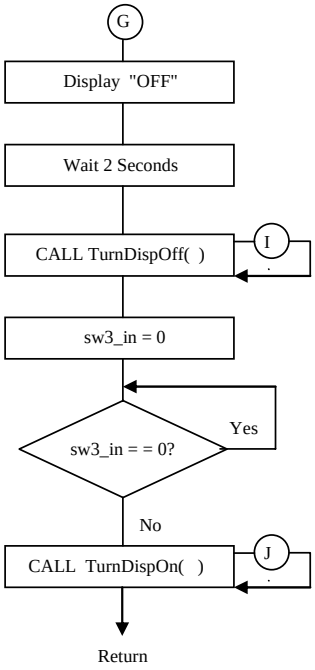
When you pres the RIGHT switch, the routine sets the PCC register to the logical OR of the current **new_pcc** variable for the lower three bits, and the **pcc_val** variable with the upper five bits. The SetPCC() routine then returns to the top of the loop and shows that PCC matches the current value of **new_pcc**.

When you press the LEFT switch, the SetPCC() routine returns to the main() program loop.

Using these settings, you can observe the power-consumption effects of changing CPU clock speed. Simply watch the current on the ammeter while changing the PCC divider. The fastest CPU speed, with PCC=0, should show the most power consumption. The slowest CPU speed (on the main clock), with PCC=4, should show the least.

2.3.12 DispOff( ) – Turn LCD and IIC0 Peripheral Off

Figure 19. Flowchart for Turning LCD and IIC0 Peripheral Off



Main() calls the DispOff() routine when you select the “DISPLAY” menu item. To demonstrate power savings when the display is disabled, the routine turns off the LCD controller and the IIC0 peripheral used internally to communicate with the LCD controller.

The routine displays “OFF” on the display for two seconds to let you know that the display is turning off, and then calls the TurnDispOff() routine to disable the display.

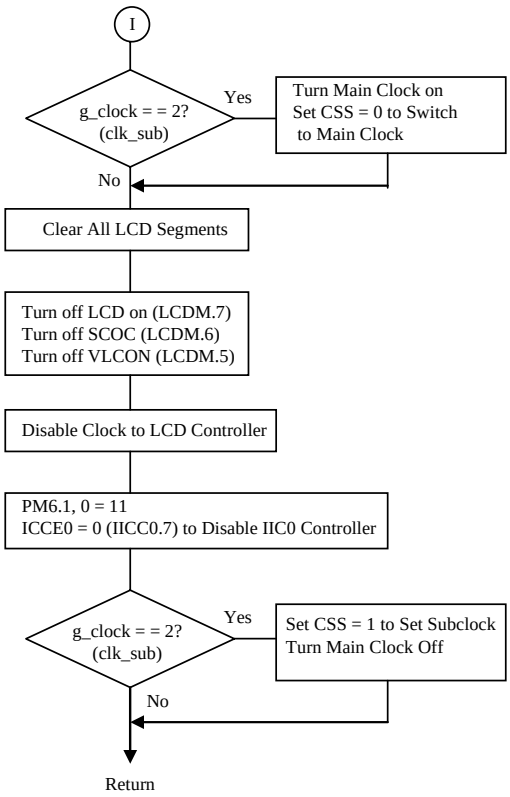
DispOff() clears the **sw3_in** variable to show no switch down and waits for **sw3_in** to be non-zero, indicating that a key has been pressed.

Once you press a key, DispOff() calls TurnDispOn() to turn the LCD back on, and returns to the main() routine.

Note that both TurnDispOff() and TurnDispOn() turn on the main clock (if the clock is off) and set the CPU to run from the main clock, thus increasing power consumption. This increase is brief for TurnDispOff(), but TurnDispOn() takes four seconds to initialize the display. When measuring power consumption, wait for a second after the display goes off. Disregard power measurements after pressing a key, even though the display is still blank for four seconds.

2.3.13 TurnDIspOff() – Turn Display Off

Figure 20. Flowchart for Turning Off Display



Both DispOff() and StandbyXxxxN() use TurnDispOff() to turn off the microcontroller's LCD controller and IIC0 peripheral. IIC0 is used internally to communicate with the LCD controller for register and segment-memory reads and writes.

The LCD controller uses a clock to multiplex the display and to boost voltages. In the demonstration program, the LCD-controller clock is based on the subclock. The IIC0 peripheral operates on the main system clock and requires that the CPU operate interrupt-service routines in a timely fashion to update the display.

When the demonstration program is operating on the subclock, the main system clock is off, and the IIC0 controller cannot communicate. Therefore, when **g_clock** is CLK_SUB, TurnDispOff() must enable the main clock in order to communicate with the LCD controller. Also, in order to process instructions quickly for interrupt service, TurnDispOff() switches the CPU to the main clock.

Once the clock switches, the routine clears all LCD segments, blanking the display. TurnDispOff() then writes to the LCD controller's LCDM register to turn off the bits LCDON (disables display), SCOC (connects all segment and common lines to ground), and VLCON (turns off voltage boost). TurnDispOff()

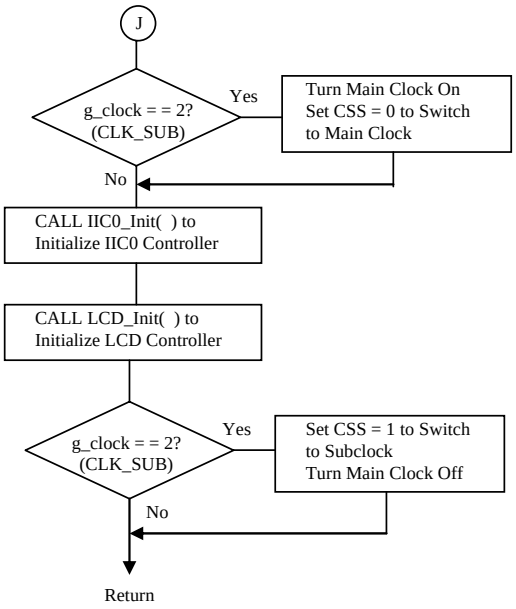
writes to the CKS register to turn off the LCD controller clock, and to the PM14 register to disable the clock output.

To disable the IIC0 controller, first TurnDispOff() sets PM6 register bits 0 and 1 to 1. These settings make the SDA0 and SCL0 pins inputs. Then the routine writes to the IICC0 register to clear the ICCE0 bit, which disables the IIC0 controller.

Then, if the system was initially running on the subclock, TurnDispOff() switches the CPU clock back to the subclock and turns the main clock off.

2.3.14 TurnDIspOn() – Turn Display On

Figure 21. Flowchart for Turning on Display



DispOff() and StandbyXxxxN() use TurnDispOff() to reinitialize the LCD controller and IIC0 peripheral after TurnDispOff() has disabled the display.

TurnDispOn() first turns the main clock on and switches the CPU clock to the main clock (if currently running on the subclock). Please see the previous section on TurnDispOff() for information about switching to the main clock.

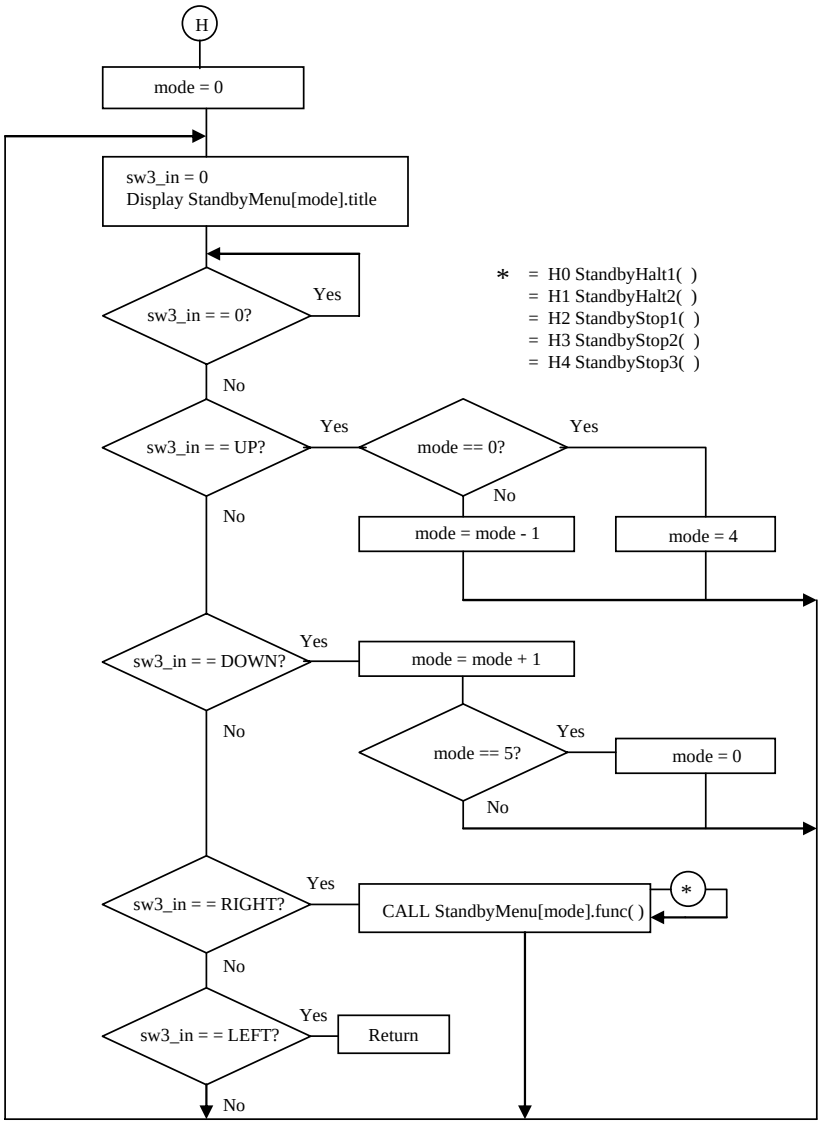
Then TurnDispOn() calls IIC0_Init() to reinitialize the IIC0 controller and calls LCD_Init() to initialize the LCD controller.

Initialization of the LCD controller in LCD_Init() involves a wait of four seconds, to allow LCD voltage boosting to reach proper levels. When TurnDispOn() is called, the display remains blank, and the CPU runs from the main clock for this period of time.

After initializing the IIC0 controller and LCD controller, TurnDispOn() returns the CPU to the subclock and turns off the main clock.

2.3.15 Standby( ) – Select Standby Mode

Figure 22. Flowchart for Selecting Standby Modes



The Standby() routine, called from main(), puts the processor into one of five standby modes. Two of these mode changes use the halt instruction, and three use the stop instruction. Standby() is similar to the main() and SetClk() routines in using a menu structure to process switch inputs.

The routine sets the **mode** variable to zero at the start of the routine. This variable is then used as an index into the StandbyMenu structure. At the top of the loop, Standby() sets **sw3_in** to zero and displays the title of the menu item selected. The StandbyMenu has the contents shown in the table below.

Table 7. Standby Menu Contents

Index (n)	StandbyMenu[n].title	StandbyMenu[n].func	Function Operation
0	" HALT 1 "	StandbyHalt1()	Halt, periodic interrupt every 0.5 seconds
1	" HALT 2"	StandbyHalt2()	Halt, no periodic interrupt
2	" STOP 1 "	StandbyStop1()	Stop, periodic interrupt every 0.5 seconds
3	" STOP 2 "	StandbyStop2()	Stop, no periodic interrupt
4	" STOP 3 "	StandbyStop3()	Stop, subclock oscillation stopped

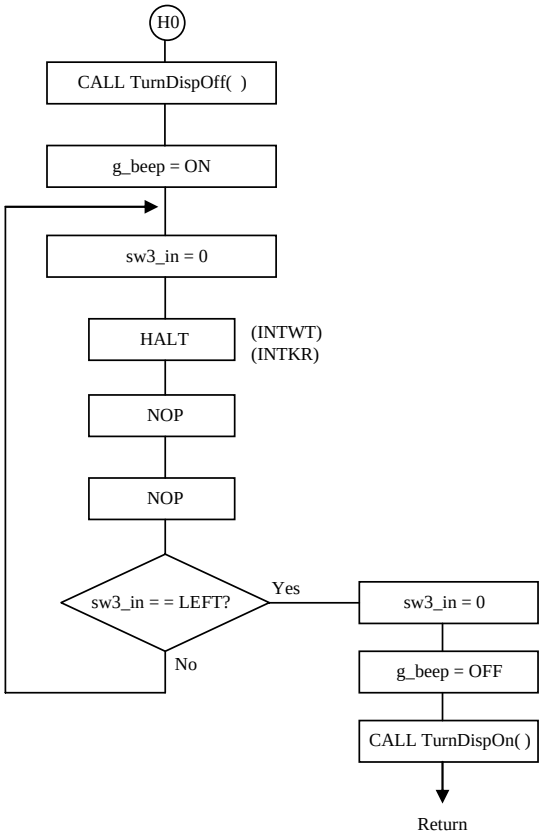
The UP and DOWN keys move through the menu options, rolling over from 4→0 or 0→4.

The RIGHT key executes the selected standby routine. As shown in the flowcharts that follow, these routines turn off the display, set up the mode, and then execute the halt or stop instruction. Since interrupts can bring the processor out of halt or stop modes, the program loops, reentering the halt or stop mode until you press the LEFT key. Pressing the LEFT key in a StandbyXxxxN() routine turns the display back on, and returns to the Standby() routine.

The LEFT key exits from the Standby() routine back to main().

2.3.16 StandbyHalt1( ) – HALT With Periodic Wake-up Interrupt

Figure 23. Flowchart for Halt with Periodic Wake-up Interrupts



The StandbyHalt1() routine executes a halt instruction with a periodic interrupt from the watch timer, INTWT, which occurs every 0.5 seconds.

StandbyHalt1() turns the display off by calling TurnDispOff() and sets the global variable **g_beep** to ON. This variable setting causes the interrupt-service routine MD_INTWT() to sound a brief beep on the buzzer/speaker every 4 seconds, indicating that the processor is waking from the halt mode.

After setting up, StandbyHalt1() sets the **sw3_in** variable to zero to clear the switch input, and executes the halt instruction. The CPU stops instruction execution, which decreases power consumption, but the subclock is still running. If the CPU clock is not the subclock, the main clock is also still running.

StandbyHalt1() follows the halt instruction with a few NOPs. If you run the program with an emulator, you can set a breakpoint on one of the NOPs to catch the exit from halt mode.

The halt mode is ended by either a key-return interrupt (INTKR), if a switch is pressed, or by the watch-Timer interrupt (INTWT), every 0.5 seconds. The appropriate interrupt-service routine executes, and the

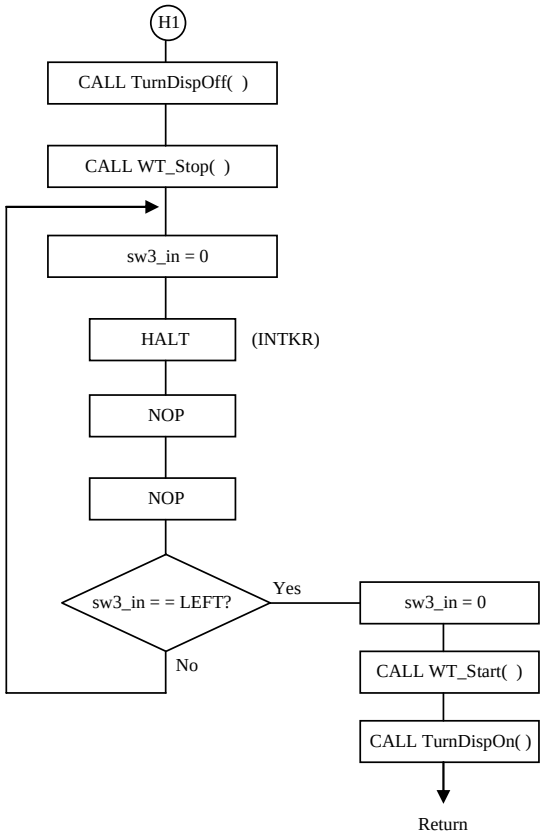
program returns to the halt loop. The routine checks to see if you pressed the LEFT switch. If not, the routine returns to the top of the loop to clear any switch input and execute the halt instruction again.

If you press the LEFT key, StandbyHalt1() clears **sw3_in**, sets **g_beep** to OFF to cancel beeping by the MD_INTWT() routine, turns the display back on with TurnDIspOn(), and returns to the Standby() routine.

Running StandbyHalt1() shows the power consumption needed by the system in halt mode with brief exits every 0.5 second to check status and possibly take some action. The power consumption without a periodic interrupt is slightly lower, as demonstrated by the next routine.

2.3.17 StandbyHalt2() – HALT with No Periodic Interrupt

Figure 24. Flowchart for Halt with no Periodic Interrupt



The StandbyHalt2() routine executes a halt instruction with no periodic interrupts, staying in halt mode until you press a switch.

StandbyHalt2() turns the display off by calling the TurnDispOff() routine, and calls WT_Stop() to stop the watch timer and prevent the periodic INTWT interrupt.

After setting up, StandbyHalt2() sets the **sw3_in** variable to zero to clear switch input, and executes the halt instruction. The CPU stops instruction execution, which decreases power consumption. The subclock is still running, however, and if the CPU clock is not the subclock, the main clock is also still running.

StandbyHalt2() follows the halt instruction with a few NOPs. If you are using an emulator, you can set a breakpoint on one of the NOPs to catch the exit from halt mode. The CPU should stay in halt mode until you press a key.

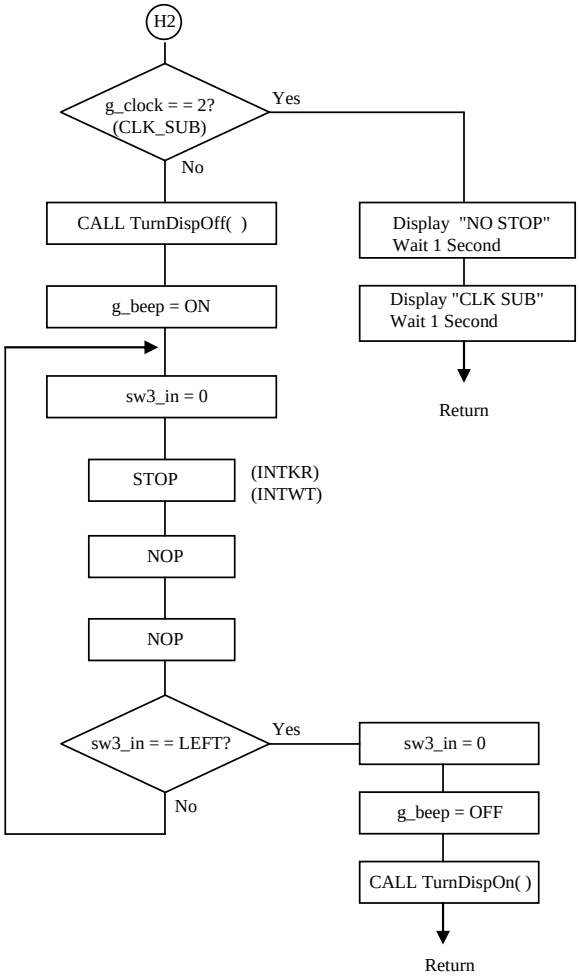
When you press a key, the key-return interrupt (INTKR) cancels the halt mode. The MD_INTKR interrupt-service routine executes to process the switch input, and returns to the halt loop. The routine checks to see if you have pressed the LEFT switch. If not, it returns to the top of the loop to clear any switch input and execute the halt instruction again.

If you press the LEFT key, StandbyHalt2() clears the **sw3_in** variable, turns the watch timer back on by calling WT_Start(), turns the display back on with TurnDIspOn(), and returns to the Standby() routine.

StandbyHalt2() illustrates power consumption when the CPU stays in the halt mode. The power consumption here is slightly less than halt mode with a periodic interrupt.

2.3.18 StandbyStop1( ) – Stop with Periodic Wake-up Interrupt

Figure 25. Flowchart for Stop with Periodic Interrupt



The StandbyStop1() routine executes a stop instruction with a periodic interrupt from the watch timer, which occurs every 0.5 seconds. This routine is similar to StandbyHalt1() but uses stop instead of halt.

You cannot enter stop mode with the CPU running from the subclock. Thus, StandbyStop1() checks **g_clock**, and if the variable is set to CLK_SUB, displays the message “NO STOP”, “CPU SUB”, and returns to the Standby() routine.

StandbyStop1() turns the display off by calling the TurnDispOff() routine, and sets the global variable **g_beep** to ON. This setting causes the interrupt-service routine for the watch-timer interrupt, MD_INTWT(), to sound a brief beep on the buzzer/speaker every 4 seconds, indicating that the processor is waking up from the stop mode.

StandbyStop1() sets the **sw3_in** variable to zero to clear the switch input and executes the stop instruction. This action stops the CPU from executing instructions and turns off the main clock. The subclock is still running.

StandbyStop1() follows the stop instruction with a few NOPs. If you run the program with an emulator, you can set a breakpoint on one of the NOPs to catch the exit from stop mode.

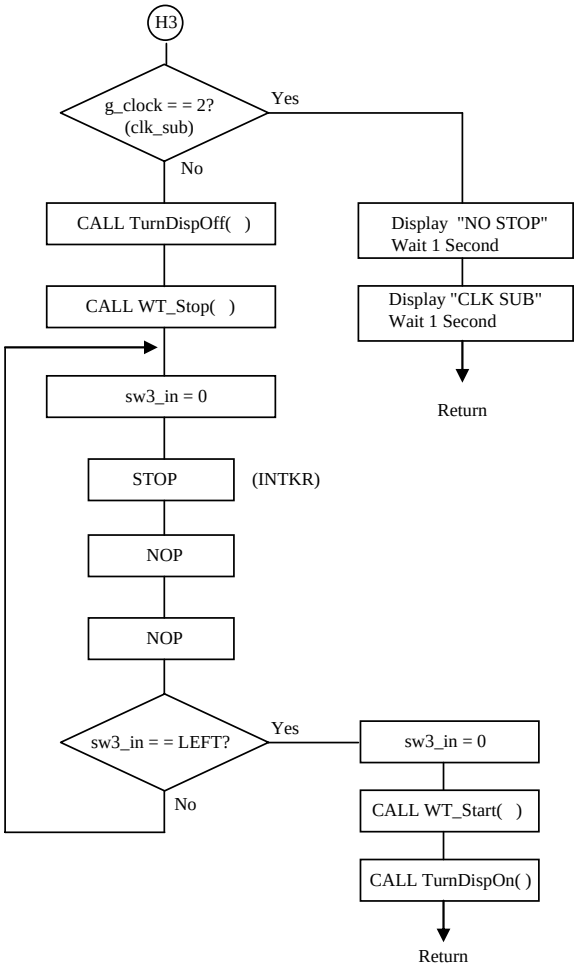
Either a key-return or watch-timer interrupt cancels the stop mode and restarts the main clock. Once the stop mode is cancelled, the appropriate interrupt-service routine executes and returns to the stop loop. The routine checks to see if you pressed the LEFT switch. If not, it returns to the top of the loop to clear any switch input and execute the stop instruction again.

If you press the LEFT key, StandbyStop1() clears the **sw3_in** variable, sets **g_beep** to OFF to cancel beeping by the MD_INTWT() routine, turns the display back on with TurnDIspOn(), and returns to the Standby() routine.

Running StandbyStop1() shows the power consumption needed by the microcontroller in stop mode with brief exits every 0.5 second to check status and possibly take some action. The power consumption in this case is slightly less than when running with the CPU set to the subclock.

2.3.19 StandbyStop2( ) – Stop with No Periodic Interrupt, Subclock Running

Figure 26. Flowchart for Stop with no Periodic Interrupt



The StandbyStop2() routine executes a stop instruction with no periodic interrupts, staying in stop mode until you press a switch. This routine is similar to StandbyHalt2() but uses stop instead of halt.

You cannot enter stop mode with the CPU running on the subclock. Thus, StandbyStop2() checks **g_clock**. If the variable is set to CLK_SUB, the routine displays the message “NO STOP”, “CPU SUB”, and returns to the Standby() routine.

StandbyStop2() turns the display off by calling the TurnDispOff() routine, and calls WT_Stop() to stop the watch timer to prevent the periodic INTWT interrupt.

StandbyStop2() sets **sw3_in** to zero to clear switch input and executes the stop. The CPU stops executing instructions, and the main clock stops. The subclock is still running.

StandbyStop2() follows the stop instruction with a few NOPs. If you run the program with an emulator, you can set a breakpoint on one of the NOPs to catch the exit from stop mode. The CPU stays in stop mode until you press a key.

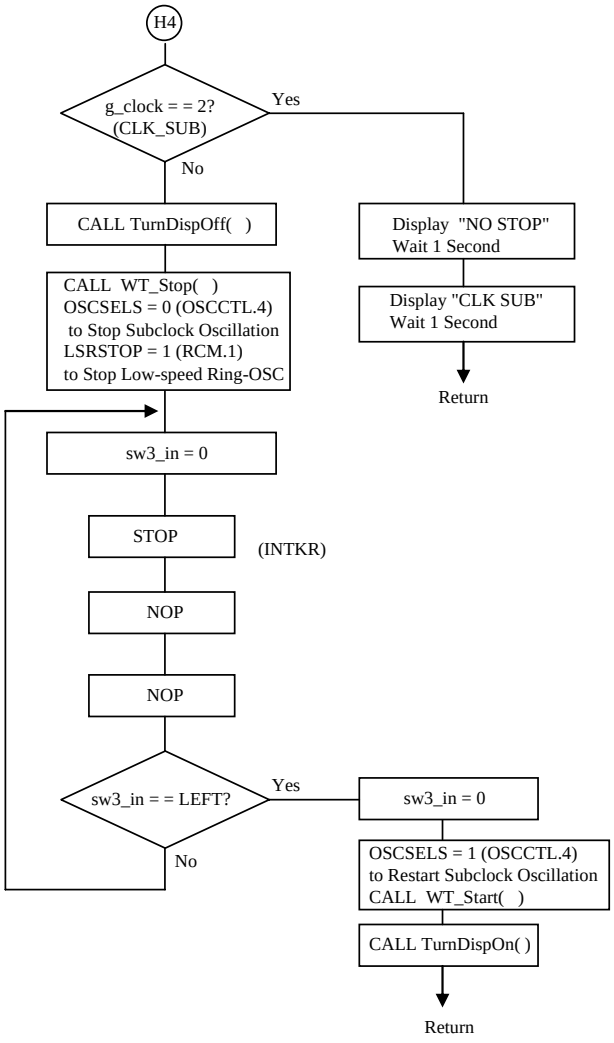
When you press a key, the key-return interrupt (INTKR) cancels the stop mode and restarts the main clock. The MD_INTKR interrupt-service routine processes the switch input and returns to the stop loop. The routine checks to see if you pressed the LEFT switch. If not, it returns to the top of the loop to clear any switch input and execute the stop instruction again.

If you press the LEFT key, StandbyStop2() clears the **sw3_in** variable, turns on the watch timer by calling WT_Start(), turns the display back on with TurnDIspOn(), and returns to the Standby() routine.

Running StandbyStop2() shows power consumption when the CPU stays in stop mode. The power consumption here is slightly less than stop mode with a periodic interrupt, and also slightly less than when running with the CPU set to the subclock.

2.3.20 StandbyStop3( ) – Stop with No Periodic Interrupt, Subclock Stopped

Figure 27. Flowchart for Stop with No Periodic Interrupt, Subclock Stopped



The StandbyStop3() routine executes a stop instruction with no periodic interrupts, staying in stop mode until you press a switch. The routine also stops the subclock to conserve the power used to drive the crystal oscillator.

You cannot enter stop mode with the CPU running from the subclock. Thus, StandbyStop3() checks **g_clock**. If the variable is set to CLK_SUB, the routine displays the message “NO STOP”, “CPU SUB”, and returns to the Standby() routine.

StandbyStop3() turns the display off by calling the TurnDispOff() routine, and calls WT_Stop() to stop the watch timer and prevent the periodic INTWT interrupt. StandbyStop3() then sets the OSCSELS bit in the OSCCTL register to zero, which sets the P123/XT1 and P124/XT2 pins in I/O port mode. These pins

connect to the 32.768-kHz subclock crystal, so changing their mode stops the crystal. The routine also sets the LSRSTOP bit in the RCM register to 1 to stop the internal low-speed oscillator. Now only the main clock is oscillating.

After setting up, StandbyStop3() sets the **sw3_in** variable to zero to clear the switch input, and executes the stop instruction. This action prevents the CPU from executing instructions and stops the main clock. At this point, no clocks are active.

StandbyStop3() follows the stop instruction with a few NOPs. If you run the program with an emulator, you can set a breakpoint on one of the NOPs to catch the exit from stop mode. The CPU stays in stop mode until you press a key.

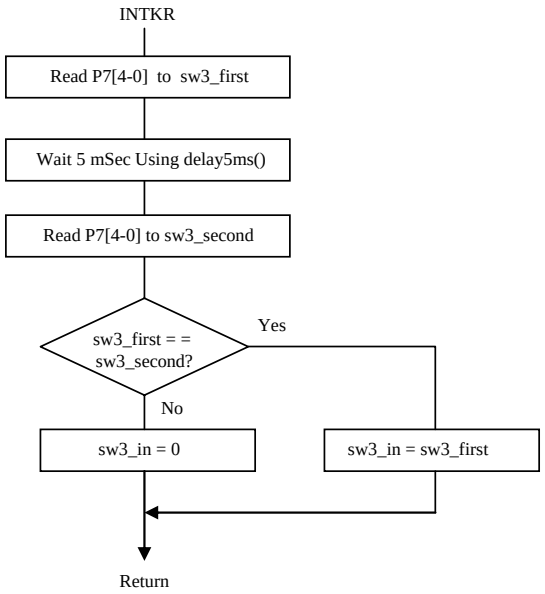
Pressing a key invokes the key-return interrupt (INTKR), which cancels the stop mode and restarts the main clock. The MD_INTKR interrupt-service routine processes the switch input and returns to the stop loop. The routine checks to see if you pressed the LEFT switch. If not, the routine returns to the top of the loop to clear any switch input and execute the stop instruction again.

If you press the LEFT key, StandbyStop3() clears the **sw3_in** variable, turns the watch timer back on by calling WT_Start(), turns the display back on with TurnDIspOn(), and returns to the Standby() routine.

Running StandbyStop3() shows power consumption when the CPU stays in stop mode, with all clocks stopped. This level of power consumption is slightly lower than that for stop mode with the subclock and internal low-speed oscillator running, and also slightly lower than when running the CPU on the subclock.

2.3.21 MD_INTKR( ) – Key-Return Interrupt-Service Routine

Figure 28. Flowchart for Key-Return Interrupt-Service Routine



The MD_INTKR() routine services the INTKR interrupt. Closing any of the key switches on the SW3 navigation switch connects the key-return input pin to GND. The negative edge on the key return pin causes the INTKR interrupt.

The MD_INTKR() routine reads the input port attached to the switches to get the current state of the pins. The routine then calls delay5msec() to delay for five milliseconds. After this 5-millisecond interval, the routine reads the input port again.

If the first and second values for the switches agree, the routine considers the switches stable. MD_INTKR() sets the global variable **SW3_in** to the value **read** (to indicate that a switch has been pressed) and returns.

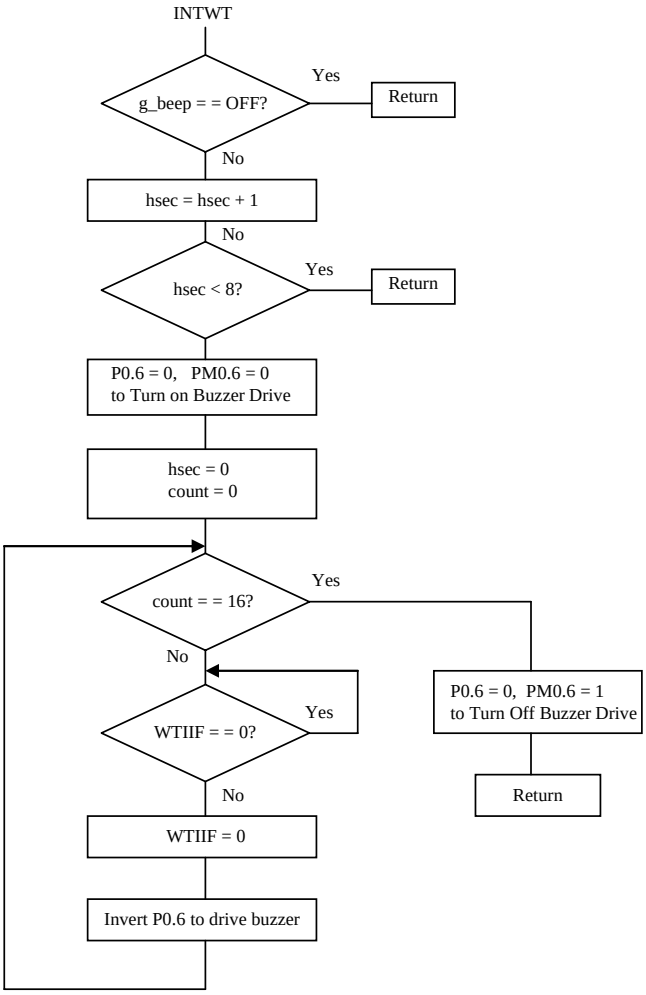
If the first and second values do not agree, MD_INTKR() sets **SW3_in** to zero (to indicate that the routine did not find a stable switch value) and returns.

If the switches are still bouncing, the next negative edge on the key return inputs causes another INTKR interrupt, and MD_INTKR() checks the switches again.

Note that the delay5msec() routine (not included in the flowchart) uses the watch-timer interval timer. This timer's interval is approximately two milliseconds, so the routine delays by counting three intervals. In StandbyStop3(), however, the subclock is not running, so delay5msec() executes an instruction loop to approximate the delay. The program adjusts the number of instructions executed in the loop based on the value in the PCC register, thus creating the same delay regardless of the PCC divider of the main system clock .

2.3.22 MD_INTWT( ) – Watch-Timer Interrupt-Service Routine

Figure 29. Flowchart for Watch-Timer Interrupt-Service Routine



Watch-timer interrupt INTWT invokes the MD_INTWT() routine. The interrupt occurs every 0.5 seconds when the subclock is running.

The MD_INTWT() routine checks the global variable **g_beep**, and if the variable is off, just returns from interrupt service.

In StandbyHalt1() or StandbyStop1(), **g_beep** is on, so MD_INTWT() increments the variable **hsec**. This variable tracks half-seconds. If the half-second counter has not reached eight (four seconds), the routine returns.

If **hsec** has reached eight, MD_INTWT() clears the variable to zero and generates a short beep. The routine generates the beep by setting pin P06 as an output, then setting the pin's value alternately low and high

every time the watch-timer interval flag is set. The watch-timer interval counter overflows every 1953 microseconds, or about every 2 milliseconds. Using this interval thus generates a square wave with a 4-millisecond cycle time—a 250-Hz frequency. The routine runs this on/off cycle 16 times, so the beep lasts about 32 milliseconds.

With P06 connected to a buzzer, the beep of the square wave tells you when the INTWT interrupt brings the CPU out of halt or stop mode periodically for processing.

2.4 Applilet's Reference Driver

NEC Electronics' Applilet program generator can automatically generate C or assembly language source code to manage peripherals for NEC Electronics microcontrollers. Please see the Appendix for the version of the Applilet used.

The Applilet produces the code for the program's basic initialization and main function; clock initialization; drivers for the key-return interrupt; initialization and control for the watch timer; and code to handle the IIC0 controller for communication with the LCD controller. After the Applilet produces the basic code, you can add code to customize the program.

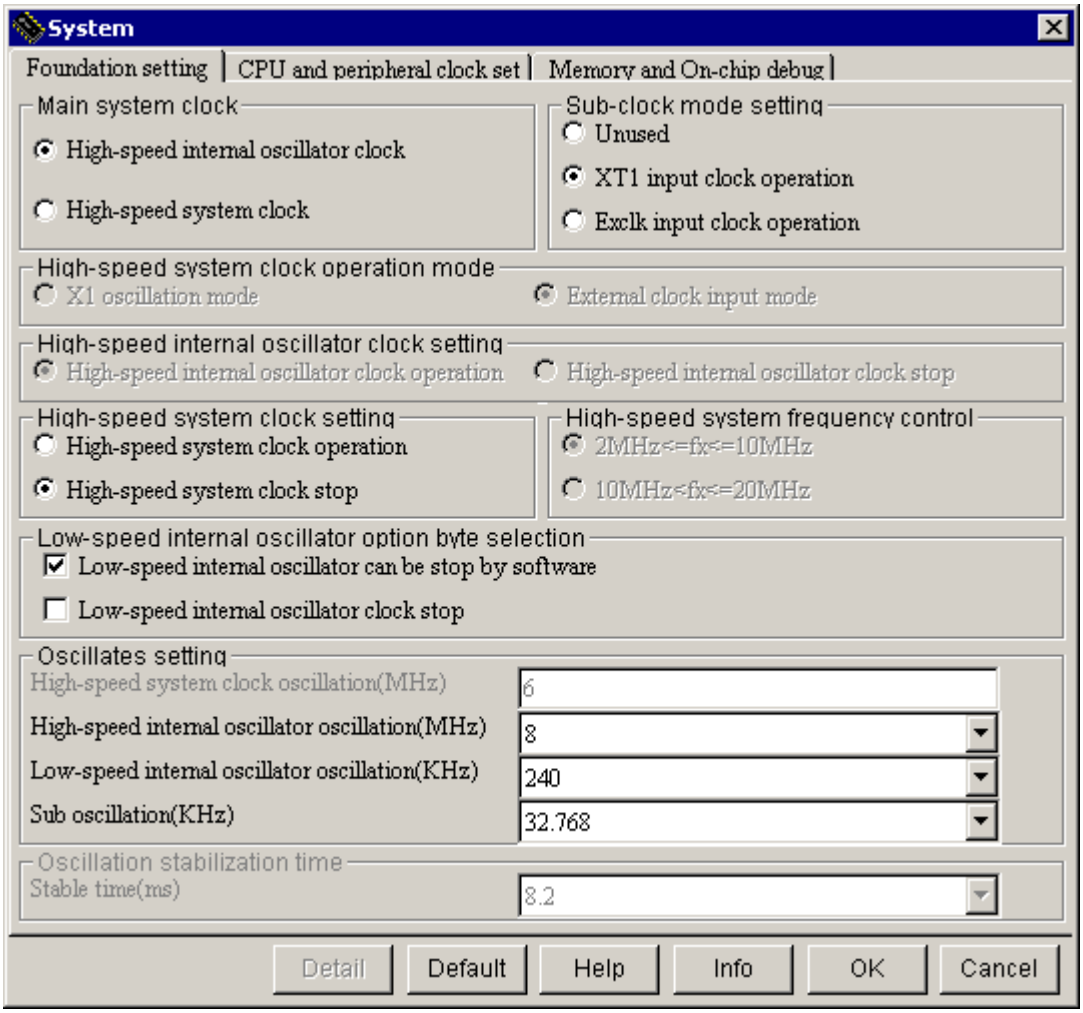
This section describes how to set up the Applilet to produce code for these peripherals, and lists the files and routines produced. Also listed are additional files not generated by the Applilet.

When you start the Applilet and select the target device, save your settings to a new project (.prx) file. The Applilet displays a dialog box that lets you select the peripheral blocks you want to set up.

2.4.1 Configuring Applilet for Clock Initialization

Select **System** and the Applilet displays the system settings for clocks. Use the **Foundation setting** tab to establish which clocks operate at startup.

Figure 30. Configuring Applilet for Clock Initialization



This dialog controls how the Clock_Init() routine initializes the clocks. When the processor starts, the CPU is running on the internal high-speed oscillator, and the external clock input is disabled. For the demonstration, you want the microcontroller to remain in that mode, so keep **High-speed internal oscillator clock** (internal high-speed oscillator) selected for the **Main system clock**. **High-speed system clock setting** is set for **High-speed system clock stop**, which ensures that the external clock remains stopped.

Because the external clock is disabled, the **High-speed system clock operation mode** section is grayed out. The choices in this section would select between a crystal on P121/X1 and P122/XT2, or a driven input on P122/EXCLK.

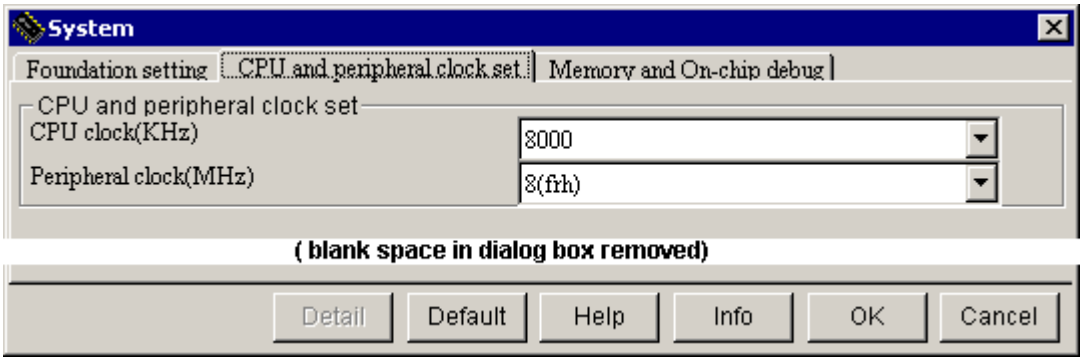
Since the high-speed internal oscillator has already been selected, the choices in the **High-speed internal oscillator clock setting** section are also grayed out.

Under **Sub-clock mode setting**, click **XT1 input clock operation** to select a crystal oscillator connected to the XT1 and XT2 pins. (Selecting **Unused** would disable the subclock, and **Exclk input clock operation** would configure the microcontroller for a driven input on the P124/XT2 pin.)

Set the **Low-speed internal oscillator option byte selection** to allow the internal low-speed oscillator to be stopped by software. This action is controlled by a bit in the option byte. You can set the option byte to prevent the internal low-speed oscillator from being stopped, which is useful when the oscillator runs the watchdog timer.

Clock speeds depend on the microcontroller specification—in this case, 8 MHz for the internal high-speed oscillator, 240 kHz for the internal low-speed oscillator, and 32.768 kHz for the subclock. The setting for the external clock is grayed out since this clock is disabled at the start.

Figure 31. CPU and Peripheral Clock Information



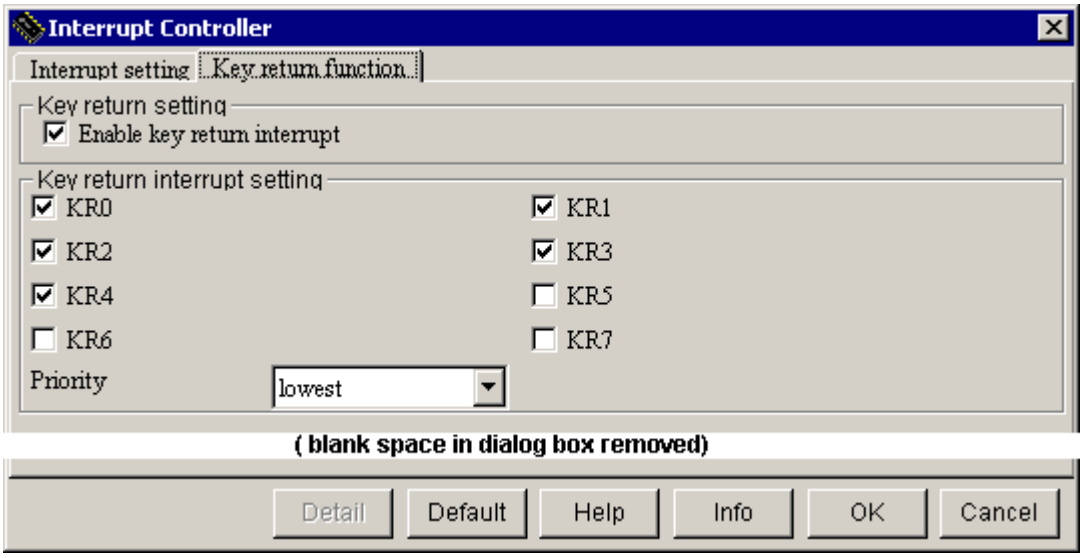
Once you have configured the foundation settings, you can move to the **CPU and peripheral clock set** settings.

The **CPU clock(kHz)** setting controls the initial main-clock division (set in the PCC register). Choose 8000 (for 8 MHz), for the fastest CPU speed. With the CPU clock sourced by the internal high-speed oscillator, the peripheral clock must be the same, so the only choice for **Peripheral clock(MHz)** is 8 MHz.

2.4.2 Configuring Applilet for Key-Return Interrupt

Select **Interrupt Controller** and then the **Key return function** tab to set details for the key-return interrupt.

Figure 32. Configuring Key-Return Interrupt



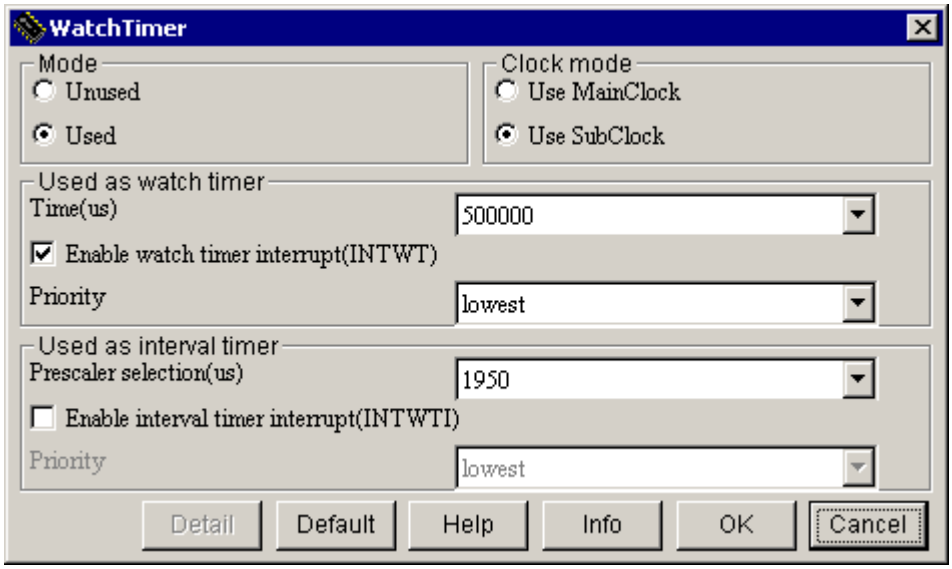
Check the **Key return setting** box to **Enable key-return interrupt**, and set the interrupt priority for lowest.

The navigation switch connects to pins P70/KR0 through P74/KR4, so check the boxes for **KR0** through **KR4**. You can use pins P75/KR5, P76/KR6 and P77/KR7 as general-purpose I/O pins without affecting the key-return interrupt function.

2.4.3 Configuring Applilet for Watch Timer

Selecting **WatchTimer** brings up a dialog showing the watch-timer settings.

Figure 33. Configuring Watch Timer



Set **Mode** to **Used** and set **Clock mode** to **Use SubClock** to establish the 32.768-kHz subclock as the watch-timer timebase.

Under **Used as watch timer** select 500000 (0.5 second) from the drop-down list, and check the box to enable the INTWT interrupt.

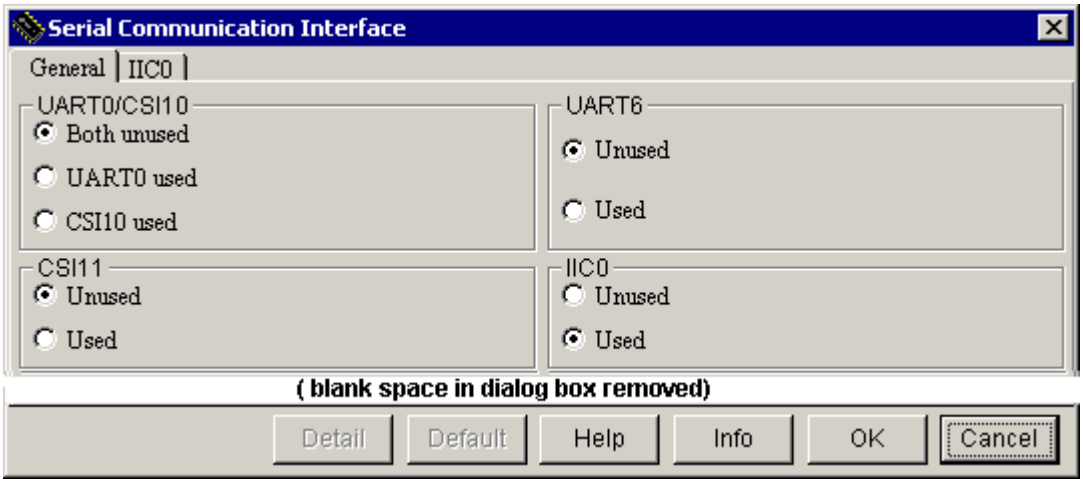
Under **Used as interval timer**, SELECT 1950 (microseconds) as the interval. Leave **Enable interval timer interrupt(INTWTI)** unchecked.

The Applilet generates WT_Init() to initialize the watch timer and WT_Start() to start the timer. The Applilet can also generate other functions for watch timer control, and you can select these additional functions either in the Applilet function view or during code generation.

2.4.4 Configuring Applilet for IIC0 Communication

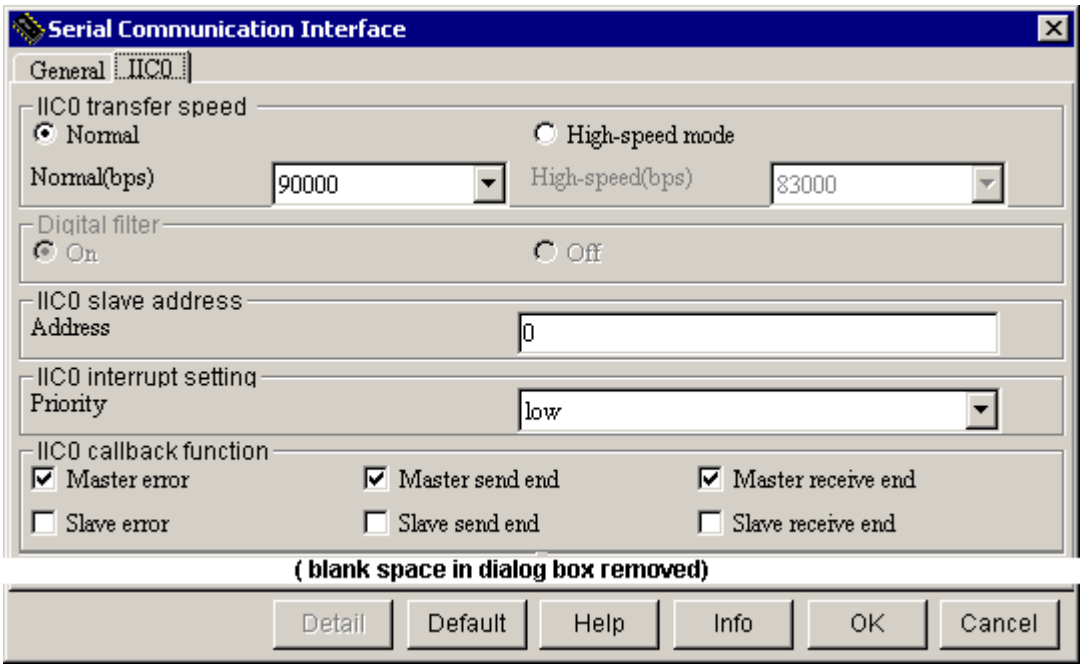
The microcontroller's IIC0 peripheral is used internally for communication with the LCD controller, so click **Serial** and select the IIC0 peripheral as **Used**.

Figure 34. Configuring Applilet for IIC0 Communication



You can now go to the **IIC0** tab to make detailed settings.

Figure 35. IIC0 Detail Settings



Set the IIC0 peripheral to operate in **Normal** mode. This selection makes the drop-down menu of communication speeds available. The speeds are based on divisions of the peripheral clock, so the choices depend on the peripheral clock. Select the highest available speed—90000 bps.

The IIC0 peripheral can operate in slave or master mode, and can switch between the two. Unless in the middle of a transfer initiated as a master, the peripheral runs in slave mode, but this demonstration does not use slave mode. Leave the slave address at the default of 0.

Set the **IIC0 interrupt setting** for low priority.

The Applilet generates several standard operating routines and can generate blank callback routines that allow you to insert code for dealing with particular events. Some of these callback routines support master communication, so check **Master error**, **Master send end** and **Master receive end**. Leave the slave-callback boxes unchecked.

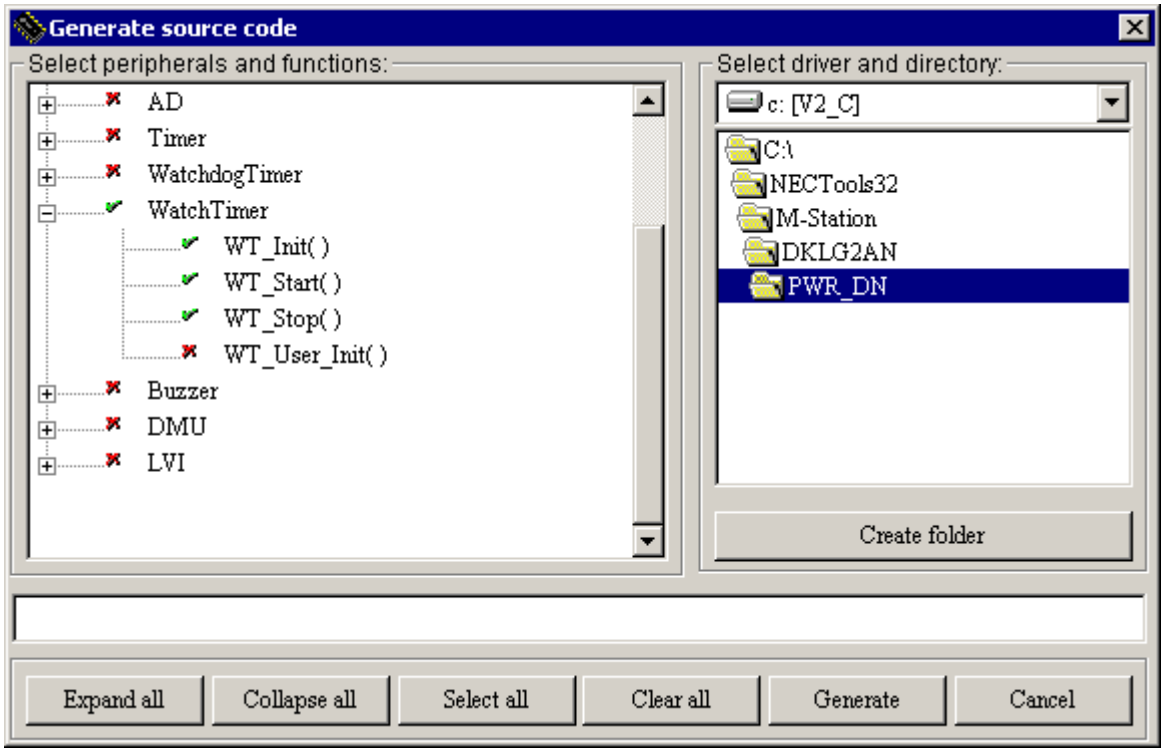
For the IIC0 peripheral, the Applilet also has optional functions not selected in the IIC0 tab. You can select these functions in the Applilet function view or during code generation.

2.4.5 Generating Code with Applilet

Once you have set up the various dialog boxes, click **Generate code**. The Applilet displays the peripherals and functions to be generated and allows you to select a directory in which to store the source code.

At this point, select the additional watch-timer function, WT_Stop(), as shown below.

Figure 36. Selecting Watch-Timer Function



Several additional functions are selected for the IIC0 peripheral as well. To make these choices, expand **Serial**, then expand **IIC0**. Point to the appropriate function and left-click to select.

When you click **Generate**, the Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and shows the list of files created in a dialog box.

To support clock initialization, the Applilet generates system.h and system.c.

To support the key-return interrupt, the Applilet generates int.h, int.c, and int_user.c.

To support the watch timer, the Applilet generates watchtimer.h, watchtimer.c and watchtimer_user.c.

To support the IIC0 peripheral, the Applilet generates serial.h, serial.c, and serial_user.c.

The Applilet generates several other files, including a main.c file with a blank main function.

2.4.6 Applilet-Generated Files and Functions for Clock Initialization

The files system.h and system.c contain the code for clock initialization.

2.4.6.1 System.h

The header file system.h contains declarations for the functions controlling the clock and definitions of values for clock initialization.

2.4.6.2 System.c

The source file system.c contains the following function generated by the Applilet for clock initialization:

void Clock_Init(void)
The Clock_Init() routine is the first function SystemInit() calls. Clock_Init() sets up the clocks as specified in the Applilet's **System** block.

2.4.7 Applilet-Generated Files and Functions for Key-Return Interrupt

The files int.h, int.c and int_user.c contain the code generated for key-return interrupt support.

2.4.7.1 Int.h

The header file int.h contains declarations for the functions controlling the key-return interrupt and definitions of values for key-return initialization. The header file macrodriver.h, used for all

Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

Code added to the Applilet-generated file provides the external declaration of the **sw3_in** variable. This variable reports the state of the key switch matrix, so including int.h enables other portions of the program to examine the switch state. You must also add the definitions of the bit patterns set in the **sw3_in** variable corresponding to switch 1 down, switch 2 down, etc.

2.4.7.2 Int.c

The source file Int.c contains the following function generated by the Applilet for the key-return interrupt:

void INT_Init(void)
The INT_Init() routine initializes the key-return register and interrupt as specified in the Applilet key-return dialog.

2.4.7.3 Int_user.c

The source file int_user.c contains stub functions for user code. These functions are empty on code generation to allow you to add application-specific code.

__interrupt void MD_INTKR(void)
This is the interrupt-service routine for the key-return interrupt INTKR, generated when a negative-going edge is seen on one of the key-return pins.

The Applilet generates this interrupt-service routine blank. You add code to use the key-return interrupt to debounce and read the key switch matrix.

You must add the declaration of the **sw3_in** variable to this file.

2.4.8 Applilet-Generated Files and Functions for Watch Timer

The code generated for watch-timer support is in the files watchtimer.h, watchtimer.c and watchtimer_user.c.

2.4.8.1 Watchtimer.h

The header file watchtimer.h contains declarations for the functions controlling the watch timer. The header file macrodriver.h, used for all Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some watch-timer functions

2.4.8.2 Watchtimer.c

The source file watchtimer.c contains the following functions generated by the Applilet for the watch timer:

void WT_Init(void)
The WT_Init() routine initializes the watch timer as specified in the Applilet watch-timer detail dialog.

void WT_Start(void)
The WT_Start() routine starts watch-timer operation by enabling the timer and the interrupt INTWT.

void WT_Stop(void)
The WT_Stop() routine stops the watch timer by disabling the timer and disabling the timer interrupts.

2.4.8.3 Watchtimer_user.c

The source file watchtimer_user.c contains stub functions for user code. These functions are empty on code generation to allow you to add application-specific code.

__interrupt void MD_INTWT(void)
This is the interrupt-service routine for the watch-timer interrupt INTWT. The Applilet generates this interrupt-service routine blank. You add code to use the watch timer to count half-seconds and to beep every four seconds.

2.4.9 Applilet-Generated Files and Functions for IIC0 Communication

The code generated for IIC0 support is in the files serial.h, serial.c and serial_user.c.

2.4.9.1 Serial.h

The header file serial.h contains declarations for the functions controlling IIC0, and definitions of values for IIC0 initialization. The header file macrodriver.h, used for all Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

You need to add external declarations for the variables used to signal IIC0 states:

- extern MD_STATUS UI_MasterError

- extern MD_STATUS UI_MasterSendEnd
- extern MD_STATUS UI_MasterReceiveEnd
- extern MD_STATUS UI_MasterFindSlave

Also add external declarations for the functions IIC0_Init( ) and IIC0_MasterStartAndSend( ).

2.4.9.2 Serial.c

The source file Serial.c contains the following Applilet-generated functions for IIC0:

void IIC0_Init(void)
This routine initializes the IIC0 peripheral as specified in the Applilet IIC0 detail dialog.

MD_STATUS IIC0_MasterStart(TransferMode mode, UCHAR adr, UCHAR wait)
This routine starts master-mode communications. The **mode** parameter is either Send or Receive to specify the direction. The **adr** parameter specifies the IIC slave address, and **wait** specifies the number of loop times to wait for a start condition to occur. This routine creates a start condition and sends the slave address.

void IIC0_Stop(void)
This routine stops the IIC0 peripheral. The demonstration program does not use this function.

MD_STATUS IIC0_MasterSendData(UCHAR* txbuf , UINT txnum)
IIC0_MasterStart( ) calls this routine after starting a Send transfer. The **txbuf** parameter points to an array of bytes to send, and the **txnum** parameter specifies the number of bytes to send. This routine sets the buffer pointer and count, and sends the first byte in the buffer. The MD_INTIIC0( ) interrupt-service routine sends the remaining bytes.

MD_STATUS IIC0_MasterReceiveData(UCHAR* rxbuf, UINT rxnum)
IIC0_MasterStart( ) calls this routine after starting a Receive transfer. The **rxbuf** parameter specifies the address of a buffer for storing received characters, and **rxnum** specifies the number of characters to receive. The demonstration program does not use this routine.

__interrupt void MD_INTIIC0(void)
This is the IIC0 interrupt-service routine, invoked by the INTIIC0 interrupt. This routine calls either IIC0_MasterHandler( ) or IIC0_SlaveHandler( ), depending on the IIC0 master or slave state.

MD_STATUS IIC0_SlaveHandler(void)
MD_INTIIC0( ) calls this routine to handle IIC0 interrupts in master mode.

MD_STATUS IIC0_MasterHandler(void)
MD_INTIIC0( ) calls this routine to handle IIC0 interrupts in slave mode.

2.4.9.3 Serial_user.c

The source file serial_user.c contains stub functions for user code. These functions are empty on code generation to allow you to add application-specific code.

You need to add the following variables to enable subroutines to check the state of IIC0 communication:

- MD_STATUS UI_MasterError (stores master errors)
- MD_STATUS UI_MasterSendEnd
- MD_STATUS UI_MasterReceiveEnd
- MD_STATUS UI_MasterFindSlave

void IIC0_User_Init(void)
IIC0_Init( ) calls this routine, but the demonstration does not use the routine.

void CALL_IIC0_MasterError(MD_STATUS flag)
After detecting an error, IIC0_MasterHandler( ) calls this routine. Add code to store the **flag** parameter in **UI_MasterError**.

void CALL_IIC0_MasterReceiveEnd(void)
IIC0_MasterHandler( ) calls this routine when the receive count reaches zero. Add code to set the **UI_MasterReceiveEnd** flag to MD_OK. The demonstration program does not use this routine.

void CALL_IIC0_MasterSendEnd(void)
IIC0_MasterHandler( ) calls this routine after sending the last transmit byte. Add code to set the **UI_MasterSendEnd** flag to MD_OK.

void CALL_IIC0_SlaveAddressMatch(void)
IIC0_SlaveHandler( ) calls this routine when the slave address received in slave mode matches the slave address for the IIC0 peripheral. The demonstration program does not use this routine.

void CALL_IIC0_MasterFindSlave(void)

This routine sets the **UI_MasterFindSlave** flag to MD_OK. The routine is called when IIC0_MasterHandler( ) receives an ACK after sending the slave address, indicating that a slave has responded to the slave address.

The following routine was not generated by the Applilet, but was written for this demonstration program:

MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum)
This routine combines the MasterStart and MasterSendData functions, since these work together to send data to a particular slave. First, IIC0_MasterStart(Send, sadr, 10) sets up a sending transfer to the specified slave address. The routine then waits for **UI_MasterFindSlave** to be set, indicating that the slave has responded.

Then IIC0_MasterSendData(txbuf, txnum) sends the specified data. The routine then waits for the **UI_MasterSendDone** flag to be set, indicating that the transfer has completed.

The routine monitors the **UI_MasterError** flag while waiting, to check for error conditions in the sending of the slave address or data transfer.

2.4.10 Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files. The files and their functions are shown below.

Table 8. Other Applilet-Generated Files

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

2.4.11 Demonstration-Program Files Not Generated by Applilet

The demonstration program also includes the following files, not generated by the Applilet.

Table 9. Demonstration-Program Files not Generated by Applilet

File	Function
define.h	Definitions of navigation switch values
lcd.h	Header file for high-level LCD functions
lcd.c	Code to write characters and strings to the LCD
LcdDrvApp.h	Header file for LCD-driver functions
LcdDrvApp.c	Code to initialize, write and read the LCD controller; these functions call Applilet-generated IIC0 routines

2.5 Demonstration Platform

The demonstration uses a development board from NEC Electronics. You may be able to duplicate the same hardware using off-the-shelf components along with the NEC Electronics microcontroller of interest.

2.5.1 Resources

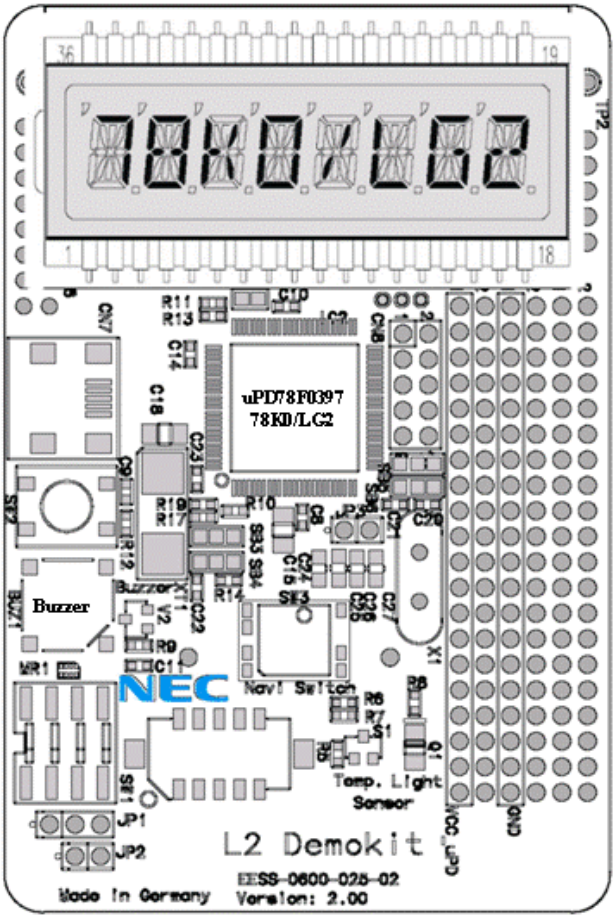
The program demonstration uses the following resources:

- DemoKit-LG2 demonstration board, with µPD78F0397 8-bit microcontroller mounted
- DemoKit-LG2 resources:
  - 8-character, 14-segment LCD
  - SW3 navigation switch
  - BUZ1 buzzer driven by P06/TO01
  - 3V battery in BAT1 receptacle
  - Ammeter between JP1.3 and JP1.2 to measure current from battery
  - Added 10-MHz crystal in location X1 (normally not populated); remove JP3

In order to make more accurate current measurements, disable the DemoKit-LG2 light sensor by removing resistor R8, and disconnect phototransistor Q1 from the 78F0397 MCU.

For details on the hardware listed above, please refer to the appropriate user manual, available from NEC Electronics upon request.

Figure 37. Demonstration Platform



2.5.2 Demonstration of Program

With the hardware configured and the µPD78F0397 microcontroller programmed with the demonstration code, the demonstration is as follows:

- Press navigation switch SW3 UP and DOWN to change menu selections
- Press navigation switch SW3 RIGHT to select submenu or activate selection
- Press navigation switch SW3 LEFT to return to previous menu
- Observe the current on ammeter in selected clock/standby mode

In submenu CPU CLK:

- Select C HSR to set clock to internal high-speed oscillator (Note 1)
- Select C EX to set clock to 10-MHz external crystal X1 (Note 2)
- Select C SUB to set clock to 32.768-kHz subclock

Note 1: The CPU and peripherals begin operation running on the internal high-speed oscillator (HSR). When you switch the CPU clock to the X1 crystal, you must switch the peripherals as well. Once you set the peripherals to use the external crystal you cannot switch them back to the HSR, even though you can switch the CPU clock back to the HSR. To return the peripherals to the HSR, reset the device or remove and reapply power.

Note 2: The option C EX selects the external X1 crystal. In the initialization process for setting X1, the program checks for crystal oscillation. If the crystal does not start oscillating within a timeout period, the program does not change the clock to C EX, but instead keeps the current clock. The routine displays an error message of "E NO OSC" on the LCD and waits for you to press a switch before continuing.

In submenu SET PCC:

- Select PCC 0 to set fastest CPU clock operation (f_{XP})
- Select PCC 1 to set CPU clock as $f_{XP}/2$
- Select PCC 2 to set CPU clock as $f_{XP}/4$
- Select PCC 3 to set CPU clock as $f_{XP}/8$
- Select PCC 4 to set slowest CPU clock operation ($f_{XP}/16$)

Note: If you set the CPU CLK to C SUB (subclock operation), changing PCC has no effect. In this case, selecting SET PCC displays PCC SUB. The routine then waits for a further SW3 key press before returning to the main menu.

Selecting main menu item DISPLAY shows "OFF" for a second, and then turns off the LCD, the LCD controller, and the IIC0 peripheral. Make your current measurements when the LCD goes blank to see the power savings from turning off the display. Press SW3 to turn the LCD on again. This restart takes four seconds, during which the current consumption includes IIC0 and the LCD controller.

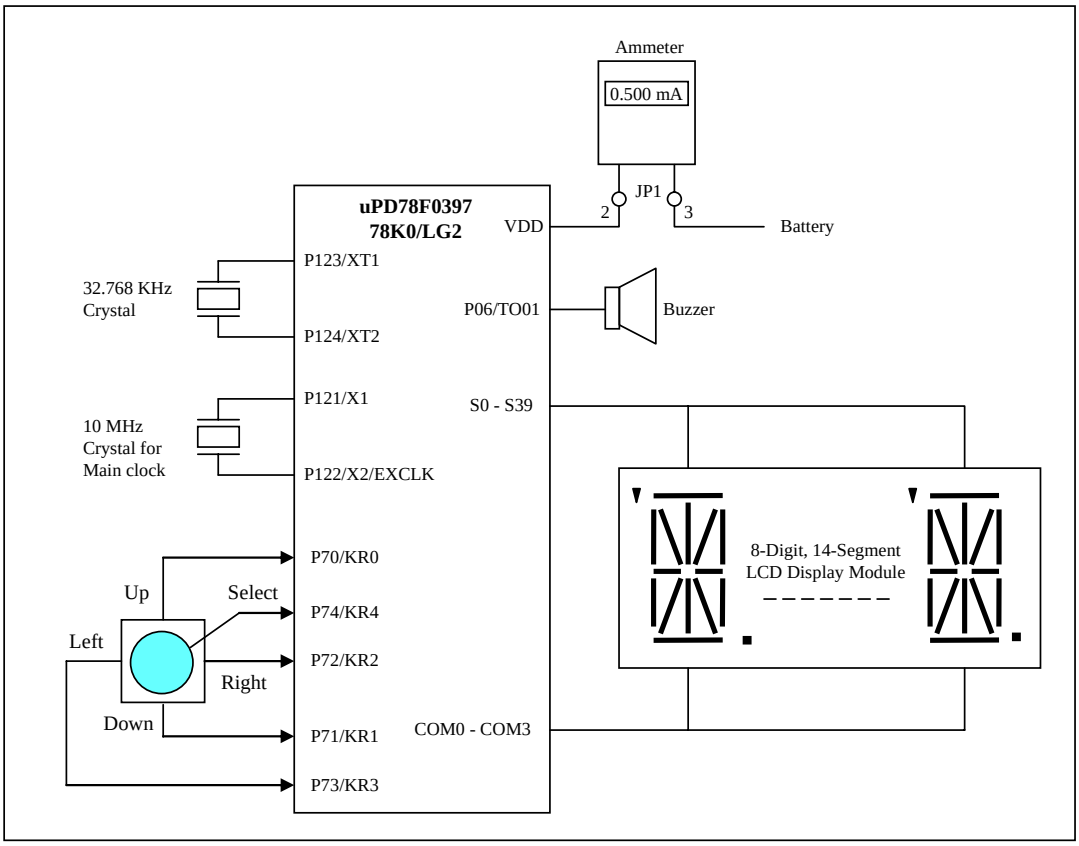
In submenu STANDBY:

- Select HALT 1 for halt mode 1 = halt instruction, subclock running, periodic watch-timer interrupt every 0.5 second, short beep every 4 seconds
- Select HALT 2 for halt mode 2 = halt instruction, subclock running, no periodic interrupts
- Select STOP 1 for stop mode 1 (Note) = stop instruction, subclock running, periodic watch-timer interrupt every 0.5 seconds, short beep every 4 seconds
- Select STOP 2 for stop mode 2 (Note) = stop instruction, subclock running, watch timer stopped
- Select STOP 3 for stop mode 3 (Note) = stop instruction, subclock stopped, watch timer stopped

Note: If you set the CPU CLK to C SUB (subclock operation), stop is not allowed. If you attempt stop mode, you see a message, and the LCD shows the submenu item again.

2.6 Hardware Block Diagram

Figure 38. Hardware Block Diagram



JP1 is the power terminal for the µPD78F0397 MCU on the DemoKit-LG2. Connect an ammeter between JP1.3 and JP1.2 to measure MCU current from the 3V battery in the BAT1 holder underneath the board.

When you attach a crystal in the board's X1 location, remove the JP3 jumper. This jumper connects a driven 6-MHz clock from the PLD to the P122/X2/EXCLK pin, in conflict with the crystal, when the board is powered from the USB cable.

2.6.1 Power Measurement Results

In a standard DemoKit-LG2, the light sensor causes variable current consumption, which interferes with current measurements. Disconnect the light sensor by removing R8 and disconnecting Q1 from the CPU. Additionally, if you operate the board from the 5V supply provided by the USB cable, the USB interface and PLD are powered and can interfere with power measurements. For this demonstration, measure power with the 3V battery (USB and PLD not operating).

Note that the following results are in-system current in various operation modes tested on DemoKit-LG2. For the power consumption of a specific microcontroller, refer to the device's user manual.

Figure 39. In-system current measurement with 8-MHz Internal High-Speed Oscillator and 3V power

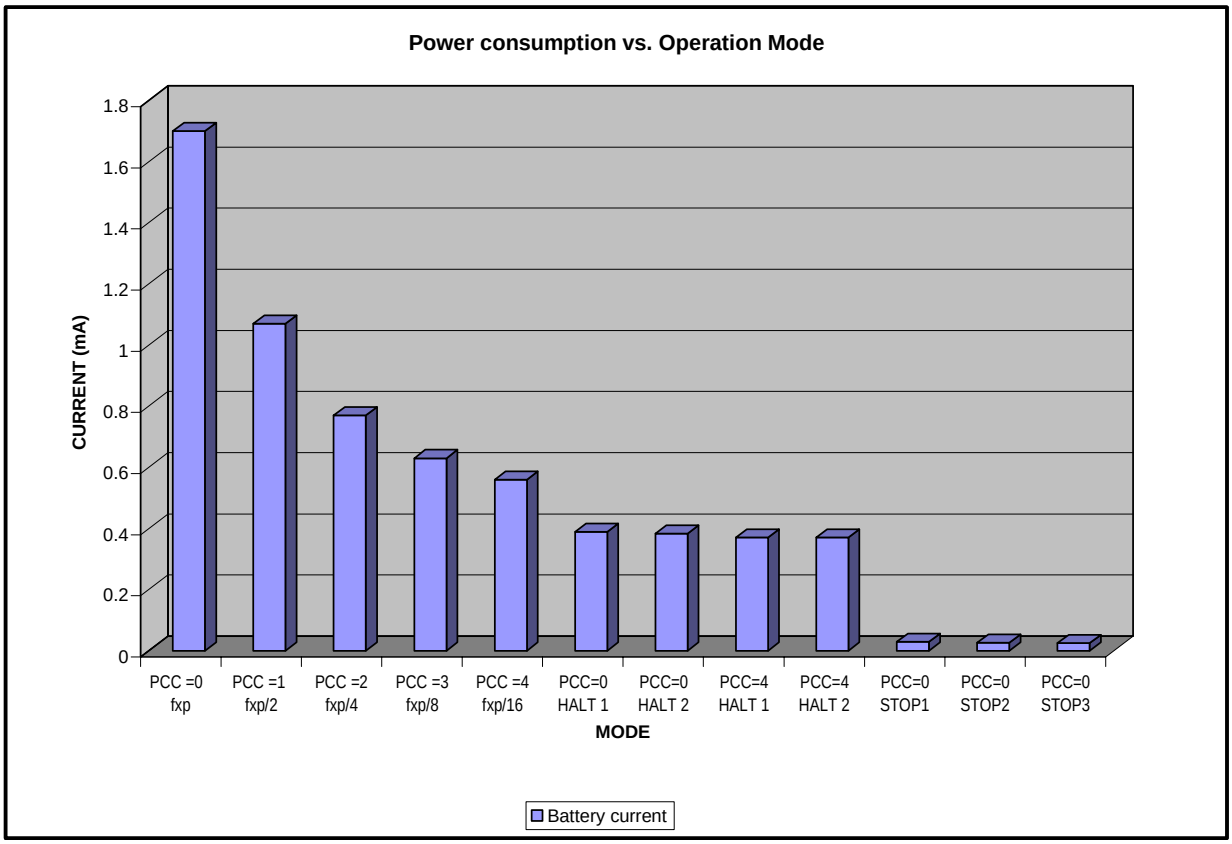


Figure 40. In-system current measurement with 10-MHz main clock and 3V power

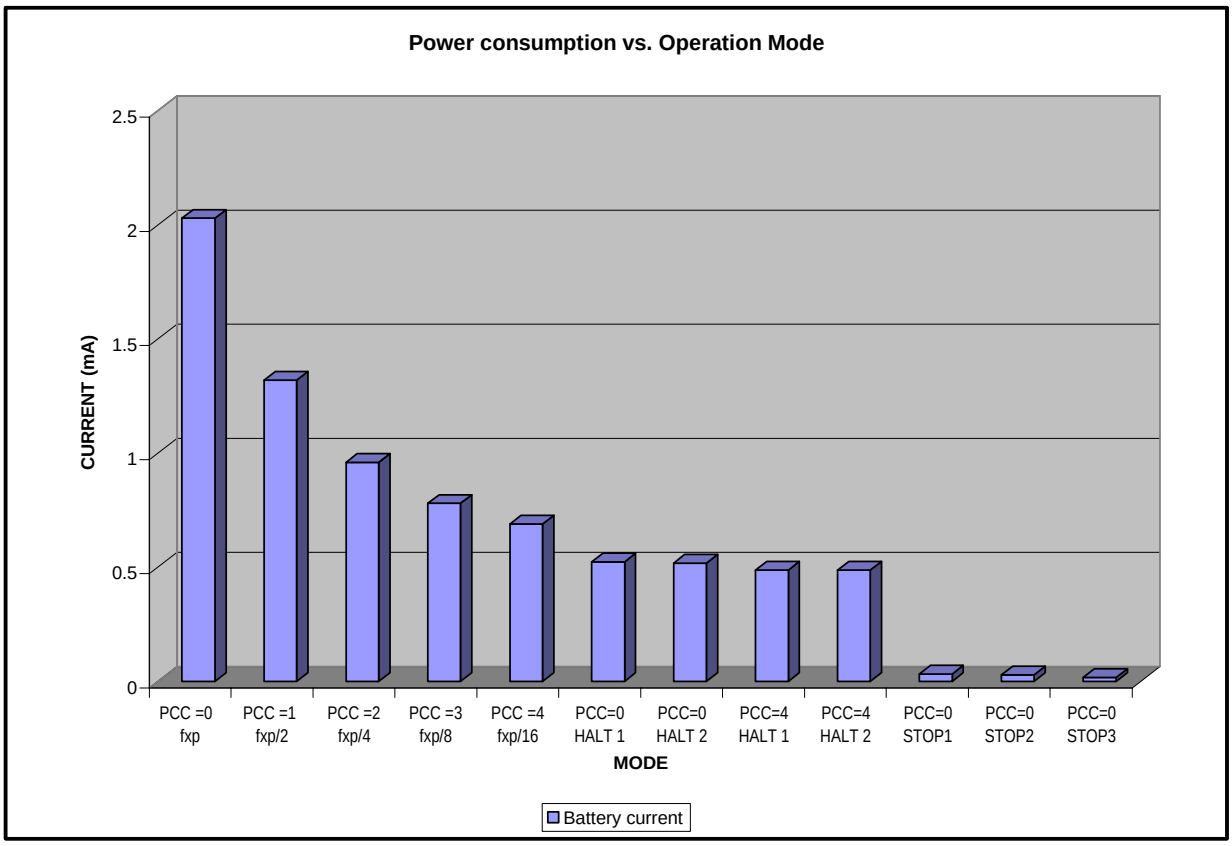
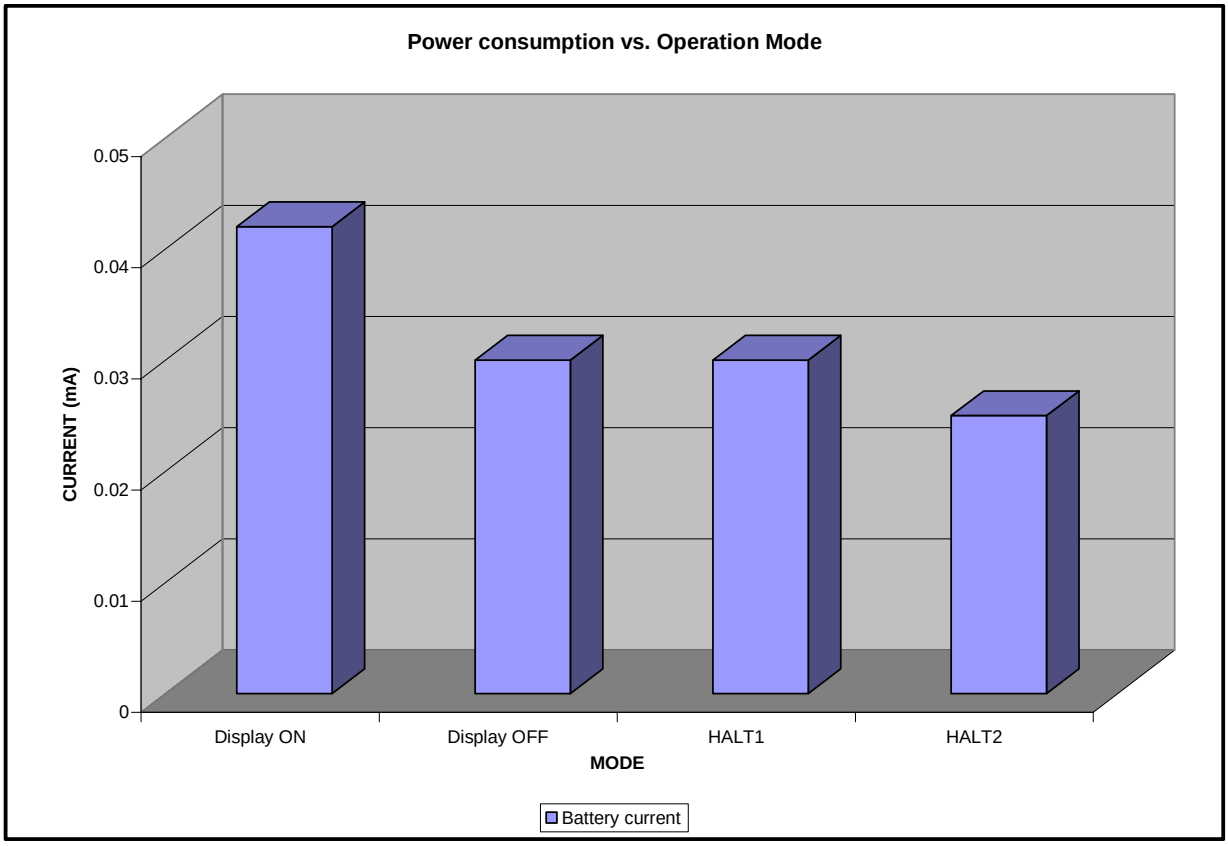


Figure 41. In-system current measurement with 32.768-KHz subclock and 3V power



2.7 Software Modules

The following files make up the software modules for the demonstration program. The table below shows which files were generated by the Applilet and which require modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 10. Demonstration Program Software Modules

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by the Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Int.h	Interrupt-related definitions	Yes	Yes[Note 1]
Int.c	Key-return interrupt-related functions	Yes	No
Int_user.h	User code for key-return interrupt	Yes	Yes[Note 1]
Serial.h	IIC0-related definitions	Yes	Yes[Note 2]
Serial.c	IIC0-related functions	Yes	Yes[Note 2]
Serial_user.c	User code for IIC0-callback routines	Yes	Yes[Note 2]
Watchtimer.h	Watch-timer related definitions	Yes	No
Watchtimer.c	Watch-timer functions	Yes	No
Watchtimer_user.c	User code for timer-interrupt handling	Yes	Yes[Note 3]
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No
define.h	Definitions of navigation-switch values	--	Yes
Lcd.h	LCD high-level function definition	--	Yes
Lcd.c	LCD high-level functions	--	Yes
LcdDrvApp.h	LCD-controller driver definitions	--	Yes
LcdDrvApp.c	LCD-controller driver functions	--	Yes

Note 1: Modify Int.h to add the declaration of **sw3_in** (the variable used to store the navigation switch debounced input) and definitions for switch-input values. Modify Int_user.c to add the variable **sw3_in**, to add the code for handling the key-return interrupt in MD_INTKR(), and to add the delay5msec() routine.

Note 2: Modify Serial.h to add the declarations of IIC0-related global variables, defined in Serial_user.c, and also to declare the functions IIC0_Init( ) (not declared by default) and IIC0_MasterStartAndSend( ). Modify Serial.c slightly to remove settings for P6 not supported for 78K0/LG2, and to replace "asm" versions of DI and EI instructions with NEC Electronics C-Compiler equivalents. Move the setting of the PM6 register to points where the ICCE0 enable is already set. Modify Serial_user.c to have IIC0-callback functions set the IIC0-related global variables, and to add the IIC0_MasterStartAndSend( ) function.

Note 3: Modify Watchtimer_user.c to add the code to handle the periodic INTWT interrupt in the MD_INTWT() routine, which beeps once every 4 seconds when the **g_beep** variable is set ON.

3. Appendix A - Development Tools

This demonstration uses the following software and hardware tools.

3.1 Software Tools

Table 11. Software Tools for Demonstration

Tool	Version	Comments
Applilet for 78K0KE2	V1.51	Source-code generation tool for 78K0/KE2 devices
PM Plus	V5.21	Project manager for program compilation and linking
CC78K0	V3.70	C Compiler for NEC Electronics 78K0 devices
RA78K0	V3.80	Assembler for NEC Electronics 78K0 devices
DF0397.78K	V1.01	Device file for µPD78F0397 device

3.2 Hardware Tools

Table 12. Hardware Tools for Demonstration

Tool	Version	Comments
DemoKit-LG2	V2.00	Demonstration kit for 78K0397 (78K0/LG2)

4. Appendix B – Software Listings

4.1 Main.c

```
/* main.c for NEC Power Down Application Note */
/*
*********************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : main.c
** Abstract : This file implements main function
** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC/CC78K0
**
*********************************************************************************
*/
/*
*********************************************************************************
** Include files
*********************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "watchtimer.h"
#include "serial.h"
/* added header for powerdown application */
#include "pwr_dn.h"

/* include files for DemoKit-LG2 LCD */
#include "lcd.h"
#include "defines.h"
#include "LcdDrvApp.h"

/*
*********************************************************************************
** MacroDefine
*********************************************************************************
*/

//-------------------------------------------------------------------------------
// Function prototypes
//-------------------------------------------------------------------------------
/* main menu functions */
void SetClk(void);
void SetPCC(void);
void DispOff(void);
void Standby(void);
```

```
//-------------------------------------------------------------------------------
// Global variables
//-------------------------------------------------------------------------------
unsigned char g_clock;           /* current clock setting */
unsigned char g_mainclock; /* setting for main clock */
unsigned char g_main_on;   /* track if main clock is on or off */
unsigned char g_iic_on;          /* track if IIC is on/off */
unsigned char g_beep;            /* whether to beep or not in Watchtimer ISR */

//-------------------------------------------------------------------------------
// Global variables
//-------------------------------------------------------------------------------
/* Main Program Menu for invoking sub-menu items */
#define MAIN_MENU_SIZE 4
struct MenuType {
       UCHAR title[9];
       void (*func)(void);
} MainMenu[MAIN_MENU_SIZE] = {
       {"CPU CLK ", SetClk},
       {"SET PCC ", SetPCC},
       {"DISPLAY ", DispOff},
       {"STANDBY ", Standby} };

/*
**-------------------------------------------------------------------------------
**
** Abstract:
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------------
*/
void main( void )
{
UCHAR menu;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       /* set global variables */
       g_clock = CLK_HSR;
       g_mainclock = CLK_HSR;
       g_main_on = ON;
       g_beep = OFF;

       /* start the Watch Timer */
       WT_Start();

       /* initialize IIC */
       IIC0_Init();
       g_iic_on = ON;

       /* initialize LCD */
       LCD_Init();

       while(!sw3_in) {
              LCD_string_shift((unsigned char *)"NEC DEMOKIT-LG2");
              LCD_string_shift((unsigned char *)"POWER DOWN APP NOTE");
#if (CLK_EX_TYPE == CLK_EX_SEL_X1)
              LCD_string_shift((unsigned char *)"EXTERNAL 10 MHZ X1 CRYSTAL");
```

```
#else
		LCD_string_shift((unsigned char *)"DRIVEN 6 MHZ EXCLK");
#endif
		LCD_string_shift((unsigned char *)"PRESS SW3 TO START");
	}

	menu = 0;

	while(1){
		/* clear switch, display current menu selection */
		sw3_in = 0;
		LCD_string(MainMenu[menu].title,0);
		while (sw3_in == 0)
			;                                    /* wait for switch input */
		/* handle switches */
		if (sw3_in == UP) {
			if (menu == 0)
				menu = MAIN_MENU_SIZE - 1;
			else
				menu = menu - 1;
		}
		if (sw3_in == DOWN) {
			menu = menu + 1;
			if (menu == MAIN_MENU_SIZE)
				menu = 0;
		}
		if (sw3_in == RIGHT) {
			/* execute selected function */
				(MainMenu[menu].func)();
		}
		/* return to top of loop to clear switch */
		/* and display new menu selection */
	} /* end while (1) */
}

//----------
// CLOCK
//----------
UCHAR SetClkHSR(void);
UCHAR SetClkEx(void);
UCHAR SetClkSub(void);

#define CLOCK_MENU_SIZE 3
struct ClockMenuType {
	UCHAR title[9];
	UCHAR (*func)(void);
} ClockMenu[CLOCK_MENU_SIZE] = {
	{" C HSR ", SetClkHSR},
	{" C EX ", SetClkEx},
	{" C SUB ", SetClkSub} };

//-----------------------------------------------------------------------------
// Module name: SetClk
// Description: Menu of choices to set CPU clock
//-----------------------------------------------------------------------------
void SetClk(void)
{
UCHAR clk;

	clk = g_clock;

	while (1) {
		sw3_in = 0;
		if (clk == g_clock) {
			LCD_putc(0, 'X'- 0x30);
```

```
                LCD_string(&(ClockMenu[clk].title[1]),1);
            } else {
                LCD_string(ClockMenu[clk].title,0);
            }
            while (sw3_in == 0)
                ;
            if (sw3_in == UP) {
                if (clk == 0)
                    clk = CLOCK_MENU_SIZE - 1;
                else
                    clk = clk - 1;
            }
            if (sw3_in == DOWN) {
                clk = clk + 1;
                if (clk == CLOCK_MENU_SIZE)
                    clk = 0;
            }
            if (sw3_in == RIGHT) {
                clk = (ClockMenu[clk].func)();
            }
            if (sw3_in == LEFT) {
                return;
            }
        }
}

//----------
// SetClkHSR
//----------
UCHAR SetClkHSR(void)
{
        /* if clock is already high-speed Internal-OSC, just return it */
        if (g_clock == CLK_HSR)
            return (CLK_HSR);
        /* if XSEL is 1, peripheral clock has been set to EXCLK or X1 */
        /* spec says XSEL can only be set once, so we can't set back; */
        /* we could set CPU clock back, but then couldn't stop X1/EXCLK */
        /* to the peripherals, so there is no point in switching CPU */

        /* we check g_mainclock also, in case XSEL was set to 1 but */
        /* MCS did not go to 1, clock remained HSR */
        /* won't happen normally - only if clock oscillates but doesn't switch */
        if ((XSEL == 1) && (g_mainclock == CLK_EX))
            return (g_clock);          /* keep the same - CLK_EX or CLK_SUB */
        RSTOP = 0;          /* start the H-S Internal-Osc in case stopped */
        while (RSTS == 0)
            ;               /* wait for it to be stable */

        /* switch off of subclock if on */
        CSS = 0;
        g_clock = CLK_HSR;
        g_mainclock = CLK_HSR;
        g_main_on = ON;                  /* main clock is on */
        return (CLK_HSR);
}

#if (CLK_EX_TYPE == CLK_EX_SEL_X1)
/* SetClkEx for 10 MHz X1 crystal */
//----------
// SetClkEx
//----------
UCHAR SetClkEx(void)
{
int i;
```

```
	/* if clock is already EXCLK, just return it */
	if (g_clock == CLK_EX)
		return (CLK_EX);

	/* if main clock is CLK_EX, and current clock is subclock, just switch back */
	if ((g_clock == CLK_SUB) && g_mainclock == CLK_EX) {
		MSTOP = 0;	/* restart main clock */
		while (OSTC.0 == 0)
			;		/* wait for oscillator to stabilize */
		CSS = 0;	/* switch from subclock */
		while (CLS == 1)
			;		/* wait for switch to take effect */
		g_clock = CLK_EX;
		g_main_on = ON;
		return (CLK_EX);
	}

	/* at this point, mainclock must be HSR, so starting X1 for first time */
	/* first switch back to HSR if running on subclock */
	if (CLS == 1) {
		RSTOP = 0;	/* switch on HSR */
		while (RSTS == 0)
			;		/* wait for stabilize */
		CSS = 0;	/* switch back from subclock */
		while (CLS == 1)
			;		/* wait for switch */
		g_clock = CLK_HSR;
		g_main_on = ON;			/* main clock is on */
	}
	/* now running on HSR */

	AMPH = 0;			/* set to zero for 2 - 10 MHz */
	PM12 &= 0x06;		/* set P122/X2/EXCLK and P121/X1 as inputs */
	OSCCTL = OSCCTL & 0x7F;		/* clear EXCLK (OSCCTL.7) */
	OSCCTL = OSCCTL | 0x40;		/* set EXCLK=0, OSCSEL=1 for X1 crystal input */
	MSTOP = 0;				/* MSTOP = 0 (MOC.7) to enable X1 oscillation
*/

	/* wait for X1 oscillator to stabilize, with timeout */
	for (i=0; i < 10000; i++) {
		if (OSTC.0 == 1)
			break; /* exit loop if OSTC bit 0 is 1, oscillator stabilized */
	}
	if (OSTC.0 == 0) {
		/* loop timed out without stabilization */
		MSTOP = 1;		/* stop clock again */
		sw3_in = 0;
		/* display error message, no oscillation on X1 */
		LCD_string((UCHAR *)"E NO OSC",0);
		while (sw3_in == 0)
			;	/* wait for keypress */
		sw3_in = 0;  /* clear switch */
		return (g_clock);   /* return showing switch back to HSR */
	}
	/* OSTC reported oscillator stabilized, set time to stabilize on STOP exit */
	OSTS = CG_X1STAB_SEL;		/* set oscillation stabilization time for exit STOP
mode */

	/* now need to switch main clock to X1 clock */
	MCM = MCM | 0x05;		/* set XSEL, MCM0 to run CPU and periph on X1 clock */
					/* this is permanent - not allowed to set
XSEL back to 0 */

	/* wait for MCS (MCM.1) to be one to indicate main clock is X1 input */
	/* Note: could check with timeout here, but if clock is oscillating, */
```

```
	/* then MCS should go to one */
	while (MCS == 0)
		;		/* wait for MCS == 1 */

	/* succeeded in changing to X1 crystal clock */

	/* stop H-S Internal OSC */
	RSTOP = 1;
	g_clock = CLK_EX;
	g_mainclock = CLK_EX;
	g_main_on = ON;			/* main clock is on */
	return (CLK_EX);
}
#endif

#if (CLK_EX_TYPE == CLK_EX_SEL_EXCLK)
/* SetClkEx for driven 6 MHz EXCLK - not used */
//----------
// SetClkEx
//----------
UCHAR SetClkEx(void)
{
int i,osccnt;
unsigned char p122_vala,p122_valb;

	/* if clock is already EXCLK, just return it */
	if (g_clock == CLK_EX)
		return (CLK_EX);

	/* if main clock is CLK_EX, and current clock is subclock, just switch back */
	if ((g_clock == CLK_SUB) && g_mainclock == CLK_EX) {
		MSTOP = 0;	/* restart main clock */
					/* no need to wait for stabilization */
		CSS = 0;	/* switch from subclock */
		while (CLS == 1)
			;		/* wait for switch to take effect */
		g_clock = CLK_EX;
		g_main_on = ON;
		return (CLK_EX);
	}

	/* at this point, mainclock must be HSR, so starting EXCLK for first time */
	/* first switch back to HSR if running on subclock */
	if (CLS == 1) {
		RSTOP = 0;	/* switch on HSR */
		while (RSTS == 0)
			;		/* wait for stabilize */
		CSS = 0;	/* switch back from subclock */
		while (CLS == 1)
			;		/* wait for switch */
		g_clock = CLK_HSR;
		g_main_on = ON;			/* main clock is on */
	}
	/* now running on HSR */

	/* start setup for EXCLK input */
	AMPH = 0;		/* set to zero for 2 - 10 MHz */
	PM12.2 = 1;		/* set P122/X2/EXCLK as input */

	/* now that P122/X2/EXCLK is input, check if EXCLK clock is running */

	/* this check needs to be done before changing bits in OSCCTL */
	osccnt = 0;		/* set count of EXCLK changes to zero */
	p122_vala = p122_valb = P12.2;
	for (i=0; i<10000; i++) {
```

```
            p122_vala = P12.2;  /* read new EXCLK input */
            if (p122_vala != p122_valb) {    /* compare with previous */
                  /* has changed, increment count */
                  osccnt++;
                  p122_valb = p122_vala;     /* save previous value */
            }
      }
      /* give error if not oscillating, don't change XSEL,MCM0 */
      if (osccnt == 0) {  /* could check for a minimum number of counts */
            MSTOP = 1;          /* stop clock again */
            sw3_in = 0;
            /* display error message, no oscillation on EXCLK */
            LCD_string((UCHAR *)"E NO EXC",0);
            while (sw3_in == 0)
                  ;     /* wait for keypress */
            sw3_in = 0;  /* clear switch */
            return (g_clock);   /* report switch back to HSR */
      }

      /* we have seen a clock input on the pin, now set for EXCLK mode */
      OSCCTL = OSCCTL | 0xC0;    /* set EXCLK, OSCSEL for external EXCLK input */
      MSTOP = 0;          /* enable EXCLK */

      /* in the case of a driven EXCLK, we do not need to wait for OSTC to */
      /* indicate stabilization; OSTC does not count with driven clock */

      /* now need to switch main clock to EXCLK clock */
      MCM = MCM | 0x05;          /* set XSEL, MCM0 to run CPU and periph on EXCLK */
                                       /* this is permanent - not allowed to set
XSEL back to 0 */

      /* wait for MCS (MCM.1) to be one to indicate main clock is EXCLK input */
      /* Note: could check with timeout here, but if clock is oscillating, */
      /* then MCS should go to one */
      while (MCS == 0)
            ;            /* wait for MCS == 1 */

      /* succeeded in changing to EXCLK driven clock */

      /* stop H-S Internal OSC */
      RSTOP = 1;
      g_clock = CLK_EX;
      g_mainclock = CLK_EX;
      g_main_on = ON;                  /* main clock is on */
      return (CLK_EX);
}
#endif /* SetClkEx for driven 6 MHz EXCLK - not used */

//----------
// SetClkSub
//----------
UCHAR SetClkSub(void)
{
      /* if clock is already Subclock, just return it */
      if (g_clock == CLK_SUB)
            return (CLK_SUB);

      /* switch to subclock */
      CSS = 1;
      /* wait for switch to take effect */
      while (CLS == 0)
            ;
      /* stop main clock */
      if (g_mainclock == CLK_HSR) {
            RSTOP = 1;   /* stop H-S Internal-OSC */
```

```
	} else {
		MSTOP = 1;	/* disable EXCLK input */
	}
	g_main_on = OFF;
	g_clock = CLK_SUB;
	return (CLK_SUB);
}



//-------------------------------------------------------------------------
// Module name: SetPCC
// Description: Menu of choices to set PCC clock
//-------------------------------------------------------------------------
void SetPCC(void){
UCHAR pcc_val,new_pcc;
UCHAR buffer[9] = " PCC N ";

	/* if running off the subclock, display "PCC SUB", wait for key, return */
	if (CSS == 1) {
		LCD_string((UCHAR *)"PCC SUB ",0);
		sw3_in = 0;		/* clear switch */
		while (sw3_in == 0)
			;			/* wait for key press */
		sw3_in = 0;		/* clear again */
		return;
	}

	pcc_val = PCC & 0xF8;	/* mask off low three bits */
	new_pcc = PCC & 0x07;	/* mask to just low three bits */

	while (1) {
		sw3_in = 0;		/* clear key */
		buffer[6] = new_pcc + 0x30;	/* display current new PCC */
		/* display an X at left if we are at current PCC */
		if (new_pcc == PCC & 0x07) {
			buffer[0] = 'X';
		} else {
			buffer[0] = ' ';
		}
		LCD_string(buffer,0);
		/* wait for a key press */
		while (sw3_in == 0)
			;			/* wait for key */
		if (sw3_in == UP) {
			/* change new_pcc 4->3->2->1->0->4 */
			if (new_pcc == 0)
				new_pcc = 4;
			else
				new_pcc = new_pcc - 1;
		}
		if (sw3_in == DOWN) {
			/* change new_pcc 0->1->2->3->4->0 */
			new_pcc = new_pcc + 1;
			if (new_pcc == 5)
				new_pcc = 0;
		}
		if (sw3_in == RIGHT) {
			/* change PCC to new value */
			PCC = pcc_val | new_pcc;
		}
		if (sw3_in == LEFT) {
			/* return from this submenu */
			return;
		}
```

```
	} /* end of while (1) loop */
}

//-------------------------------------------------------------------------
// Module name: TurnDispOff
// Description: Turns LCD Display and IIC off, switching to high power if necessary
//-------------------------------------------------------------------------
void TurnDispOff(void)
{
UCHAR disp_fast;
	/* switch to fast CPU if necessary */
	disp_fast = Disp_Fast();

	/* turn off LCD display and IIC communication */
	LcdDrvOff();

	/* disable ICC0 */
	SetIORBit(PM6, 0x03);	/* turn off drive of SCL0 and SDA0 to conserve power */
	ClrIORBit(IICC0, 0x80);		/* stop transfer */

	/* return to slow CPU if set fast */
	if (disp_fast)
		Disp_Slow();
}

//-------------------------------------------------------------------------
// Module name: TurnDispOn
// Description: Turns IIC and LCD Display back on, switching to high power if necessary
//-------------------------------------------------------------------------
void TurnDispOn(void)
{
UCHAR disp_fast;
	/* switch to fast CPU if necessary */
	disp_fast = Disp_Fast();

	/* reenable IIC */
	IIC0_Init();

	/* reinitialize LCD */
	LCD_Init();

	/* return to slow CPU if set fast */
	if (disp_fast)
		Disp_Slow();
}


//-------------------------------------------------------------------------
// Module name: DispOff
// Description: Turns LCD Display off, waits for key
//-------------------------------------------------------------------------
void DispOff(void){

	LCD_string((UCHAR*)" OFF ",0);	/* display OFF for two seconds */
	Wait(40);

	TurnDispOff();	/* turn LCD and IIC off */

	/* wait until a key is pressed - user can measure current here */
	sw3_in = 0;  /* clear key switch */
	while (sw3_in == 0)
		;

	TurnDispOn();/* turn IIC and LCD back on */
```

```
	/* return to main menu */
}

//----------
// STANDBY
//----------
void StandbyHalt1();
void StandbyHalt2();
void StandbyStop1();
void StandbyStop2();
void StandbyStop3();

#define STBY_MENU_SIZE 5
struct StbyMenuType {
	UCHAR title[9];
	void (*func)(void);
} StandbyMenu[STBY_MENU_SIZE] = {
	{" HALT 1 ", StandbyHalt1},
	{" HALT 2 ", StandbyHalt2},
	{" STOP 1 ", StandbyStop1},
	{" STOP 2 ", StandbyStop2},
	{" STOP 3 ", StandbyStop3} };

#define STBY_HALT1  0
#define STBY_HALT2  1
#define STBY_STOP1  2
#define STBY_STOP2  3
#define STBY_STOP3  4

//-------------------------------------------------------------------------------
// Module name: Standby
// Description: Menu of choices of HALT/STOP modes
//-------------------------------------------------------------------------------
void Standby(void)
{
UCHAR mode;

	mode = STBY_HALT1;

	while (1) {
		sw3_in = 0;
		LCD_string(StandbyMenu[mode].title,0);
		while (sw3_in == 0)
			;
		if (sw3_in == UP) {
			if (mode == 0)
				mode = STBY_MENU_SIZE - 1;
			else
				mode = mode - 1;
		}
		if (sw3_in == DOWN) {
			mode = mode + 1;
			if (mode == STBY_MENU_SIZE)
				mode = 0;
		}
		if (sw3_in == RIGHT) {
			(StandbyMenu[mode].func)();
		}
		if (sw3_in == LEFT) {
			return;
		}
	}
}

//-------------------------------------------------------------------------------
```

```
// Module name: StandbyHalt1
// Description: Halt, keep periodic watch timer interrupt going
//------------------------------------------------------------------------------
void StandbyHalt1(void)
{
       TurnDispOff();      /* turn LCD and IIC off */
       g_beep = ON; /* set for periodic beep */
       sw3_in = 0;         /* set no switch down */

       while (1) {
              HALT();             /* execute HALT to stop CPU */
              NOP();       /* INTWT will bring out of HALT every 0.5 sec */
              NOP();       /* or INTKR will bring out of HALT if SW3 pressed */
              NOP();
              if (sw3_in == LEFT)
                           break; /* end loop if LEFT pressed */
              sw3_in = 0;         /* otherwise clear key and HALT again */
       }

       sw3_in = 0;         /* clear switch */
       g_beep = OFF;/* turn periodic beep off */
       TurnDispOn();/* turn IIC and LCD back on */
}

//------------------------------------------------------------------------------
// Module name: StandbyHalt2
// Description: Halt, stop watch timer, subclock still running
//------------------------------------------------------------------------------
void StandbyHalt2(void)
{
       TurnDispOff();      /* turn LCD and IIC off */

       sw3_in = 0;         /* set no switch down */
       WT_Stop();          /* stop the watch timer */

       while (1) {
              HALT();             /* execute HALT to stop CPU */
              NOP();       /* INTKR will bring out of HALT if SW3 pressed */
              NOP();
              NOP();
              if (sw3_in == LEFT)
                           break; /* end loop if LEFT pressed */
              sw3_in = 0;         /* otherwise clear key and HALT again */
       }

       sw3_in = 0;         /* clear switch */
       WT_Start();         /* restart watch timer */
       TurnDispOn();/* turn IIC and LCD back on */
}

//------------------------------------------------------------------------------
// Module name: StandbyStop1
// Description: Execute STOP, subclock running, watchtimer running
//------------------------------------------------------------------------------
void StandbyStop1(void)
{
       if (g_clock == CLK_SUB) {
              /* cannot enter STOP mode if running on subclock */
              LCD_string((UCHAR*)"NO STOP ",0);
              Wait(20);
              LCD_string((UCHAR*)"CLK SUB ",0);
              Wait(20);
              return;
       }
       TurnDispOff();      /* turn LCD and IIC off */
```

```
	g_beep = ON;  /* set for periodic beep */
	sw3_in = 0;		/* set no switch down */

	while (1) {
		STOP();			/* execute STOP to stop main clock */
		NOP();		/* INTWT will bring out of STOP every 0.5 sec */
		NOP();		/* or INTKR will bring out of STOP if SW3 pressed */
		NOP();
		if (sw3_in == LEFT)
				break; /* end loop if LEFT pressed */
		sw3_in = 0;		/* otherwise clear key and STOP again */
	}

	sw3_in = 0;		/* clear switch */
	g_beep = OFF;/* turn periodic beep off */
	TurnDispOn();/* turn IIC and LCD back on */
}

//-------------------------------------------------------------------------
// Module name: StandbyStop2
// Description: Stop watchtimer, Execute STOP
//-------------------------------------------------------------------------
void StandbyStop2(void)
{
	if (g_clock == CLK_SUB) {
		/* cannot enter STOP mode if running on subclock */
		LCD_string((UCHAR*)"NO STOP ",0);
		Wait(20);
		LCD_string((UCHAR*)"CLK SUB ",0);
		Wait(20);
		return;
	}
	TurnDispOff();	/* turn LCD and IIC off */
	WT_Stop();		/* stop watchtimer */
	sw3_in = 0;		/* set no switch down */

	while (1) {
		STOP();			/* execute STOP to stop main clock */
		NOP();		/* INTKR will bring out of STOP if SW3 pressed */
		NOP();
		NOP();
		if (sw3_in == LEFT)
				break; /* end loop if LEFT pressed */
		sw3_in = 0;		/* otherwise clear key and STOP again */
	}

	sw3_in = 0;		/* clear switch */
	WT_Start();		/* restart Watchtimer */
	TurnDispOn();/* turn IIC and LCD back on */
}

//-------------------------------------------------------------------------
// Module name: StandbyStop3
// Description: Stop watchtimer, stop subclock, Execute STOP
//-------------------------------------------------------------------------
void StandbyStop3(void)
{
	if (g_clock == CLK_SUB) {
		/* cannot enter STOP mode if running on subclock */
		LCD_string((UCHAR*)"NO STOP ",0);
		Wait(20);
		LCD_string((UCHAR*)"CLK SUB ",0);
		Wait(20);
		return;
	}
```

```
	TurnDispOff();		/* turn LCD and IIC off */
	WT_Stop();		/* stop watchtimer */
	OSCSELS = 0;  /* set XT1/XT2 to input port mode to stop subclock */
	LSRSTOP = 1;  /* stop low-speed Internal-OSC if not already stopped */

	sw3_in = 0;		/* set no switch down */

	while (1) {
		STOP();			/* execute STOP to stop main clock */
		NOP();		/* INTKR will bring out of STOP if SW3 pressed */
		NOP();
		NOP();
		if (sw3_in == LEFT)
			break; /* end loop if LEFT pressed */
		sw3_in = 0;		/* otherwise clear key and STOP again */
	}

	sw3_in = 0;		/* clear switch */
	OSCSELS = 1;  /* restart subclock oscillator */
	WT_Start();		/* restart Watchtimer */
	TurnDispOn();/* turn IIC and LCD back on */
}
```

4.2 Pwr_dn.h

```
/*
** pwr_dn.h
** Header file for Power-Down Application Note Program
*/

/* definitions of clock types */
#define CLK_HSR     0      /* clock is high-speed Internal-OSC */
#define CLK_EX      1      /* clock is EXCLK input */
#define CLK_SUB     2      /* clock is Subclock crystal */

/* definitions of whether CLK_EX is X1 crystal or driven EXCLK */
#define CLK_EX_SEL_X1            1      /* X1 crystal */
#define CLK_EX_SEL_EXCLK   2      /* driven EXCLK */

#define CLK_EX_TYPE       CLK_EX_SEL_X1/* set program for X1 crystal */

#ifndef OFF
#define OFF  0
#endif
#ifndef ON
#define ON   1
#endif

/* Global variables */
extern unsigned char g_clock;           /* current clock setting */
extern unsigned char g_mainclock;/* setting for main clock */
extern unsigned char g_main_on;         /* track if main clock is on or off */
extern unsigned char g_iic_on;          /* track if IIC is on/off */
```

```
extern unsigned char g_beep;            /* whether to beep or not in Watchtimer ISR */

UCHAR Disp_Fast();
void Disp_Slow();
```

4.3 Macrodriver.h

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : macrodriver.h
** Abstract : This is the general header file
** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC/CC78K0
**
************************************************************************
*/
#ifndef      _MDSTATUS_
#define _MDSTATUS_

#pragma sfr
#pragma di
#pragma ei
#pragma NOP
#pragma HALT
#pragma STOP

/* data type defintion */
typedef      unsigned long ULONG;
typedef      unsigned int  UINT;
typedef      unsigned short      USHORT;
typedef      unsigned char UCHAR;
typedef      unsigned char BOOL;

#define      ON     1
#define      OFF    0

#define      TRUE   1
#define      FALSE  0

#define IDLE  0                  /* idle status */
#define READ  1                  /* read mode */
#define WRITE 2                  /* write mode */
```

```
#define SET  1
#define CLEAR 0

#define MD_STATUS          unsigned short
#define MD_STATUSBASE             0x0

/* status list definition */
#define MD_OK                     MD_STATUSBASE+0x0   /* register setting OK */
#define MD_RESET                  MD_STATUSBASE+0x1   /* reset input */
#define MD_SENDCOMPLETE                 MD_STATUSBASE+0x2   /* send data complete */
#define MD_OVF                          MD_STATUSBASE+0x3   /* timer count overflow */

/* error list definition */
#define MD_ERRORBASE              0x80
#define MD_ERROR            MD_ERRORBASE+0x0    /* error */
#define MD_RESOURCEERROR    MD_ERRORBASE+0x1    /* no resource available */
#define MD_PARITYERROR            MD_ERRORBASE+0x2    /* UARTn parity error */
#define MD_OVERRUNERROR           MD_ERRORBASE+0x3    /* UARTn overrun error */
#define MD_FRAMEERROR             MD_ERRORBASE+0x4    /* UARTn frame error */
#define MD_ARGERROR         MD_ERRORBASE+0x5    /* Error agrument input error */
#define MD_TIMINGERROR            MD_ERRORBASE+0x6    /* Error timing operation error */
#define MD_SETPROHIBITED    MD_ERRORBASE+0x7    /* setting prohibited */
#define MD_DATAEXISTS             MD_ERRORBASE+0x8    /* Data to be transferred next exists
in TXBn register */
#define MD_SPT                    MD_STATUSBASE+0x8   /*IIC stop*/
#define MD_NACK                   MD_STATUSBASE+0x9   /*IIC no ACK*/
#define MD_SLAVE_SEND_END   MD_STATUSBASE+0x10  /*IIC slave send end*/
#define MD_SLAVE_RCV_END    MD_STATUSBASE+0x11   /*IIC slave receive end*/
#define MD_MASTER_SEND_END MD_STATUSBASE+0x12   /*IIC master send end*/
#define MD_MASTER_RCV_END   MD_STATUSBASE+0x13  /*IIC master receive end*/

/* main clock and subclock as clock source */
enum ClockMode { HiRingClock, SysClock };

/* the value for IMS and IXS */
#define MEMORY_IMS_SET            0xCC
#define MEMORY_IXS_SET            0x00
/* clear IO register bit and set IO register bit */
#define ClrIORBit(Reg, ClrBitMap)Reg &= ~ClrBitMap
#define SetIORBit(Reg, SetBitMap)Reg |= SetBitMap

enum INTLevel { Highest, Lowest };


#define      SYSTEMCLOCK  8000000
#define      SUBCLOCK     32768
#define      MAINCLOCK    8000000
#define      FRCLOCK      8000000
#define      FRCLOCKLOW   240000


#endif
```

4.4 System.h

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : system.h
** Abstract : This file implements device driver for SYSTEM module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
#define      CG_X1STAB_SEL 0x5
#define      CG_X1STAB_STA 0x1f
#define      CG_CPU_CLOCKSEL     0x0

enum CPUClock { SystemClock, Sys_Half, Sys_Quarter, Sys_OneEighth, Sys_OneSixteen,
Sys_SubClock };
enum PSLevel { PS_STOP, PS_HALT };
enum StabTime { ST_Level0, ST_Level1, ST_Level2, ST_Level3, ST_Level4 };

void Clock_Init( void );

#endif
```

4.5 Systeminit.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
```

```
** incurred by customers or third parties arising from the use of this file.
**
** Filename : systeminit.c
** Abstract : This file implements macro initialization.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
****************************************************************************
*/
/*
****************************************************************************
** Include files
****************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "watchtimer.h"
#include "serial.h"
/*
****************************************************************************
** MacroDefine
****************************************************************************
*/

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Init every Macro
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void SystemInit( void )
{
      /* Clock generator initiate */
      Clock_Init();
      /* INT initiate */
      INT_Init();
      /* WT initiate */
      WT_Init();
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Init hardware setting
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
```

```
*/
void hdwinit( void )
{
      DI( );
      SystemInit( );
      EI( );
}
```

4.6 System.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : system.c
** Abstract : This file implements device driver for System module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
/*
************************************************************************
** MacroDefine
************************************************************************
*/

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     Init the Clock Generator and Oscillation stabilization time.
**
** Parameters:
**     None
**
** Returns:
```

```
**     None
**
**------------------------------------------------------------------------
*/
void Clock_Init( void )
{
       ClrIORBit(MCM, 0x05);                    /* high-Internal-OSC operate for CPU */

       SetIORBit(MCM, 0x01);                    /* peripheral hardware clock:frh */
       SetIORBit(PM12, 0x18);                   /* P123/124 input mode */
       ClrIORBit(OSCCTL, 0x20);          /* XT1 input mode */
       SetIORBit(OSCCTL, 0x10);
       SetIORBit(MOC, 0x80);                    /* stop X1 clock */
       PCC = CG_CPU_CLOCKSEL;
}
```

4.7 Int.h

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : int.h
** Abstract : This file implements device driver for INT module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/

#ifndef      _MDINT_
#define      _MDINT_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
#define      EGP_INT      0x0
#define      EGN_INT      0x0
#define      PU7_KR 0x1f
#define      PM7_KR 0x1f
#define      KRM_KR 0x1f

enum ExternalINT {
```

```
       EX_INTP0, EX_INTP1, EX_INTP2, EX_INTP3,
       EX_INTP4, EX_INTP5, EX_INTP6, EX_INTP7
       };

enum INTInputEdge {
       None, RisingEdge, FallingEdge, BothEdge
       };

enum MaskableSource {
       INT_LVI, INT_INTP0, INT_INTP1, INT_INTP2,
       INT_INTP3, INT_INTP4, INT_INTP5, INT_SRE6,
       INT_SR6, INT_ST6, INT_CSI10_ST0, INT_TMH1,
       INT_TMH0, INT_TM50, INT_TM000, INT_TM010,
       INT_AD, INT_SR0, INT_WTI, INT_TM51,
       INT_KR, INT_WT, INT_INTP6, INT_INTP7,
       INT_IIC0_DMU, INT_CSI11, INT_TM001, INT_TM011
       };
void INT_Init( void );
__interrupt void MD_INTKR( void );

/* global value used for debounced switch input */
extern __sreg volatile unsigned char sw3_in;

#endif
```

4.8 Int.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : int.c
** Abstract : This file implements device driver for INT module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/

/*
************************************************************************
** Include files
************************************************************************
*/
```

```
#include "macrodriver.h"
#include "int.h"
/*
*******************************************************************************
** MacroDefine
*******************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes the external interrupt, key return function.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void INT_Init( void )
{
       EGP = EGP_INT;
       EGN = EGN_INT;

       KRMK = 1;                        /* disable INTKR */
       PU7 |= PU7_KR;
       PM7 |= PM7_KR;
       KRM = KRM_KR;                    /* set KR input mode */
       KRPR = 1;
       KRIF = 0;
       KRMK = 0;                        /* enable INTKR */
}
```

4.9 Int_user.h

```
/*
*******************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : int_user.c
** Abstract : This file implements device driver for INT module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
```

```
**
** Compiler : NEC/CC78K0
**
*****************************************************************************
*/

#pragma      interrupt    INTKR  MD_INTKR

/*
*****************************************************************************
** Include files
*****************************************************************************
*/
#include "macrodriver.h"
#include "int.h"
/*
*****************************************************************************
** MacroDefine
*****************************************************************************
*/

/* add include for Watchtimer functions */
#include "watchtimer.h"
/* global value used for debounced switch input */
__sreg volatile unsigned char sw3_in;

/* ------------------
** Function: delay5msec
** Delay for 5 milliseconds
** use Watchtimer if subclock running
** use instruction delay if subclock stopped
*/
void delay5msec(void)
{
unsigned char count,count2;
unsigned char wt_off = 0;

       if (OSCSELS == 0) {
              /* subclock is not running (we are on 8 MHz or 6 MHz clock) */
              /* if PCC=0, each instruction is 0.25 usec; need 20,000 for 5 msec */
              /* do 25 NOPs * 800 = 25 * 50 * 16 */
              /* but if PCC=1, 2, etc, clock is twice, four, etc times as slow */
              /* so we need 25 * 50 * (16/2) or 25 * 50 * (16/4), etc. */
              count2 = 16;
              count = PCC & 0x07; /* get PCC divider bits */
              while (count != 0) {
                     count2 = count2 >> 1;
                     count--;
              }
              /* now count2 is proper multiplier of 25 * 50 * n */
              while (count2 != 0) {
                     for (count = 0; count < 50; count++) {
                            NOP(); NOP(); NOP(); NOP(); NOP();
                            NOP(); NOP(); NOP(); NOP(); NOP();
                            NOP(); NOP(); NOP(); NOP(); NOP();
                            NOP(); NOP(); NOP(); NOP(); NOP();
                            NOP(); NOP(); NOP(); NOP(); NOP();
                     }
                     count2--;
              }
              /* done with delay using instructions */
       } else {
              /* subclock is running, use Watchtimer */
              if (WTM1 == 0) {
                     /* watch timer is off, start it to time our interval */
```

```
                    wt_off = 1;
                    WT_Start();
              }
        for (count = 0; count < ((5000/1950)+1); count++) {
              while (WTIIF == 0)
                    ;
              WTIIF = 0;
        }
        if (wt_off) {
              /* watch timer was off when we came in, stop it again */
              WT_Stop();
        }
       }
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     INTKR Interrupt service routine.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTKR( void )
{
unsigned char sw3_first,sw3_second;

 sw3_first= (~P7) & 0x1f; // read SW3 first time

 delay5msec();                          // delay 5 milliseconds

 sw3_second= (~P7) & 0x1f; // read SW3 second time
 if(sw3_first==sw3_second)
 sw3_in=sw3_first; // debounce SW3
 else
 sw3_in=0;
}
```

4.10 Serial.h

```
/*
*****************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
```

```
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : serial.h
** Abstract : This file implements device driver for SERIAL module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
*****************************************************************************
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_
/*
*****************************************************************************
** Global variables
*****************************************************************************
*/
#define       IIC0_SLAVEADDRESS   0x0


enum TransferMode { Send, Receive };
MD_STATUS IIC0_MasterStart( enum TransferMode , UCHAR , UCHAR );
MD_STATUS IIC0_MasterSendData( UCHAR* , UINT );
MD_STATUS IIC0_MasterReceiveData( UCHAR* , UINT );
void IIC0_Stop( void );
__interrupt void MD_INTIIC0( void ) ;
void IIC0_User_Init( void );
MD_STATUS IIC0_SlaveHandler( void );
MD_STATUS IIC0_MasterHandler( void );
void CALL_IIC0_SlaveAddressMatch( void );
void CALL_IIC0_MasterFindSlave( void );
void CALL_IIC0_MasterSendEnd( void );
void CALL_IIC0_MasterReceiveEnd( void );
void CALL_IIC0_MasterError( MD_STATUS flag );

/* added flags set by callback routines for use by upper level routine */
extern MD_STATUS UI_MasterError;
extern MD_STATUS UI_MasterSendEnd;
extern MD_STATUS UI_MasterReceiveEnd;
extern MD_STATUS UI_MasterFindSlave;

/* added definition for initialize routine */
void IIC0_Init( void );

/* added combined function */
MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum);

#endif
```

4.11 Serial.c

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

```
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : serial.c
** Abstract : This file implements device driver for SERIAL module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/
#define FIX_PM6_SET 1     /* move setting of PM6 to after IICE0 set on */

#pragma interrupt   INTIIC0            MD_INTIIC0

/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"
UCHAR  iic0_m_sta_flag;          /* start flag for send address check by master mode */
UCHAR  *iic0_m_send_pbuf;        /* send data pointer by master mode */
UINT   iic0_m_send_size;         /* send data size by master mode */
UCHAR  *iic0_m_rev_pbuf;         /* receive data pointer by master mode */
UINT   iic0_m_rev_size;          /* receive data size by master mode */
UCHAR  iic0_s_sta_flag;          /* start flag for send address check by slave mode */
UCHAR  *iic0_s_send_pbuf;        /* send data pointer by slave mode */
UINT   iic0_s_send_size;         /* send data size by slave mode */
UCHAR  *iic0_s_rev_pbuf;         /* receive data pointer by slave mode */
UINT   iic0_s_rev_size;          /* receive data size by slave mode */

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes IIC0 module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void IIC0_Init( void )
{
      ClrIORBit(IICC0, 0x80);               /* stop IIC0 */

      SetIORBit(MK1H, 0x1);                 /* disable interrupt */

#if (FIX_PM6_SET == 1)
#else
```

```
      ClrIORBit(PM6, 0x03);                  /* port setting */
#endif
// remove setting of P6 for 78F0397 (78K0/LG2)
//     ClrIORBit(P6, 0x03);

      SetIORBit(IICF0, 0x02);                /* start-condition doesn't need stop-
condition */
      SetIORBit(IICF0, 0x01);                /* comunication reserve - disable */
      SetIORBit(IICC0, 0x10);                /* stop-condition interrupt - enable */
      ClrIORBit(IICC0, 0x08);                /* interrupt control - 8 clock falling edge
*/

      /* transfer clock */
      /* fprs/88 */
      ClrIORBit(IICCL0, 0x08);         /* normal mode */
      ClrIORBit(IICCL0, 0x3);                /* CL00 = 0 CL01 = 0 */
      ClrIORBit(IICX0, 0x1);                 /* disable extension */

      /* selection interrupt priority */
      SetIORBit(PR1H, 0x1);                  /* interrupt priority low */

      ClrIORBit(MK1H, 0x1);                  /* enable interrupt */

      IIC0_User_Init( );

      return;
}

/*
**-------------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for start IIC0 by master mode.
**
** Parameters:
**     enum TransferMode mode :   select transfer mode
**           Send :       send data
**           Receive :    receive data
**     UCHAR adr :  set address for select slave
**     UCHAR wait :  set wait for need waiting when get start condition
**
** Returns:
**     MD_OK
**     MD_ERROR
**     MD_ARGERROR
**
**-------------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterStart( enum TransferMode mode, UCHAR adr, UCHAR wait )
{
      /* bus check */
//     __asm("di"); // replace in-line assembly with psuedo-function
      DI();
      if( IICF0 & 0x40 ){
//           __asm("ei");        /* bus busy */
             EI();  // replace in-line assembly with psuedo-function
             return MD_ERROR;
      }

      /* start IIC0 */
      SetIORBit(IICC0, 0x18);          /* SPIE0 = WTIM0 = 1 */
      SetIORBit(IICC0, 0x80);          /* IICE0 = 1 */
#if (FIX_PM6_SET == 1)
      ClrIORBit(PM6, 0x03);                  /* port setting */
#endif
```

```
	SetIORBit(IICC0, 0x02);		/* generate start condition */
//	__asm("ei");
	EI();  // replace in-line assembly with psuedo-function

	/* wait */
	while( wait-- );

	if( !(IICS0 & 0x2) ){	/* check start condition */
		return MD_ERROR;
	}

	/* set transfer mode to address */
	/* slave would be selected trans or receive from bit0 at address */
	if( mode == Send ){
		ClrIORBit(adr, 0x01);	/* if master is send mode, clear bit0 */
	}
	else if( mode == Receive ){
		SetIORBit(adr, 0x01);	/* if master is receive mode, set bit0 */
	}
	else{
		return MD_ARGERROR;
	}

	iic0_m_sta_flag = 0;
	IIC0 = adr;		/* send address */

	return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**	This function is responsible for stop IIC0.
**
** Parameters:
**	None
**
** Returns:
**	None
**
**-------------------------------------------------------------------------
*/
void IIC0_Stop( void )
{
#if (FIX_PM6_SET == 1)
	SetIORBit(PM6, 0x03);			/* port setting */
#endif
	ClrIORBit(IICC0, 0x80);		/* stop transfer */
	return;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**	This function is responsible for IIC0 data transfering by master mode.
**
** Parameters:
**	UCHAR* txbuf :	transfer buffer pointer
**	UINT txnum :	buffer size
**
** Returns:
**	MD_OK
```

```
**     MD_ERROR :    cannot send address
**
**-------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterSendData(UCHAR* txbuf, UINT txnum)
{
       if( iic0_m_sta_flag == 0 ){
              return MD_ERROR;           /* cannnot send address */
       }

       /* set parameter */
       iic0_m_send_size = txnum;
       iic0_m_send_pbuf = txbuf;

       IIC0 = *iic0_m_send_pbuf ++ ;           /* start transfer */
       iic0_m_send_size--;

       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is responsible for IIC0 data receiving by master mode.
**
** Parameters:
**     UCHAR* rxbuf :      receive buffer pointer
**     USHORT rxnum :      buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR :    cannot send address
**
**-------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterReceiveData(UCHAR* rxbuf, UINT rxnum)
{
       if( iic0_m_sta_flag == 0 ){
              return MD_ERROR;           /* cannnot send address */
       }

       /* set parameter */
       iic0_m_rev_size = rxnum;
       iic0_m_rev_pbuf = rxbuf;

       ClrIORBit(IICC0, 0x08);                 /* clear WTIM0 */
       SetIORBit(IICC0, 0x04);                 /* set ACKE0 */

       SetIORBit(IICC0, 0x20);                 /* start receive */

       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     IIC0 interrupt service routine
**
** Parameters:
**     None
**
** Returns:
**     None
```

```
**
**----------------------------------------------------------------------------
*/
__interrupt void MD_INTIIC0( void )
{
      MD_STATUS sta;
      if( IICS0 & 0x80 ) {
            sta = IIC0_MasterHandler();
      }
      else {
            sta = IIC0_SlaveHandler();
      }
}

/*
**----------------------------------------------------------------------------
**
** Abstract:
**     The function call at IIC0 interrupt request
**
** Parameters:
**     None.
**
** Returns:
**     MD_OK
**     MD_ERROR :    cannot get address
**                   not slave mode
**     MD_SLAVE_RCV_END : all data received
**     MD_SLAVE_SEND_END : all data sended
**     MD_SPT : get stop condition
**
**----------------------------------------------------------------------------
*/
MD_STATUS IIC0_SlaveHandler( void )
{
      /* control for stop condition */
      if( IICS0 & 0x01 ){                           /* get stop condition */
            /* slave send end and get stop condition */
            if( iic0_s_sta_flag &&( iic0_s_send_size == 0 ) ){
                  return MD_SLAVE_SEND_END;
            } else {
                  return MD_SPT;
            }
      }

      /* control for get address */
      if( iic0_s_sta_flag == 0 ){
            if( !(IICS0 & 0x20) ){                  /* check EXC0 -> external code */
                  if( IICS0 & 0x10 ){               /* check COI0 -> address */
                        iic0_s_sta_flag = 1;
                        CALL_IIC0_SlaveAddressMatch();   /* slave address match */
                  }
                  else{
                        return MD_ERROR;
                  }
            }
            else{
                  return MD_ERROR;
            }
      }

      /* slave send control */
      else if( IICS0 & 0x08 ){
            if(!( IICS0 & 0x04 )){                  /* check ACKD0 -> acknowledge */
                  return MD_NACK;
```

```
            }
            IIC0 = *iic0_s_send_pbuf ++ ;
            iic0_s_send_size--;
      }
      /* slave receive control */
      else{
            *iic0_s_rev_pbuf ++ = IIC0;
            iic0_s_rev_size--;
            SetIORBit(IICC0, 0x20);                         /* WREL1 = 1 start receive */
            if( iic0_s_rev_size == 0 ){                     /* check all data received */
                  ClrIORBit(IICC0, 0x04);                   /* clear ACKE0 */
                  return MD_SLAVE_RCV_END;
            }
      }
      return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     The function call at IIC0 interrupt request.
**
** Parameters:
**     None.
**
** Returns:
**     MD_OK
**     MD_ERROR :    cannot get ack after sended address
**                         not master mode
**                         slave did not send ack
**     MD_MASTER_RCV_END : all data received
**     MD_MASTER_SEND_END : all data sended
**
**-------------------------------------------------------------------------
*/
MD_STATUS IIC0_MasterHandler( void )
{
      /* control for stop condition */
      if( !( IICF0 & 0x40 ) ){                      /* get stop condition */
            CALL_IIC0_MasterError(MD_SPT);
            return MD_SPT;
      }

      /* control for sended condition */
      if( !iic0_m_sta_flag ){
            if( IICS0 & 0x4 ){                      /* check ACK */
                  iic0_m_sta_flag = 1;                    /* address complete */
                  CALL_IIC0_MasterFindSlave();
            }
            else{
                  CALL_IIC0_MasterError(MD_NACK);
                  return MD_NACK;
            }
      }
      /* master send control */
      else if( IICS0 & 0x8 ){
            if( !(IICS0 & 0x4) ){                   /* check ACK */
                  SetIORBit(IICC0, 0x01);                 /* generate stop condition */
                  CALL_IIC0_MasterError(MD_NACK);
                  return MD_NACK;
            }

            if( !iic0_m_send_size ){                /* sended finish */
                  SetIORBit(IICC0, 0x01);                 /* generate stop condition */
```

```
                    CALL_IIC0_MasterSendEnd( );
                    return MD_MASTER_SEND_END;
              }
              IIC0 = *iic0_m_send_pbuf ++ ;                  /* send data */
              iic0_m_send_size--;
       }
       /* master receive control */
       else {
              *iic0_m_rev_pbuf ++ = IIC0;                    /* receive data */
              iic0_m_rev_size--;
              if( iic0_m_rev_size == 0 ){                    /* receive finish */
                    ClrIORBit(IICC0, 0x04);                  /* ACK STOP */
                    SetIORBit(IICC0, 0x01);                  /* generate stop condition */
                    CALL_IIC0_MasterReceiveEnd( );
                    return MD_MASTER_RCV_END;
              }
              SetIORBit(IICC0, 0x20);                        /* start receive */
       }
       return MD_OK;
}
```

4.12 Serial_user.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : serial_user.c
** Abstract : This file implements device driver for SERIAL module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "serial.h"

/* added flags set by callback routines for use by upper level routine */
MD_STATUS UI_MasterError;
```

```
MD_STATUS UI_MasterSendEnd;
MD_STATUS UI_MasterReceiveEnd;
MD_STATUS UI_MasterFindSlave;

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function is an empty function for user code when IIC0 initializing
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/

void IIC0_User_Init( void )
{

}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Master Error,
**     callback function open for users operation
**
** Parameters:
**     MD_STATUS flag
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterError( MD_STATUS flag )
{
       /* user operation */
       UI_MasterError = flag;
       return;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Master recevice finish,
**     callback function open for users operation
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterReceiveEnd( void )
{
       /* user operation */
       UI_MasterReceiveEnd = MD_OK;
```

```
	return;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Master send finish,
**     callback function open for users operation
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterSendEnd( void )
{
	/* user operation */
	UI_MasterSendEnd = MD_OK;
	return;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     IIC0 slave address match
**     callback function for users operation
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_SlaveAddressMatch( void )
{
	/* user operation */
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Master find the slave address
**     callback function open for users operation
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void CALL_IIC0_MasterFindSlave( void )
{
	/* user operation */
	UI_MasterFindSlave = MD_OK;
}
```

```
/*
**-------------------------------------------------------------------------
**
** Abstract:
**     Combines IIC0_MasterStart(Send, (sadr), 10)
** and IIC0_MasterSendData(UCHAR* txbuf, UINT txnum)
**
** Parameters:
**     UCHAR sadr :  set address for select slave
**     UCHAR* txbuf :      transfer buffer pointer
**     UINT txnum :  buffer size
**
** Returns:
**     MD_OK
**     MD_ERROR
**     MD_ARGERROR
** MD_NACK - timeout on slave address
**
**-------------------------------------------------------------------------
*/

MD_STATUS IIC0_MasterStartAndSend(UCHAR sadr, UCHAR* txbuf, UINT txnum)
{
MD_STATUS status;
int i,j;

       // Init IIC in case it needs it
//     IIC0_Init();

       // set up for first operation
       UI_MasterError = MD_OK;
       UI_MasterSendEnd = MD_ERROR;
       UI_MasterFindSlave = MD_ERROR;

       status = IIC0_MasterStart(Send, sadr, 10);
       if (status != MD_OK) {
              return status;
       }

       i = 10000;
       do {
              i--;
       } while ( (UI_MasterFindSlave == MD_ERROR) && (UI_MasterError == MD_OK) && ( i > 0 )
);

       if (i == 0) {
              return MD_NACK;
       }

       if (UI_MasterError != MD_OK) {
              return UI_MasterError;
       }

       // got slave address ok here
       status = IIC0_MasterSendData(txbuf, txnum);    // send data bytes
       if (status != MD_OK) {
              return status;
       }
       i = 10000;
       do {
              i--;
       } while ( (UI_MasterError == MD_OK) && (UI_MasterSendEnd == MD_ERROR) && (i > 0) );

       if (i == 0) {
```

```
            return MD_NACK;
      }
      if (UI_MasterError != MD_OK) {
            return UI_MasterError;
      }
      if (UI_MasterSendEnd != MD_OK) {
            return UI_MasterSendEnd;
      }

      // fix to interact with other LCD code for now
//    SetIORBit(MK1H, 0x1);                  /* disable interrupt */
//    ClrIORBit(IICC0, 0x10);                /* clear SPIE0 bit */
//    ClrIORBit(IF1H, 0x01);                 /* clear IICIF0 bit */
      return MD_OK;/* no error */
}
```

4.13 Watchtimer.h

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : watchtimer.h
** Abstract : This file implements device driver for watchtimer module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/
#ifndef _MDWATCHTIMER_
#define _MDWATCHTIMER_
/*
************************************************************************
** MacroDefine
************************************************************************
*/
void WT_Init( void );
MD_STATUS WT_Start( void );
MD_STATUS WT_Stop( void );
__interrupt void MD_INTWT( void );

#endif
```

4.14 Watchtimer.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename :watchtimer.c
** Abstract :This file implements device driver for watchtimer module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler :NEC/CC78K0
**
************************************************************************
*/

/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "watchtimer.h"
/*
************************************************************************
** MacroDefine
************************************************************************
*/

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     This function initializes the watch timer module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void WT_Init( void )
{
      WTM = 0;
      WTPR = 1;           /* low priority level */
      WTMK = 0;
      WTM.7 = 1;          /* watch timer clock: fw = fsub */
```

```
	ClrIORBit(WTM, 0x0c);	/* watch time: 2^14/fw (0.5s at 32.768Kz) */
	ClrIORBit(WTM, 0x70);	/* interval time: 2^6/fw */
	SetIORBit(WTM, 0x20);
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function restarts the watch timer after stopping.
**
** Parameters:
**     None
**
** Returns:
**     MD_OK
**
**-------------------------------------------------------------------------
*/
MD_STATUS WT_Start( void )
{
	/* Enable watch timer operation */
	WTM1 = 1;
	/* Start the 5-bit counter */
	WTM0 = 1;
	WTMK = 0;		/* INTWT enable */

	return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
** Abstract:
**     This function stops the watch timer.
**
** Parameters:
**     None
**
** Returns:
**     MD_OK
**
**-------------------------------------------------------------------------
*/
MD_STATUS WT_Stop( void )
{
	WTMK = 1;		/* INTWT disable */
	/* stop the 5-bit counter */
	WTM1 = 0;
	/* stop watch timer operation */
	WTM0 = 0;

	return MD_OK;
}
```

4.15 Watchtimer_user.c

```
/*
************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : watchtimer_user.c
** Abstract : This file implements device driver for watchtimer module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler : NEC/CC78K0
**
************************************************************************
*/

#pragma interrupt INTWT MD_INTWT
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#include "watchtimer.h"
/* add include of power down variables */
#include "pwr_dn.h"

/*
************************************************************************
** MacroDefine
************************************************************************
*/

/*
**-----------------------------------------------------------------------
**
** Abstract:
**     INTWT interrupt service routine.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------
*/
static hsec = 0;    /* half-second counter */

__interrupt void MD_INTWT( void )
{
UCHAR count;
       /* if beep is off, just return */
       if (g_beep == OFF)
              return;
       /* beep is on, count up half-second counter until 4 seconds */
       hsec = hsec + 1;
```

```
        if (hsec < 8)
                return;             /* haven't reached the time to beep yet */
        hsec = 0;
        /* make a brief sound by toggling P06 at WTI frequency */
        P0.6 = 0;           /* set output low */
        PM0.6 = 0;          /* turn to output */
        for (count = 0; count < 16; count++) {
                while (WTIIF == 0)
                        ;
                WTIIF = 0;
                P0 = P0 ^ 0x40;           /* flip bit to drive high and low alternately*/
        }
        P0.6 = 0;                         /* set latch low */
        PM0.6 = 1;                        /* turn back to input */
}
```

4.16 Option.inc

```
;*****************************************************************************
;**
;** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
;** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
;**
;** Copyright(C) NEC Electronics Corporation 2002 - 2005
;** All rights reserved by NEC Electronics Corporation.
;**
;** This program should be used on your own responsibility.
;** NEC Electronics Corporation assumes no responsibility for any losses
;** incurred by customers or third parties arising from the use of this file.
;**
;** Filename : option.asm
;** Abstract : This file implements OPTION-BYTES/SECURITY-ID setting.
;** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;** Device :  uPD78F0537
;**
;** Compiler :      NEC/CC78K0
;**
;*****************************************************************************

;
;*****************************************************************************
;** MacroDefine
;*****************************************************************************
;
OPTION_BYTE   EQU    00H
POC81  EQU    00H
POC82  EQU    00H
POC83  EQU    00H
CG_ONCHIP     EQU    02H
CG_SECURITY0  EQU    0ffH
CG_SECURITY1  EQU    0ffH
CG_SECURITY2  EQU    0ffH
CG_SECURITY3  EQU    0ffH
CG_SECURITY4  EQU    0ffH
CG_SECURITY5  EQU    0ffH
```

```
CG_SECURITY6  EQU    0ffH
CG_SECURITY7  EQU    0ffH
CG_SECURITY8  EQU    0ffH
CG_SECURITY9  EQU    0ffH
```

4.17 Option.asm

```
;*******************************************************************************
;**
;** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
;** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
;**
;** Copyright(C) NEC Electronics Corporation 2002 - 2005
;** All rights reserved by NEC Electronics Corporation.
;**
;** This program should be used on your own responsibility.
;** NEC Electronics Corporation assumes no responsibility for any losses
;** incurred by customers or third parties arising from the use of this file.
;**
;** Filename : option.asm
;** Abstract : This file implements OPTION-BYTES/SECURITY-ID setting.
;** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;** Device :  uPD78F0537
;**
;** Compiler :      NEC/CC78K0
;**
;*******************************************************************************

;
;*******************************************************************************
;** Include files
;*******************************************************************************
$ INCLUDE (option.inc)
       OPT_SET CSEG AT 80H
OPTION:      DB     OPTION_BYTE
       DB     POC81
       DB     POC82
       DB     POC83
       ONC_SET CSEG AT 84H
ONCHIP:      DB     CG_ONCHIP

       CSEG SECUR_ID
SECURITY0:   DB     CG_SECURITY0
SECURITY1:   DB     CG_SECURITY1
SECURITY2:   DB     CG_SECURITY2
SECURITY3:   DB     CG_SECURITY3
SECURITY4:   DB     CG_SECURITY4
SECURITY5:   DB     CG_SECURITY5
SECURITY6:   DB     CG_SECURITY6
SECURITY7:   DB     CG_SECURITY7
SECURITY8:   DB     CG_SECURITY8
SECURITY9:   DB     CG_SECURITY9
END
```

4.18 defines.h

```
#define UP 0x01
#define DOWN 0x02
#define RIGHT 0x04
#define LEFT 0x08
#define SELECT 0x10

#define BAUDRATE 115200
```

4.19 Lcd.h

```
/*
*********************************************************************************
**
** This file was created for the NEC Application Notes
**
** Copyright(C) NEC Electronics Corporation 2002 - 2006
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : lcd.h
** Abstract : This file implements header for LCD functions on DemoKit-LG2
**
** Device :  uPD78F0397
**
** Compiler : NEC/CC78K0
**
*********************************************************************************
*/
#ifndef      _LCD_H_
#define      _LCD_H_

void Wait(unsigned char Number);
void LCD_Init(void);

__callt extern void LCD_putc(unsigned char digit, unsigned char data);
__callt extern void LCD_string(unsigned char const *point, unsigned char dpos);
__callt extern void LCD_string_shift(unsigned char const *point);

#endif /* _LCD_H_ */
```

4.20 Lcd.c

```
/*
********************************************************************************
**
** This file was created for the NEC Application Notes
**
** Copyright(C) NEC Electronics Corporation 2002 - 2006
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : lcd.c
** Abstract : This file implements LCD functions on DemoKit-LG2
** Modified from DemoKit-LG2 example code to use LcdDrvApp.h
** Modified for Power-down Application Note
**
** Device :  uPD78F0397
**
** Compiler : NEC/CC78K0
**
********************************************************************************
*/
#include "macrodriver.h"
#include "watchtimer.h"    /* for Watchtimer routines */
#include "int.h"    /* for sw3_in definition */
#include "serial.h" /* for IIC functions */
#include "defines.h"
#include "lcd.h"
#include "LcdDrvApp.h"
#include "pwr_dn.h"        /* for power down application */

//-------------------------------------------------------------------------------
// Global constants: minimum ASCII table for 14 segment LCD panel
//-------------------------------------------------------------------------------
unsigned const short characters[43] = {

                                                    0x0e70, // '0'
   0x2060, // '1'
   0x4c32, // '2'
   0x4872, // '3'
   0x4262, // '4'
   0x4a52, // '5'
   0x4e52, // '6'
   0x0070, // '7'
   0x4e72, // '8'
   0x4a72, // '9'
   0x500a, // '+'
   0x4002, // '-'
   0x4232, // '°'
   0x0800, // '_'
   0x2004, // '/'
   0x4c42, // 'o'
   0x0000, // ' ' => space
   0x4672, // 'A'
   0x4e42, // 'B'
   0x0e10, // 'C'
   0x4c62, // 'D'
   0x0e12, // 'E'
   0x0612, // 'F'
   0x4e50, // 'G'
```

```
   0x4662, // 'H'
   0x1008, // 'I'
   0x0c70, // 'J'
   0xa602, // 'K'
   0x0e00, // 'L'
   0x2661, // 'M'
   0x8661, // 'N'
   0x0e70, // 'O'
   0x4632, // 'P'
   0x8e70, // 'Q'
   0xc632, // 'R'
   0x4a52, // 'S'
   0x1018, // 'T'
   0x0e60, // 'U'
   0x8061, // 'V'
   0x9664, // 'W'
   0xa005, // 'X'
   0x2009, // 'Y'
   0x2814 // 'Z'
  };

// string of spaces for clearing display
const unsigned char *s_clear =" ";

//-------------------------------------------------------------------------------
// Global variables
//-------------------------------------------------------------------------------
__sreg unsigned char transmit_buffer[2];
__sreg char message_byte_count;

//-------------------------------------------------------------------------------
// Module name: Wait
// Description: This module delays the program for (number * 50ms).
//-------------------------------------------------------------------------------
void Wait(unsigned char number)
{
UCHAR count;
UCHAR wt_off = 0;

       if (WTM1 == 0) {
              /* watch timer is off, start it to time our interval */
              wt_off = 1;
              WT_Start();
       }
       while (number > 0) {
              for (count = 0; count < ((50000/1950)+1); count++) {
                     while (WTIIF == 0)
                            ;
                     WTIIF = 0;
              }
              number--;
       }
 if (wt_off) {
       /* watch timer was off when we came in, stop it again */
       WT_Stop();
 }
}

//-------------------------------------------------------------------------------
// Module name: LCD_Init
// Description: Calls LcdDrv functions to initialize LCD
//-------------------------------------------------------------------------------
void LCD_Init(void)
{
       LcdDrvInit();
```

```
	LcdDrvOnWait();

	Wait(80); // voltage boost wait time = 4s
	LcdDrvOn();
}

//-------------------------------------------------------------------------
// Module: LCD_putc
// Description: Sent character to LCD controller
//-------------------------------------------------------------------------
__callt void LCD_putc (unsigned char digit, unsigned char data)
{
unsigned char disp_fast;
	disp_fast = Disp_Fast();
 // convert special characters
 switch(data)
 {
 case 0x2f: data = 0x0d; // '_'
 break;
 case 0xf0: data = 0x10; // ' ' => space
 break;
 case 0x80: data = 0x0c; // '°'
 break;
 case 0xfb: data = 0x0a; // '+'
 break;
 case 0xfd: data = 0x0b; // '-'
 break;
 case 0xff: data = 0x0e; // '/'
 break;
 case 0x3f: data = 0x0f; // 'o'
 break;
 default: break;
 }

 // load transmit buffer
 transmit_buffer [0] = ((characters[data])& 0xff00)>>8;
 transmit_buffer [1] = ((characters[data])& 0x00ff);
 message_byte_count = 2;

 digit<<=1;

 LcdDrvSegWrite(&transmit_buffer[0],digit,message_byte_count);
	if (disp_fast) {
		Disp_Slow();
	}
}

//-------------------------------------------------------------------------
// Module: LCD_string
// Description: Send character string to LCD module
//-------------------------------------------------------------------------
__callt void LCD_string(unsigned char const *point, unsigned char dpos)
{
unsigned char disp_fast;
	disp_fast = Disp_Fast();
 while(dpos<=7)
 {
 if(point[0])
 {
 LCD_putc(dpos,*point-0x30);
 *point++;
 }
 else
 {
 LCD_putc(dpos,0xf0);
```

```
 }
 dpos++;
 }
       if (disp_fast) {
              Disp_Slow();
       }
}

//-------------------------------------------------------------------------------
// Module: LCD_string_shift
// Description: Send character string to LCD module
//-------------------------------------------------------------------------------
__callt void LCD_string_shift(unsigned char const *point)
{
 unsigned char dpos=7;
 while(dpos!=0xff)
 {
 if(sw3_in)
 {
 LCD_string(&s_clear[0],0);
 return;
 }
 LCD_string(point,dpos);
 dpos--;
 Wait(4);
 }
 *point++;
 dpos=0;
 while(point[0])
 {
 if(sw3_in)
 {
 LCD_string(&s_clear[0],0);
 return;
 }
 *point++;
 LCD_string(point,dpos);
 Wait(4);
 }
}

static UCHAR disp_save = OFF;
static UCHAR main_save = OFF;
static UCHAR clock_save = OFF;
static UCHAR iic_save = OFF;

/* if CPU is slow, make fast for display */
UCHAR Disp_Fast(void)
{
       if (disp_save == ON)
              return OFF;
       main_save = g_main_on;
       if (g_main_on == OFF) {
              disp_save = ON;
              if (g_mainclock == CLK_HSR){
                     RSTOP = 0;
                     while (RSTS == 0)
                            ;
              } else {
                     MSTOP = 0;
                     /* if we are using X1 crystal, need to wait for Osc stablilze time */
#if (CLK_EX_TYPE == CLK_EX_SEL_X1)
                     while (OSTC.0 == 0)
                            ;            /* wait for X1 clock to stabilize */
#endif
```

```
                NOP();
            }
            g_main_on = ON;
        }
        clock_save = g_clock;
        if (g_clock == CLK_SUB) {
            disp_save = ON;
            CSS = 0;      /* set to main clock */
            if (g_mainclock == CLK_HSR)
                g_clock = CLK_HSR;
            else
                g_clock = CLK_EX;
        }
        iic_save = g_iic_on;
        if (g_iic_on == OFF) {
            disp_save = ON;
            IIC0_Init();
            g_iic_on = ON;
        }
        return (disp_save);
}

/* if we set fast to display, set slow again */
void Disp_Slow()
{
        if (disp_save == OFF)
            return;
        disp_save = OFF;
        if (iic_save == OFF) {
            PM6 |= 0x03;
            IIC0_Stop();
            g_iic_on = OFF;
        }
        if (clock_save == CLK_SUB) {
            CSS = 1;
            while (CLS == 0);
                ;
            g_clock = CLK_SUB;
        }
        if (main_save == OFF) {
            if (g_mainclock == CLK_HSR) {
                RSTOP = 1;
            } else {
                MSTOP = 1;
            }
            g_main_on = OFF;
        }
}
```

4.21 LcdDrvApp.h

```
/************************************************************************
;
; NNNNNN NN EEEEEEEEEEEEEEEEE CCCCCCCCCCCCCCC
; NNNNNNNN NN EEEEEE CCCCCC
; NNNNNNNNNN NN EEEEEE CCCCCC
```

```
; NN NNNNNNNN NN EEEEEEEEEEEEEEEE CCCCCC
; NN NNNNNNNN NN EEEEEE CCCCCC
; NN NNNNNNNNNN EEEEEE CCCCCC
; NN NNNNNN EEEEEEEEEEEEEEE CCCCCCCCCCCCCC
;
; NEC Electronics 78K0/Lx2
;
;*****************************************************************************
; 78K0/Lx2 LCD Control Sample Program
;*****************************************************************************
; LCD Controller/Driver Control -- Function and constant definition file
;*****************************************************************************
;[History]
; 2005.06.-- Newly created
; 2006.01.18 - mod to use Applilet-generated IIC routines instead of iic.c
; - mod for DemoKit-LG2, LCDC = 0x02 instead of 0x0C
;*****************************************************************************/
/*============================================================================
; External reference
;============================================================================*/
#ifndef _LCDDRVAPP_H_
#define _LCDDRVAPP_H_

/*== Various interface functions ==*/
/* LCD driver initialization processing */
extern unsigned char LcdDrvInit( void );
/* LCD driver display ON wait start pre-processing
 (used only when internal step-up mode is selected) */
extern unsigned char LcdDrvOnWait( void );
/* LCD driver display ON processing */
extern unsigned char LcdDrvOn( void );
/* LCD driver display OFF processing */
extern unsigned char LcdDrvOff( void );
/* LCD driver control register write processing */
extern unsigned char LcdDrvCtrWrite( unsigned char *src,
   unsigned char addr,
   unsigned char size );
/* LCD driver segment data write processing */
extern unsigned char LcdDrvSegWrite( unsigned char *src,
   unsigned char addr,
   unsigned char size );
/* LCD driver segment data clear processing */
extern unsigned char LcdDrvSegClr( void );
/* Single byte write processing in LCD driver control register */
extern unsigned char LcdDrvCtrWrite1Byte( unsigned char addr,
    unsigned char data );
/* Single byte write processing of LCD driver segment data */
extern unsigned char LcdDrvSegWrite1Byte( unsigned char addr,
    unsigned char data );
/*== Control register address value ==*/
#define CLDR_ADDR_LCDMD 0x00 /* 0x00: LCD mode select register */
#define CLDR_ADDR_LCDM 0x01 /* 0x01: LCD display mode register */
#define CLDR_ADDR_LCDC 0x02 /* 0x02: LCD clock control register */
#define CLDR_ADDR_VLCG0 0x03 /* 0x03: LCD step-up control register */

/*== Error type value ==*/
enum
{
 CLDR_ERR_NONE /* 0: No errors  */
, CLDR_ERR_NACK /* 1: NACK received  */
, CLDR_ERR_BUSY /* 2: Busy (communication disabled) */
, CLDR_ERR_PARA /* 3: Parameter error */
};

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

```
; Definition of constants
;
; Settings in registers that control the LCD controller/driver
; and main unit's control register
;=========================================================================*/
/*=======================================================================
; Settings in control register for LCD chip's LCD control unit
;=======================================================================*/
/*
;[Communication format]
;
; Address Bit
; 7 6 5 4 3 2 1 0
; +-------+-------+-------+-------+-------+-------+-------+-------+
; 00H LCDMD |SEGSET2|SEGSET1|SEGSET0| 0 | 0 | 0 |MDSET1 |MDSET0 |
; +-------+-------+-------+-------+-------+-------+-------+-------+
; MDSET1/0 : Selects LCD reference voltage generator
; SEGSET2-0 : Sets number of segments (fixed as 40)
;
; +-------+-------+-------+-------+-------+-------+-------+-------+
; 01H LCDM | LCDON | SCOC | VLCON | 0 | 0 | LCDM2 | LCDM1 | LCDM0 |
; +-------+-------+-------+-------+-------+-------+-------+-------+
; LCDM2-0 : Selects LCD controller/driver display mode
; VLCON : Enables/disables operation of step-up circuit
; SCOC : Controls segment pins/common pins output
; LCDON : Enables/disables LCD display
;
; +-------+-------+-------+-------+-------+-------+-------+-------+
; 02H LCDC | 0 | 0 | 0 | 0 | LCDC3 | LCDC2 | LCDC1 | LCDC0 |
; +-------+-------+-------+-------+-------+-------+-------+-------+
; LCDC1/0 : Sets LCD clock
; LCDC3/2 : Sets LCD source clock
;
; +-------+-------+-------+-------+-------+-------+-------+-------+
; 03H VLCG0 |CTSEL1 |CTSEL0 | 0 | 0 | 0 | 0 | 0 | GAIN |
; +-------+-------+-------+-------+-------+-------+-------+-------+
; GAIN : Sets reference voltage level
; CTSEL1/0 : Selects contrast adjustment
;
;
; **Selects each register's settings from the following patterns.
;
;=======================================================================*/
/*-----------------------------------------------------------------------
; LCD mode select register (register address: 00H)
;-----------------------------------------------------------------------*/
/*--- Selects LCD reference voltage generator ---*/

// #define CLDR_LCDMD 0b00000000 /* <1> External resistor division mode */
// #define CLDR_LCDMD 0b00000001 /* <2> Internal resistor division mode */
 #define CLDR_LCDMD 0b00000010 /* <3> Internal step-up mode */
/* 76543210   */
/* -----000 ....... 0: Bit must be set */
/* ||||||||   */
/* ||||||||  **Settings are from patterns <1> to <3> above */
/* ||||||||     */
/* ||||||++-- MDSET1/0 Sets LCD reference voltage generator */
/* |||+++---- <000 fixed>   */
/* +++------- <000 fixed> Selects number of segments (SEGSET2/1/0) */

/*-----------------------------------------------------------------------
; LCD display mode register (register address: 01H)
;-----------------------------------------------------------------------*/
/*-- Selects LCD controller/driver display mode --*/
/*    */
```

```
/*   | Resistor | Step-up | */
/*   | division mode | mode | */
/*   | Time | Bias | Time | Bias | */
/*   |division|method|division|method| */
 #define CLDR_LCDM 0b11100000 /* <1> | 4 | 1/3 | 4 | 1/3 | */
// #define CLDR_LCDM 0b11100001 /* <2> | 3 | 1/3 | 3 | 1/3 | */
// #define CLDR_LCDM 0b11100010 /* <3> | 2 | 1/2 | 4 | 1/3 | */
// #define CLDR_LCDM 0b11100011 /* <4> | 3 | 1/2 | 3 | 1/3 | */
// #define CLDR_LCDM 0b11100100 /* <5> | Static |Set prohibited | */
/* 76543210   */
/* XXX--000..... 0: Bit must be set, */
/* |||||||| X: Bit setting not required, control by software */
/* ||||||||     */
/* ||||||||  **Settings are from patterns <1> to <5> above */
/* ||||||||  **Bits 7 to 5 are controlled by software */
/* ||||||||     */
/* |||||+++-- LCDM2/1/0 Selects LCD controller/driver disp mode */
/* |||++----- <00 fixed>   */
/* ||+------- VLCON (1 fixed) Enables/disables step-up circuit */
/* |+-------- SCOC (1 fixed) Controls segment/common pins output */
/* +--------- LCDON (1 fixed) Enables/disables LCD display */

/*-------------------------------------------------------------------
; LCD clock control register (register address: 02H)
;-------------------------------------------------------------------*/
/*-- Selects LCD source clock (fLCD) & LCD clock --*/
/*    */
/*   | LCD source | LCD clock | */
/*   |clock (fLCD)| LCD clock | */
// #define CLDR_LCDC 0b00000000 /* <1> | fPCL | fLCD/2^6 | */
// #define CLDR_LCDC 0b00000001 /* <2> | fPCL | fLCD/2^7 | */
 #define CLDR_LCDC 0b00000010 /* <3> | fPCL | fLCD/2^8 | */
// #define CLDR_LCDC 0b00000011 /* <4> | fPCL | fLCD/2^9 | */
// #define CLDR_LCDC 0b00001000 /* <5> | fPCL/2 | fLCD/2^6 | */
// #define CLDR_LCDC 0b00001001 /* <6> | fPCL/2 | fLCD/2^7 | */
// #define CLDR_LCDC 0b00001010 /* <7> | fPCL/2 | fLCD/2^8 | */
// #define CLDR_LCDC 0b00001011 /* <8> | fPCL/2 | fLCD/2^9 | */
// #define CLDR_LCDC 0b00001100 /* <9> | fPCL/2^2 | fLCD/2^6 | */
// #define CLDR_LCDC 0b00001101 /* <10>| fPCL/2^2 | fLCD/2^7 | */
// #define CLDR_LCDC 0b00001110 /* <11>| fPCL/2^2 | fLCD/2^8 | */
// #define CLDR_LCDC 0b00001111 /* <12>| fPCL/2^2 | fLCD/2^9 | */
/* 76543210    */
/* ----0000 ..... 0: Bit must be set  */
/* ||||||||    */
/* ||||||||  **Settings are from patterns <1> to <12> above */
/* ||||||||    */
/* ||||||++-- LCDC1/0 Sets LCD clock */
/* ||||++---- LCDC3/2 Sets LCD source clock */
/* ++++------ <0000 fixed>  */
/*-------------------------------------------------------------------
; LCD step-up control register (register address:03H)
;-------------------------------------------------------------------*/
/*-- Selects reference voltage (VLC2) level & contrast adjustment (TYP. value) -*/
/*    */
/*   |Reference| Contrast | */
/*   | voltage | adjustment | */
/*   | (VLC2) | (TYP. value) | */
/*   |level *1 | VLC0 | VLC1 | VLC2 | */
 #define CLDR_VLCG0 0b10000000 /* <1> |1.5V | 4.89V | 3.27V | 1.633V | */
// #define CLDR_VLCG0 0b11000000 /* <2> |1.5V | 4.71V | 3.13V | 1.567V | */
// #define CLDR_VLCG0 0b00000000 /* <3> |1.5V | 4.50V | 3.00V | 1.500V | */
// #define CLDR_VLCG0 0b01000000 /* <4> |1.5V | 4.29V | 2.87V | 1.433V | */
// #define CLDR_VLCG0 0b10000001 /* <5> |1.0V | 3.29V | 2.27V | 1.133V | */
// #define CLDR_VLCG0 0b11000001 /* <6> |1.0V | 3.21V | 2.13V | 1.067V | */
// #define CLDR_VLCG0 0b00000001 /* <7> |1.0V | 3.00V | 2.00V | 1.000V | */
```

```
// #define CLDR_VLCG0 0b01000001 /* <8> |1.0V | 2.79V | 1.87V | 0.933V | */
/* 76543210     */
/* 00-----0 ...... 0: Bit must be set   */
/* ||||||||     */
/* ||||||||  **Settings are from patterns <1> to <8> above */
/* ||||||||     */
/* |||||||+-- GAIN Sets reference voltage level */
/* ||+++++--- <00000 fixed>   */
/* ++-------- CTSEL1/0 Selects contrast adjustment */
/*       */
/* *1: The reference voltage level is selected 1.5 V when the target LCD panel */
/* is rated at 4.5V, and is selected as 1.0 V when the target LCD panel is */
/* rated at 3.0 V.     */
/*====================================================================
; Definition of main control register settings
;====================================================================*/
/*--------------------------------------------------------------------
; Selects clock output
;--------------------------------------------------------------------*/
/*-- Selects PCL's output clock --*/
/* fSUB=32.768kHz */
/* fPRS=peripheral clock */
/*    | fPRS=10MHz | fPRS=20MHz | */
// #define CLDR_CKS 0b00000110 /* <1> | fPRS/2^6 | 156.25kHz | 312.5 kHz | */
// #define CLDR_CKS 0b00000111 /* <2> | fPRS/2^7 | 78.125kHz | 156.25kHz | */
 #define CLDR_CKS 0b00001000 /* <3> | fSUB | 32.768kHz | 32.768kHz | */
/* 76543210     */
/* ---X0000 ...... 0: Bit must be set   */
/* |||||||| X: Bit setting not required, control by software */
/* ||||||||     */
/* ||||||||  **Settings are from patterns <1> to <3> above */
/* ||||||||  **Bit 4 is controlled by software  */
/* ||||||||     */
/* ||||++++-- CCS3/2/1/0 Selects PCL's output clock */
/* |||+------ CLOE (0 fixed) Enables/disables clock output to LCD */
/* +++------- <000 fixed>    */
/*-----------------------------------------< END OF FILE >----------*/
#endif /* _LCDDRVAPP_H */
```

4.22 LcdDrvApp.c

```
/*****************************************************************************
;
; NNNNNN NN EEEEEEEEEEEEEEEEE CCCCCCCCCCCCCCC
; NNNNNNNN NN EEEEEE CCCCCC
; NNNNNNNNNN NN EEEEEE CCCCCC
; NN NNNNNNNN NN EEEEEEEEEEEEEEEEE CCCCCC
; NN NNNNNNNN NN EEEEEE CCCCCC
; NN NNNNNNNNN EEEEEE CCCCCC
; NN NNNNNN EEEEEEEEEEEEEEEE CCCCCCCCCCCCCCC
;
; NEC Electronics 78K0/Lx2
;
;*****************************************************************************
; 78K0/Lx2 LCD control sample program
;*****************************************************************************
```

```
; LCD controller/driver control -- Processing file--
;************************************************************************
;[History]
; 2005.06.-- Newly created
; 2005.07.01 [050701] Provisionally added voltage supply to VLC0
; 2006.01.11 Modified to use Applilet-generated IIC routines - rdh
; 2006.01.27 Correction in LcdDrvOff in turning off VLCON - rdh
; 2006.03.30 Use routines for writing only for Application Notes
************************************************************************/
#pragma sfr

/*=======================================================================
; INCLUDE
;=======================================================================*/
#include "macrodriver.h"
#include "serial.h"
#include "LcdDrvApp.h"

static void LcdDrvClkOut( void );
static void LcdDrvClkStop( void );

/*=======================================================================
; Definition of control area for LCD driver and various settings
;=======================================================================*/
/*===========
; Slave ID
;==========*/
#define CSLV_ID_LCDCTL 0b01110000 /* Control register (LCDCTL) */
#define CSLV_ID_LCDSEG 0b01110010 /* Segment data (LCDSEG) */

/*===========
; Definition of control register settings on main side
;==========*/
/*-- Clock output select register --*/
#define LDR_CKS CKS
#define LDR_CKS_CLOE LDR_CKS.4 /* Clock output enable/disable */

/*-- Port/port mode register (directly connected in microcontroller) --*/
#define PO_LDR_RST P13.0 /* Reset to LCD chip */
#define PM_LDR_OUT PM14.0 /* Clock output to LCD chip */

/*-- Port/port mode register --*/  /*[050701]>>*/
#define PO_VLC0_HL P7.7 /* Voltage supply to VLC0 */
#define PM_VLC0_HL PM7.7 /* Voltage supply to LLC0 [050701] */

/*===========
 Control register setting
==========*/
/* Selects LCD reference voltage generator: internal step-up mode */
#define CLDR_LCDMD_VOL 0b00000010

/*****************************************************************
; LCD driver initialization
;----------------------------------------------------------------
; [I N] -
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy
;*****************************************************************/
unsigned char LcdDrvInit( void )
{
 register unsigned char result = CLDR_ERR_NONE;

 PO_LDR_RST = 1; /* Cancels LCD chip's reset status */
 LDR_CKS = ( CLDR_CKS & 0b00001111); /* Output clock setting (CCS3-0) */

 /*-- Enables clock output to LCD chip --*/
```

```
 LcdDrvClkOut();

 /*-- Selects reference voltage generator --*/
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDMD, CLDR_LCDMD );

 /*-- Clears segment data --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvSegClr();
 }

 /*-- Selects display mode --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00000111 ));
 }

 /*-- LCD clock setting --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDC, CLDR_LCDC );
 }
 return ( result );
}

/*****************************************************************
; LCD driver display ON wait start pre-processing (when step-up mode is selected)
;----------------------------------------------------------------
; << Note >>
;
; Only N is called when internal step-up mode is selected for the reference
; voltage generator. After this processing is called, a wait period
; of at least 500 ms should occur, then the "LcdDrvOn" should be called.
;
;----------------------------------------------------------------
; [I N] -
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy
;*****************************************************************/
unsigned char LcdDrvOnWait( void )
{
#if (CLDR_LCDMD==CLDR_LCDMD_VOL)
 /* Reference voltage generator: internal step-up mode */
 register unsigned char result = CLDR_ERR_NONE;

 /*-- Enables clock output to LCD chip --*/
 LcdDrvClkOut();

 /*-- Sets LCD step-up level and contrast--*/
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_VLCG0, CLDR_VLCG0 );

 /*-- Enables LCD step-up --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00100111 ));
 }
 /* -- After 500 ms, the "LcdDrvOnWait" function must be called -- */
 return ( result );
#else/*!(CLDR_LCDMD==CLDR_LCDMD_VOL)*/
 /* Reference voltage generator: resistor division mode */
 return ( 0 );
#endif/*(CLDR_LCDMD)*/
}

/*****************************************************************
; LCD driver ON processing
;----------------------------------------------------------------
; << Note >>
;
; When internal step-up mode has been selected for the reference voltage
```

```
; generator, after the "LcdDrvOnWait" has been called, a wait period of
; at least 500 ms must occur before calling the next function.
;
;-------------------------------------------------------------------
; [I N] -
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy
;*******************************************************************/
unsigned char LcdDrvOn( void )
{
 register unsigned char result = CLDR_ERR_NONE;

#if (CLDR_LCDMD==CLDR_LCDMD_VOL)
 /* Reference voltage generator: internal step-up mode */
 /*-- Setting of deselect potential output --*/
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b01100111 ));

 /*-- Display ON setting --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b11100111 ));
 }

#else/*!(CLDR_LCDMD==CLDR_LCDMD_VOL)*/
 /* Reference voltage generator: resistor division mode */
 /*-- Enables clock output to LCD chip --*/
 LcdDrvClkOut();

 /*-- Setting of deselect potential output --*/
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b01000111 ));

 /*-- Display ON setting --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b11000111 ));
 }

#endif/*(CLDR_LCDMD)*/
 return ( result );
}

/*******************************************************************
; LCD driver display OFF processing
;-------------------------------------------------------------------
; [I N] -
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy
;
; *Clears AX register
;*******************************************************************/
unsigned char LcdDrvOff( void )
{
 register unsigned char result = CLDR_ERR_NONE;

 /*-- Clears segment data --*/
 result = LcdDrvSegClr();

 /*-- Display OFF setting --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b01100111 ));
 }

 /*-- Segment/common buffer output disable setting --*/
 if( result == CLDR_ERR_NONE ){
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00100111 ));
 }

#if (CLDR_LCDMD==CLDR_LCDMD_VOL)
```

```
 /*-- LCD step-up disable setting --*/
 if( result == CLDR_ERR_NONE ){
#if 0 /* correction 060127 - bit 5 is 1, does not turn off VLCON */
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00100111 ));
#else /* correction turns off VLCON by having bit 5 as zero */
 result = LcdDrvCtrWrite1Byte( CLDR_ADDR_LCDM, ( CLDR_LCDM & 0b00000111 ));
#endif
 }

#endif/*(CLDR_LCDMD)*/

 if( result == CLDR_ERR_NONE ){
 /*-- Disables clock output to LCD chip --*/
 LcdDrvClkStop();
 }
 return ( result );
}

/*****************************************************************
; Writes LCD driver control data
;----------------------------------------------------------------
; [I N] src : Control register data's storage address
; addr : Control register address value
; size : Number of bytes to be transmitted (4 bytes maximum)
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
;*****************************************************************/
unsigned char SLDRCTLW( unsigned char *src,
  unsigned char addr, unsigned char size )
{
 register unsigned char result = CLDR_ERR_NONE;
 register unsigned char cnt;
 MD_STATUS status;
 unsigned char buf[5];
 unsigned char uc;

 /*-- Enables clock output to LCD chip --*/
 LcdDrvClkOut();

 /*-- Checks parameters --*/
 if(( size == 0 )||( addr > 0x03 )||(( 0x03+1 - addr ) < size )){
 result = CLDR_ERR_PARA;
 }
 buf[0] = addr;
 for (uc = 0; uc < size; uc++) {
 buf[uc+1] = src[uc];
 }

 status = IIC0_MasterStartAndSend( CSLV_ID_LCDCTL, buf, size + 1 );
 if (status != MD_OK)
 result = CLDR_ERR_NACK;
 return ( result );
}

/*****************************************************************
; Writes LCD driver segment data
;----------------------------------------------------------------
; [I N] src : Address of segment data to be written
; addr : Data address for start of write operation
; size : Number of bytes to be transmitted (maximum: 20 bytes)
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
;----------------------------------------------------------------
; ** Smooths data before transmitting segment data.
;
; <Received data>
; bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
```

```
; +----+
; 1st byte (00H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
; |<- segment:S1 ->|<- segment:S0 ->|
;
; +----+----+----+----+----+----+----+----+
; 2nd byte (01H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
;  :
; : :
; +----+----+----+----+----+----+----+----+
; 19th byte (12H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
; +----+----+----+----+----+----+----+----+
; 20th byte (13H)|COM3|COM2|COM1|COM0|COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
; |<- segment:S39 ->|<- segment:S38 ->|
;
; <Sent data>
; bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
; +----+----+----+----+----+----+----+----+
; 1st byte(00H)| 0 | 0 | 0 | 0 |COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
;   |<- segment:S0 ->|
;    Address:00H |
;
; +----+----+----+----+----+----+----+----+
; 2nd byte (01H)| 0 | 0 | 0 | 0 |COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
;  :
;  :
; +----+----+----+----+----+----+----+----+
; 39th byte (26H)| 0 | 0 | 0 | 0 |COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
; +----+----+----+----+----+----+----+----+
; 40th byte (27H)| 0 | 0 | 0 | 0 |COM3|COM2|COM1|COM0|
; +----+----+----+----+----+----+----+----+
;   |<- segment:S39 ->|
;   | Address:27H |
;
;*****************************************************************/
unsigned char LcdDrvSegWrite( unsigned char *src,
  unsigned char addr, unsigned char size )
{
 register unsigned char result = CLDR_ERR_NONE;
 register unsigned char cnt;
 MD_STATUS status;
 unsigned char buf[41];
 unsigned char uci,ucd;

 /*-- Enables clock output to LCD chip --*/
 LcdDrvClkOut();

 /*-- Checks parameters --*/
 if(( size == 0 )||( addr > 0x13 )||(( 0x13+1 - addr ) < size )){
 result = CLDR_ERR_PARA;
 }

 buf[0] = addr*2;
 for (uci = 0, ucd = 1; uci < size; uci++, ucd = ucd + 2) {
 buf[ucd] = src[uci] & 0x0f ;
 buf[ucd+1] = (src[uci] >> 4) & 0x0f;
 }

 status = IIC0_MasterStartAndSend( CSLV_ID_LCDSEG, buf, (size * 2) + 1 );
```

```
 if (status != MD_OK)
 result = CLDR_ERR_NACK;
 return ( result );
}

/*****************************************************************
; Clears LCD driver segment data
;-----------------------------------------------------------------
; [I N] -
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy
;*****************************************************************/
unsigned char LcdDrvSegClr( void )
{
 register unsigned char result = CLDR_ERR_NONE;
 register unsigned char cnt;
 MD_STATUS status;
 unsigned char buf[41];
 unsigned char uc;

 /*-- Enables clock output to LCD chip --*/
 LcdDrvClkOut();

 buf[0] = 0; // start at location zero
 for (uc = 1; uc < 41; uc++) {
 buf[uc] = 0;
 }

 status = IIC0_MasterStartAndSend( CSLV_ID_LCDSEG, buf, 41 );
 if (status != MD_OK)
 result = CLDR_ERR_NACK;
 return ( result );
}

/*****************************************************************
; Writes to LCD driver control register (1 byte setting)
;-----------------------------------------------------------------
; [I N] addr : control register address value
; data : control register data
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
;*****************************************************************/
unsigned char LcdDrvCtrWrite1Byte( unsigned char addr, unsigned char data )
{
 register unsigned char result = CLDR_ERR_NONE;
 MD_STATUS status;
 unsigned char buf[2];
 unsigned char uc;

 /*-- Enables clock output to LCD chip --*/
 LcdDrvClkOut();

 buf[0] = addr;
 buf[1] = data;

 status = IIC0_MasterStartAndSend( CSLV_ID_LCDCTL, buf, 2 );
 if (status != MD_OK)
 result = CLDR_ERR_NACK;
 return ( result );
}

/*****************************************************************
; Writes LCD driver segment data (1 byte setting)
;-----------------------------------------------------------------
; [I N] addr : Control register address value
; data : control register data
; [OUT] 0= Setting OK, 1 = NACK received, 2 = Busy, 3 = Parameter error
```

```
;*****************************************************************/
unsigned char LcdDrvSegWrite1Byte( unsigned char addr, unsigned char data )
{
 register unsigned char result = CLDR_ERR_NONE;
 register unsigned char work;
 MD_STATUS status;
 unsigned char buf[5];
 unsigned char uc;

 /*-- Enables clock output to LCD chip --*/
 LcdDrvClkOut();

 buf[0] = addr;
 buf[1] = data & 0x0f;
 buf[2] = ( data >>4 ) & 0x0f;

 status = IIC0_MasterStartAndSend( CSLV_ID_LCDSEG, buf, 3 );
 if (status != MD_OK)
 result = CLDR_ERR_NACK;
 return ( result );
}

/*****************************************************************
; Enables clock and power output to LCD chip
;-----------------------------------------------------------------
; [I N] -
; [OUT] -
;*****************************************************************/
static void LcdDrvClkOut( void )
{
 PM_LDR_OUT = 0;
 LDR_CKS_CLOE = 1;
#if (CLDR_LCDMD==CLDR_LCDMD_VOL)  /*[050701]>>*/
 PO_VLC0_HL = 0; /* Low output (power is supplied to VLC0)*/
#else /*!(CLDR_LCDMD==CLDR_LCDMD_VOL)*/
 PO_VLC0_HL = 1; /* High output (power is not supplied to VLC0)*/
#endif/*(CLDR_LCDMD)*/  /*[050701]<<*/
}

/*****************************************************************
; Disables clock and power output to LCD chip
;-----------------------------------------------------------------
; [I N] -
; [OUT] -
;*****************************************************************/
static void LcdDrvClkStop( void )
{
 LDR_CKS_CLOE = 0;
 PM_LDR_OUT = 1;
 PO_VLC0_HL = 0;  /*[050701]*/
}

/*-----------------------------------------------< END OF FILE >---*/
```