Application Note

Safety Features of NEC Electronics Microcontrollers

Document no. U18270EU1V0AN00

Revision History

Date	Revision	Section	Description
July 2006	—	—	First release

Contents:

1. Introduction

This document provides simple examples of how to use the peripherals included in NEC Electronics' microcontroller devices.

This document includes:

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

The Applilet is a software tool that generates peripheral driver code. It affords a convenient means of generating code for quick evaluation.

Reference the device user manual and other related documents for further details.

1.1 Overview of Safety Features

The safety features built into NEC Electronics' microcontrollers enable you to start the system in reset mode and restart from the reset condition. Other safety features apply to power supply problems and software loops. For example, when the power supply voltage sags below a specified level, a low-voltage detection circuit interrupts the system in order to save the system's state. If the program goes into an infinite loop, a watchdog timer resets the system. These features keep the system running in a safe condition.

This application note shows how to use the following safety features:

♦ Reset by external reset pin

♦ Power-on-clear reset

♦ Low-voltage detector reset or interrupt

♦ Watchdog-timer reset

2. Safety Features of NEC Electronics Microcontrollers

This document provides a short overview of each safety feature in the NEC Electronics 78K0/Kx2 family of devices.

2.1 Overview of Safety Features

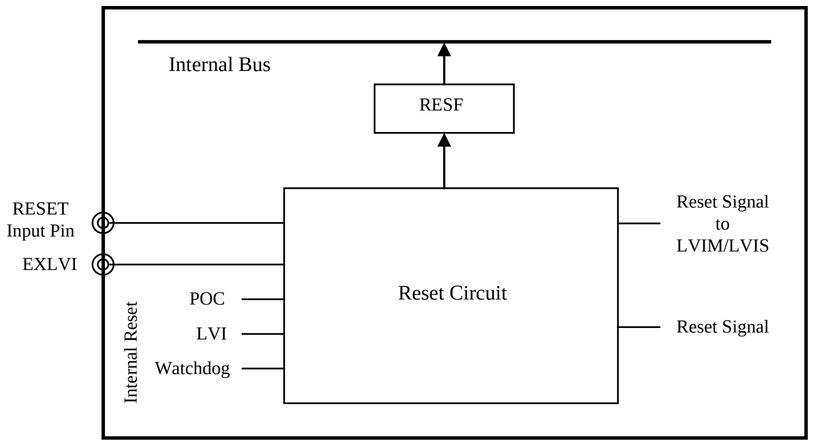
2.1.1 Reset

The following hardware resources generate a reset signal:

- External reset-input pin
- Watchdog-timer overflow (internal)
- Power-on-reset (internal)
- Low-voltage detection (internal)

There is no functional difference between an external and internal reset. In both cases, after asserting the reset signal, the reset vector (locations 0000H-0001H) specifies where program execution starts. Reset also sets I/O registers and port pins (except RESF) to default values.

Figure 1. Block Diagram for Reset



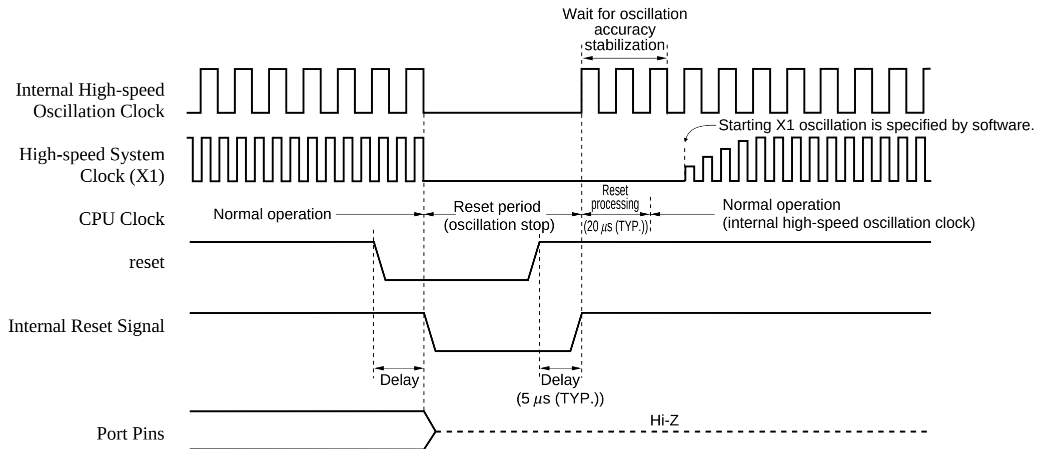
The reset operation sets the reset flag, which identifies the cause of the reset.

Table 1. RESF Status when Reset Request is Generated

Reset Source / Flag	$\overline{\text{reset}}$ Input	Reset by POC	Reset by WDT	Reset by LVI
WDTRF	Cleared (0)	Cleared (0)	Set (1)	Held
LVIRF			Held	Set (1)

Figure 2 shows the timing of the external-reset pin and internal-reset signal. The delay after asserting or releasing avoids the possibility of repeated internal resets due to noise on the reset pin. Once the internal reset releases, program execution starts using the internal high-speed internal-oscillator clock. After the external X1 oscillator stabilizes, the system can switch to the CPU clock to use X1 as the system clock.

Figure 2. Timing of External-Reset Pin and Internal-Reset Signal


Wait for oscillation accuracy stabilization
Internal High-speed Oscillation Clock
Starting X1 oscillation is specified by software.
High-speed System Clock (X1)
CPU Clock
Normal operation
Reset period (oscillation stop)
Reset processing (20 μs (TYP.))
Normal operation (internal high-speed oscillation clock)
reset
Internal Reset Signal
Delay
Delay (5 μs (TYP.))
Port Pins
Hi-Z

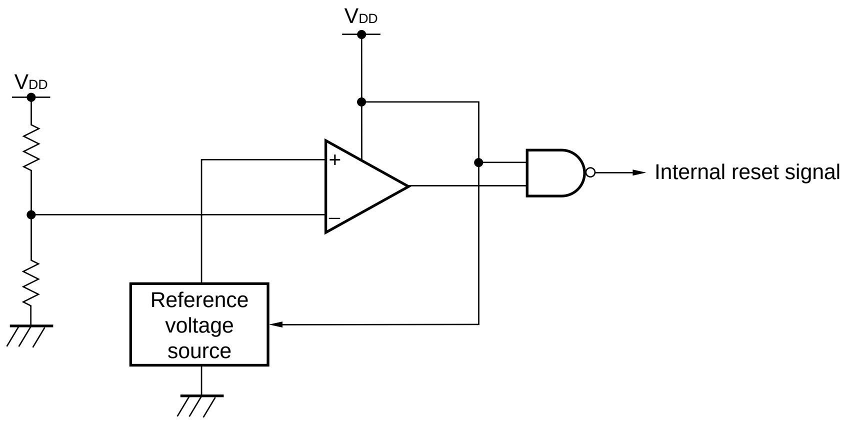
2.1.2 Power-On-Clear (POC) Reset

The power-on-clear function offers the following features:

- ♦ Holds the device in reset during power up, while VDD stabilizes
- ♦ Lets you select a value for the power-up (POC) voltage (1.59 or 2.7V, selected with the option byte)
- ♦ Resets the device if VDD falls to a level which may cause unstable operation

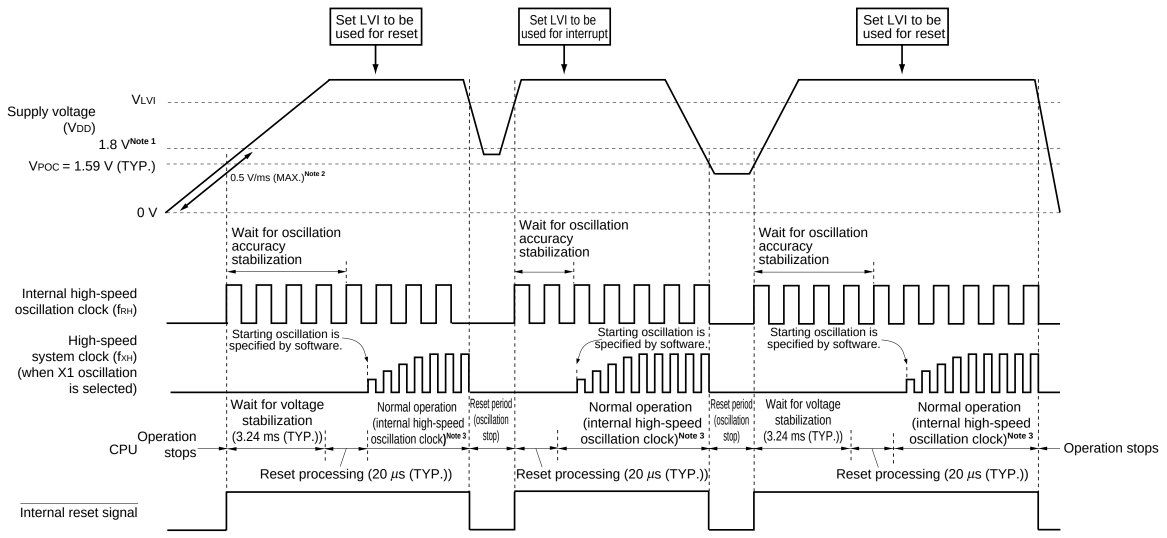
The power-on-clear generates an internal reset at the power-on of VDD. In the 1.59V mode, when the power-supply voltage (VDD) exceeds 1.59 ±0.5V the reset releases the internal reset. In the 2.7/1.59V mode, the internal reset signal releases when VDD = 2.7 ±0.2V. During system operation, if the power-on-clear function detects a drop in VDD below 1.59V, it generates an internal reset.

Figure 3. Power-On-Clear Circuit



The power-on-clear circuit compares the supply voltage (VDD) and the detection voltage (Vpoc) and generates an internal reset when VDD falls below Vpoc. The POC circuit releases the reset when VDD rises above Vpoc.

Figure 4. 1.59V POC Mode Operation


Set LVI to be used for reset
Set LVI to be used for interrupt
Set LVI to be used for reset
Supply voltage (VDD)
VLVI
1.8 V Note 1
VPOC = 1.59 V (TYP.)
0.5 V/ms (MAX.) Note 2
0 V
Wait for oscillation accuracy stabilization
Internal high-speed oscillation clock (fRH)
High-speed system clock (fXH) (when X1 oscillation is selected)
Starting oscillation is specified by software.
Wait for voltage stabilization (3.24 ms (TYP.))
Normal operation (internal high-speed oscillation clock) Note 3
Reset period (oscillation stop)
CPU
Operation stops
Reset processing (20 μs (TYP.))
Operation stops
Internal reset signal

You can select Vpoc at power up. This value is stored in flash memory. For the 78K0/Kx2 family of microcontrollers, location 0081H, bit 0 (POCMODE) specifies Vpoc. If POCMODE is 0, Vpoc is 1.59V; if POCMODE is 1, Vpoc is 2.7V.

In either case, after power up, Vpoc is 1.59V. If VDD drops below this level, power-on clear asserts an internal reset, holding it until VDD rises above 1.59V.

2.1.3 Low-Voltage Detect (LVI)

The low-voltage detect (LVI) function offers the following features:

- ♦ Selectable reset or interrupt operation for low-voltage condition
- ♦ Ability to check VDD or the voltage on an external input pin
- ♦ Selectable VDD from 16 voltage levels

The low-voltage detect circuit compares a voltage against an internal reference and generates an internal interrupt or internal reset when the voltage falls below the reference value. The circuit can compare against VDD, tapping off of a resistor divider to select a specific voltage. The low-voltage detect circuit also supports an external input, EXLVI, generating an interrupt or reset when the voltage on this pin drops below the reference voltage of 1.21V.

Figure 5. Low-Voltage Detector Block Diagram

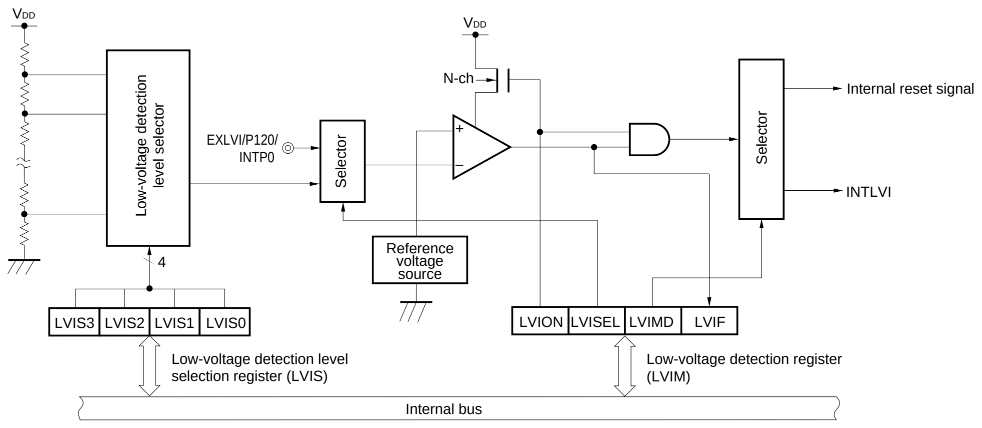


Table 2. Registers Controlling Low-Voltage Detect

Registers	Symbol	Description of Functions
Low-voltage detection register	LVIM	Sets low-voltage detection and operation mode
		Enable/disable low-voltage detection
		Select internal or external input (EXLVI)
		Set low-voltage operation mode and flag
Low-voltage detection level Selection register	LVIS	Selects low-voltage detection level
		Selects up to 16 levels
Port-mode register	PMx	Sets port mode to input or output for EXLVI pin

To configure low-voltage detection when using reset mode:

- ♦ Set the LVIMK bit in the MK0L register to mask the LVI interrupt.
- ♦ Clear the LVIIF interrupt flag in the IF0L register.
- ♦ Disable the LVI detector by clearing the LVION bit in the LVIM register.

- ♦ Select the VDD check or EXLVI check modes with the LVISEL bit in the LVIM register.
- ♦ If using VDD check mode, set the voltage level in the LVIS register.
- ♦ If using EXLVI check mode, use the appropriate PM register to set the EXLVI pin to input.
- ♦ Set the LVIMD bit to 0 to temporarily select interrupt mode.
- ♦ Turn on the LVION bit to enable the detector.
- ♦ Wait a minimum of 10 microseconds for the detector to stabilize.
- ♦ Set the LVIMD bit to 1 to select reset mode.

To configure low-voltage detection when using interrupt mode:

- ♦ Set the LVIMK bit in the MK0L register to mask the LVI interrupt.
- ♦ Clear the LVIIF interrupt flag in the IF0L register.
- ♦ Disable the LVI detector by clearing the LVION bit in the LVIM register.
- ♦ Select the VDD check or EXLVI check modes with the LVISEL bit in the LVIM register.
- ♦ If using VDD check mode, set the voltage level in the LVIS register.
- ♦ If using EXLVI check mode, use the appropriate PM register to set the EXLVI pin to input.
- ♦ Set the LVIMD bit to 0 to select interrupt mode.
- ♦ Set the priority for the LVI interrupt using the PR0L register.
- ♦ Turn on the LVION bit to enable the detector.
- ♦ Wait a minimum of 10 microseconds for the detector to stabilize.
- ♦ Clear the LVIMK bit to 0 to enable the LVI interrupt.

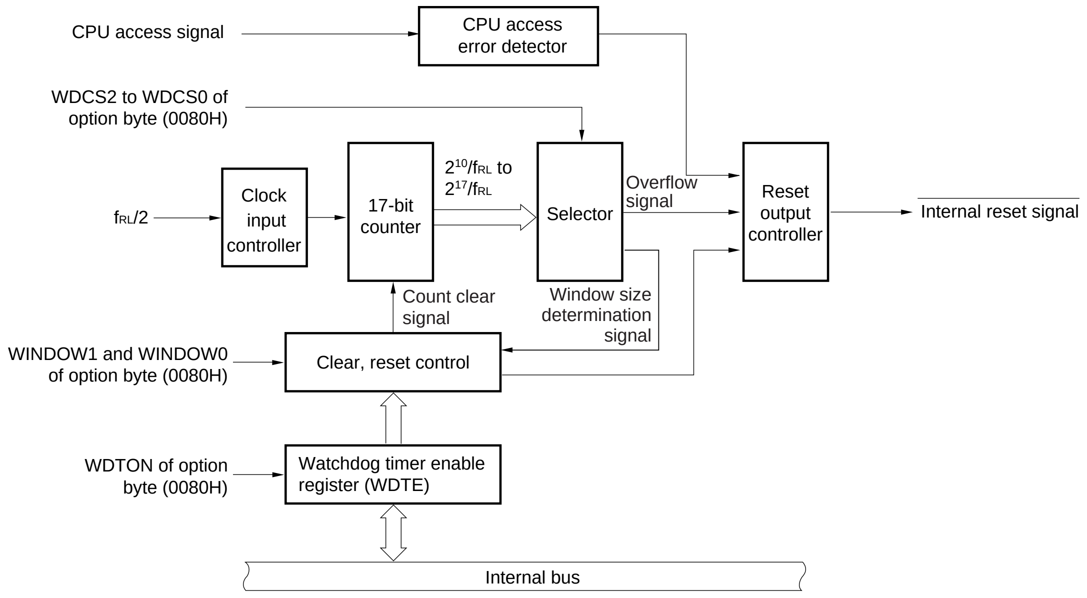
2.1.4 Watchdog Timer (WDT)

The watchdog timer (WDT) offers the following features:

- ♦ Reset of the microcontroller in the event of runaway program loops
- ♦ Operation on internal oscillator, which functions even if the external clock fails
- ♦ Selectable clock frequency to control how often the timer must be cleared
- ♦ Optional protection against stopping the internal oscillator
- ♦ Protection against incorrect values written to clear the watchdog timer
- ♦ Optional protection against clearing the watchdog timer at incorrect times (window open feature)
- ♦ Reset on fetch or read/write of illegal memory areas

The watchdog timer (WDT) operates on the internal low-speed-oscillation clock. When enabled and running, the watchdog timer counter must be cleared periodically or the counter overflows and generates an internal reset. A program clears the counter by writing a special value to the watchdog-timer enable register (WDTE).

Figure 6. Watchdog Timer


CPU access signal
CPU access error detector
WDCS2 to WDCS0 of option byte (0080H)
$f_{RL}/2$
Clock input controller
17-bit counter
$2^{10}/f_{RL}$ to $2^{17}/f_{RL}$
Selector
Overflow signal
Reset output controller
Internal reset signal
Count clear signal
Window size determination signal
WINDOW1 and WINDOW0 of option byte (0080H)
Clear, reset control
WDTON of option byte (0080H)
Watchdog timer enable register (WDTE)
Internal bus

The process used to start the timer works differently in the various NEC Electronics microcontrollers. For some microcontrollers you write to a register, while others require you to set a value in flash memory. Once the watchdog timer is running, you cannot disable it. This feature prevents a runaway program from accidentally disabling the timer. You can also select a mode that prevents disabling the low-speed internal-oscillator, thus preventing programs from stopping watchdog timer’s clock.

Your program must write a specific value to the WDTE register to clear the watchdog-timer counter; in the case of the 78K0/Kx2 family, the value is 0ACH. Writing any other value to WDTE generates a reset, preventing a program loop from writing to WDTE with the wrong value.

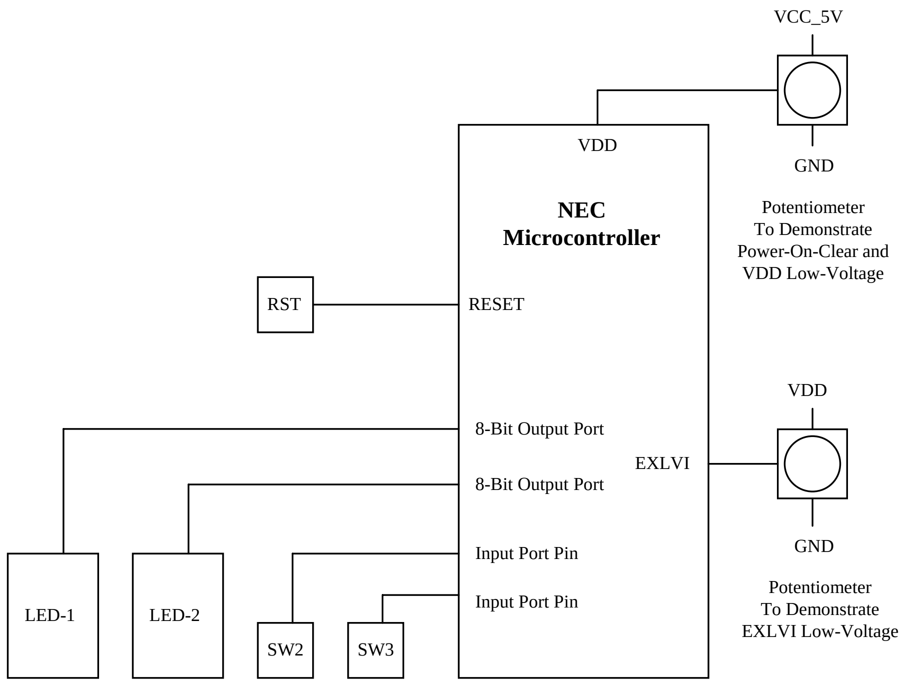
The watchdog-timer window-open period protects against a runaway program writing the correct value (0ACH) to the WDTE register, but doing it at the wrong time. If you set the window open period to 25%, for instance, the program must write 0ACH to the WDTE register after 75% of the overflow time has passed, but before the full overflow time. Using this feature requires correct program structure to ensure that the program writes to the WDTE register at the proper time. You can, however, set the window-open period to 100%, which allows the program to write to the register at any time.

In the 78K0/Kx2 NEC Electronics microcontroller family, you start the watchdog timer with a bit contained in the option byte at 0080H. This option byte also selects the frequency of the watchdog-timer counter and the window-open period.

2.2 Program Description and Specification

The hardware required to demonstrate microcontroller safety features includes an NEC Electronics microcontroller connected to a reset switch, with other switches for input, a two-digit LED display for output, and variable-voltage controls for VDD and EXLVI. This hardware demonstrates reset, power-on-clear, low-voltage detect, and watchdog timer operation. The variable-voltage control for VDD uses the tap of a potentiometer connected to the VDD power supply such that turning the potentiometer varies VDD from 0 to 5V.

Figure 7. Demonstration Hardware



On start up, the demonstration program reads the RESF register and displays the value in the LED display. This value indicates the cause of the previous reset.

To demonstrate external reset, ground the reset input by pressing the RST switch.

To show power-on-clear, adjust the potentiometer to reduce VDD below Vpoc. When power is applied, the device remains in reset, and the LED display remains blank, because it is not driven. Adjusting the potentiometer to raise VDD brings the microcontroller out of reset when VDD exceeds Vpoc. The display then shows the RESF flag value; this value will be 00 for a power-on-clear reset. Reducing VDD causes a power-on-clear when VDD falls below 1.59V. This event blanks the LED display. Turning VDD back above 1.59V releases the power-on-clear reset and displays the RESF value of 00.

To demonstrate the low-voltage detect (LVI) operation, use the same potentiometer. Once the display shows the RESF value, you can select one of several tests by pressing SW2. Then press SW3 to execute that test. To show how the microcontroller checks VDD, the program sets one of two VDD voltage levels, and sets the LVI for either reset or interrupt. Reducing VDD with the potentiometer then causes a reset (blanking the display) or an interrupt (displaying "LI") when the voltage falls below the VDD set point. Raising VDD clears the reset and displays the RESF flag, indicating LVI as the cause of the reset.

Demonstrating EXLVI requires that you connect a separate potentiometer between VDD and GND, with its tap connected to the EXLVI input. To run the demonstration, select an EXLVI input for reset or interrupt mode. Then reducing the EXLVI input below 1.21V (EXLVI threshold) causes a reset (blanking the display) or an interrupt (displaying "LI"). Raising the EXLVI voltage clears the reset and shows the RESF flag, indicating LVI as the cause of the reset.

To demonstrate the watchdog timer, the program sets a timer to periodically interrupt and clear the watchdog timer before the timer's counter overflows. To start this routine, select the watchdog-timer demo. The LED then displays a blinking value; the speed of blinking and the value shown represent the interrupt frequency. Repeated presses on SW3 increase the interval between interrupts. The display increments the value and slows the blinking rate. When the interval exceeds the watchdog timer's overflow time, the watchdog timer resets the program. The RESF value displayed indicates a watchdog-timer reset.

Specifications:

- POC voltage is set to 1.59V mode. The option to use 2.7/1.59V mode is also given.
- LVI is initialized to interrupt when VDD falls below 3.93V.
- The demonstration illustrates LVI reset and interrupt operation.
- You can set an LVI VDD of 3.93 or 2.39V.
- The watchdog timer overflows in 68 milliseconds and has a 100% window-open period.
- Timer H1 provides periodic interrupts to clear the watchdog timer, with a variable interval.
- Timer 00 debounces input switches, considering them stable after 10 milliseconds.

2.3 Software Flow Charts

The demonstration program consists of:

- Initialization code for the program, called before the main() program starts
- The main program loop, which displays reset flags, checks switch status, and runs tests
- Subroutines to demonstrate LVI, EXLVI, and watchdog timer (WDT) features
- Subroutines generated by the Applilet to handle Timer 00 and Timer H1 starting, stopping and changing

- ♦ Subroutines generated by the Applilet for starting, stopping and changing LVI conditions
- ♦ Subroutines generated by the Applilet for watchdog timer clearing
- ♦ Subroutines for handling timer and LVI interrupts (Applilet-generated stub interrupt service routines, with user code added)
- ♦ Subroutines for switch input and LED-display output

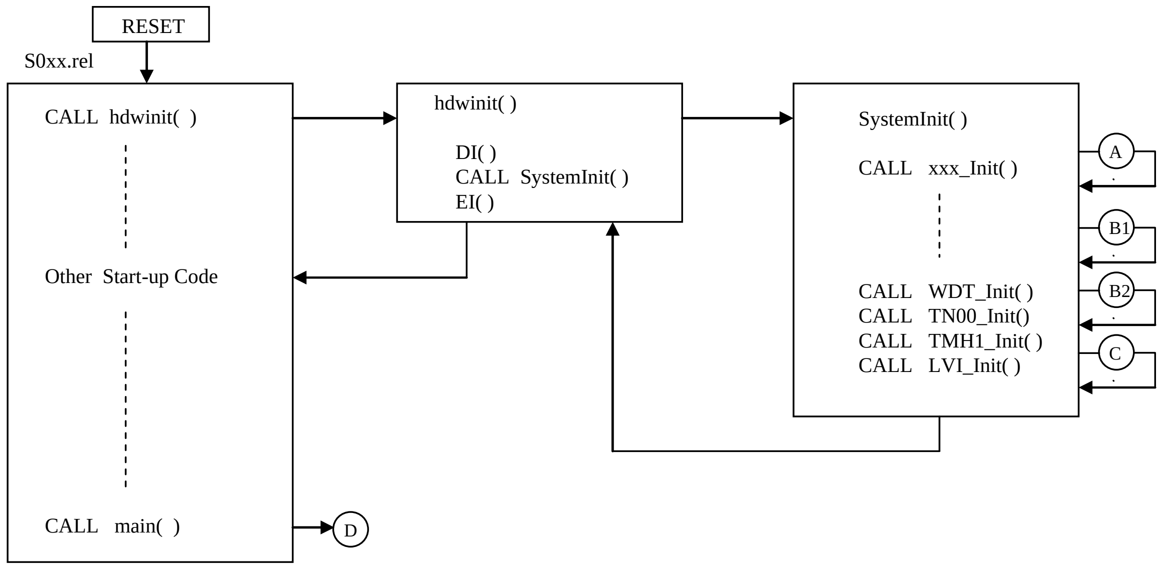
The flowcharts describe the initialization, the main program, demonstration subroutines, and timer and LVI interrupt-service routines. The Appendix provides full listings.

2.3.1 Program Startup and Initialization

For 78K0 programs written in the C language, an object code file such as s0l.rel, linked into the user program, provides the startup code. This startup code calls a function named hdwinit( ); you can place hardware initialization code here.

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

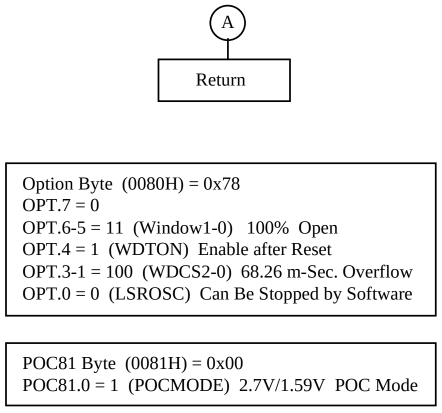
Figure 8. System Startup and Initialization



When the hdwinit( ) function completes, the startup code calls main( ). Thus, the program initializes all peripherals before main( ) starts.

2.3.2 WDT_Init() – Watchdog-Timer Initialization, Option Byte 0080H and 0081H Settings

Figure 9. Initializing Watchdog Timer



Option Byte (0080H) = 0x78
OPT.7 = 0
OPT.6-5 = 11 (Window1-0) 100% Open
OPT.4 = 1 (WDTON) Enable after Reset
OPT.3-1 = 100 (WDCS2-0) 68.26 m-Sec. Overflow
OPT.0 = 0 (LSROSC) Can Be Stopped by Software

POC81 Byte (0081H) = 0x00
POC81.0 = 1 (POCMODE) 2.7V/1.59V POC Mode

SystemInit() calls WDT_Init() as part of the hardware initialization process. This routine contains only a return.

In the NEC Electronics microcontroller used for this demonstration, an option byte controls the initial settings and startup for the watchdog timer. This option byte is located in flash memory at location 0080H. In some NEC Electronics microcontrollers, however, a register controls those settings, and for those microcontrollers the Applilet puts the code to control the register in the WDT_Init() routine.

Bit 4 of the option byte, WDTON, controls whether the watchdog timer runs initially or only after releasing the reset. For this demonstration, the routine sets this bit to 1, allowing the timer to run.

Bits 6 and 5, WINDOW1 and WINDOW0, control the window-open time. The routine sets both of these bits to 1, for a 100% window-open time, removing any restrictions on when the program can write to the watchdog-timer register to clear it.

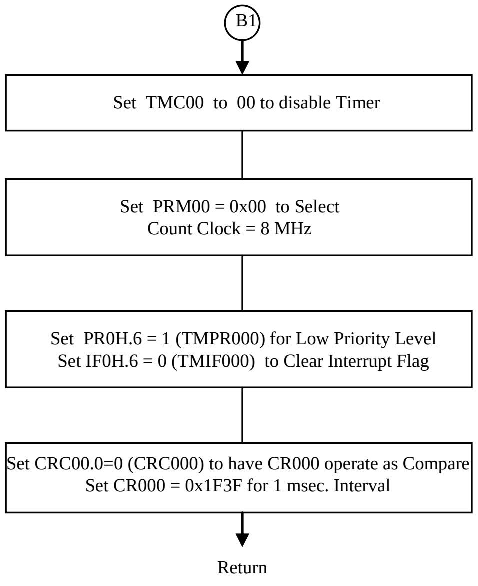
Bits 3, 2, and 1, WDCS2 through WDCS0, control the watchdog-timer overflow time. These bits select a different number of counts of the low-speed internal-oscillator for overflow. The routine sets these bits to a value of 100, which selects a clock division of 16,384/f_{RL}. Typically, the internal low-speed oscillator runs at 240 kHz, giving a watchdog-timer overflow time of 16,384/240,000 seconds, or 68.27 milliseconds. Once you release the reset, the program must write 0xAC to the watchdog-timer register, WDTE, within this 68.27-millisecond period or the microcontroller resets.

Bit 0, LSROSC, controls whether software can stop the internal low-speed oscillator. Setting the bit to 0 allows the program to stop the oscillator. The program stops the oscillator by setting the LSRSTOP bit in the RCM register. This bit prevents the watchdog timer from counting, even when the microcontroller is in HALT or STOP modes, regardless of whether the internal low-speed oscillator is stopped or not. Setting the LSROSC bit to 1 allows the internal low-speed oscillator to keep running, even if the LSRSTOP bit is set. You can use this setting to ensure that the watchdog timer always runs. The demonstration program sets LSROSC to 0, allowing the program to stop the internal low-speed oscillator.

The POC option byte, located at 0081H, controls the power-on clear behavior. Bit 0 of this byte, POCMODE, controls the power-on clear operating mode. Setting this bit to 0 selects the 1.59V POC mode so that the internal reset releases when VDD reaches 1.59V. Setting the bit to 1 selects the 2.7/1.59V mode; on initial power-up the microcontroller does not release the internal reset until VDD reaches 2.7V. In either mode, the microcontroller asserts an internal reset whenever VDD falls below 1.59V, releasing the interrupt once VDD rises above this level. The demonstration uses the 1.59V mode.

2.3.3 TM00_Init() – Timer 00 Initialization for 1-Millisecond Periodic Interrupt

Figure 10. Flowchart for 1-Millisecond Periodic Interrupt


B1
Set TMC00 to 00 to disable Timer
Set PRM00 = 0x00 to Select Count Clock = 8 MHz
Set PR0H.6 = 1 (TMPR000) for Low Priority Level
Set IF0H.6 = 0 (TMIF000) to Clear Interrupt Flag
Set CRC00.0=0 (CRC000) to have CR000 operate as Compare
Set CR000 = 0x1F3F for 1 msec. Interval
Return

SystemInit() calls the TM00_Init() routine to set the 16-bit Timer 00 (TM00) to its interval-timer mode and to cause an interrupt every 1 millisecond. The demonstration program uses this interrupt to debounce switches.

First TM00_Init() disables the timer while changing register settings.

The routine sets the prescaler-mode register, PRM00, which controls the timer clock. The PRM00 register is set to 0x00, instructing Timer 00 to use f_{PRS}, the main peripheral clock, as its count clock. In this demonstration, f_{PRS} uses the 8-MHz internal high-speed oscillator as its source. Thus, the count register (TM00) increments once every 1/8,000,000 seconds (0.125 microseconds).

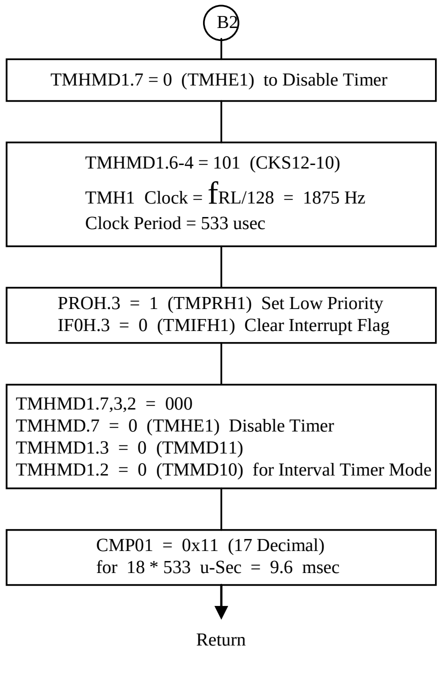
TM00_Init() sets the registers related to the INTTM000 interrupt to low priority and clears the interrupt flag. The routine leaves the mask register controlling the timer-interrupt enable in its default disabled state.

NEC Electronics microcontroller interrupts default to low priority, although you can set them to high priority. Within each of these priority classifications, each interrupt source has a priority relative to other interrupts. In this case, INTTM000 has a lower priority than either INTTMH1 (Timer H1 interrupt) or INTLVI (low-voltage interrupt). The demonstration uses low priority for all interrupts, thus servicing INTTM000 after either INTTMH1 or INTLVI, if multiple interrupts occur simultaneously.

TM00_Init() sets the CRC00 register to use CR000 as a compare register and provides a compare value of 0x1F3F (7999 decimal). This value causes CR000 to match TM00 every (7999 + 1) counts, which occurs once every 8000 * 0.125 microseconds = 1,000 microseconds = 1 millisecond. The Applilet calculates the compare value to provide the 1 millisecond interval.

2.3.4 TMH1_Init() – Timer H1 Initialization for 10-Millisecond Periodic Interrupt

Figure 11. Flowchart for 10-Millisecond Periodic Interrupt


B2
TMHMD1.7 = 0 (TMHE1) to Disable Timer
TMHMD1.6-4 = 101 (CKS12-10)
TMH1 Clock = fRL/128 = 1875 Hz
Clock Period = 533 usec
PROH.3 = 1 (TMPRH1) Set Low Priority
IF0H.3 = 0 (TMIFH1) Clear Interrupt Flag
TMHMD1.7,3,2 = 000
TMHMD.7 = 0 (TMHE1) Disable Timer
TMHMD1.3 = 0 (TMMD11)
TMHMD1.2 = 0 (TMMD10) for Interval Timer Mode
CMP01 = 0x11 (17 Decimal)
for 18 * 533 u-Sec = 9.6 msec
Return

SystemInit() calls TMH1_Init() to set the 8-bit Timer H1 (TMH1) for interval-timer operation with an interval of about 10 milliseconds. This timer generates a periodic interrupt to reset the watchdog timer before the timer overflows.

First TMH1_Init() disables the timer, setting the TMHE1 enable bit in the TMHMD1 register to 0, while making other settings.

Bits 6, 5, and 4 of TMHMD1—the CKS12, CKS11, and CKS10 bits—control the clock source for TMH1. TMH1_Init() sets these bits to 101 to select f_{RL}/128. Since f_{RL} is the internal low-speed oscillator, the clock frequency is 240,000/128 or 1875 Hz. Each count takes about 533 microseconds.

Because the internal low-speed oscillator is the clock source for both the watchdog timer and TMH1, variations in its frequency do not affect the relative rates of the two timers.

TMH1_Init() sets the TMPRH1 bit (PR0L.3) to 1 to select low priority for the INTTMH1 interrupt. The routine also clears the TMIFH1 interrupt flag (IF0L.3).

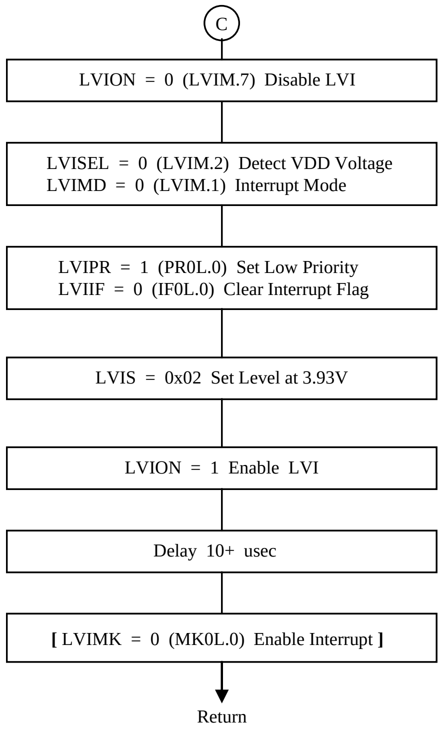
As described earlier, you can assign interrupts a high priority or let them default to low priority. In addition, each interrupt source has a priority relative to the others. For example, INTTMH1 has a higher priority than INTTM000 (Timer 00 interrupt), but a lower priority than INTLVI (low-voltage interrupt). If you set the interrupts within the same priority group, this order prevails. The demonstration uses all low-priority interrupts, so the microcontroller services INTTMH1 before INTTM000, but after INTLVI, if multiple interrupts occur simultaneously.

Bits 3 and 2 of TMHMD1 (the TMMD11 and TMMD10 bits) control the timer-operation mode. TMH1_Init() sets these bits to 00 for interval-timer operation, which triggers the INTTMH1 interrupt when the TMH1 timer register matches the CMP01 compare register.

The initialization routine sets the CMP01 compare register to 0x11 (17 decimal). This setting results in an INTTMH1 interrupt every (17 + 1) * 533 microseconds = 9.6 milliseconds. Because the initialization leaves the TMHE1 enable bit at 0, the interrupt leaves timer TMH1 stopped. Calling TMH1_Start() starts the timer again.

2.3.5 LVI_Init() – Initializing Low-Voltage Detection

Figure 12. Flowchart for Initializing Low-Voltage Detection


C
LVION = 0 (LVIM.7) Disable LVI
LVISEL = 0 (LVIM.2) Detect VDD Voltage
LVIMD = 0 (LVIM.1) Interrupt Mode
LVIPR = 1 (PR0L.0) Set Low Priority
LVIIF = 0 (IF0L.0) Clear Interrupt Flag
LVIS = 0x02 Set Level at 3.93V
LVION = 1 Enable LVI
Delay 10+ usec
[LVIMK = 0 (MK0L.0) Enable Interrupt]
Return

SystemInit() calls LVI_Init() to set the operating mode of the low-voltage detection circuit, initially setting the circuit to interrupt if VDD falls below 3.93V.

First, LVI_Init() clears the LVION bit (LVIM.7) to 0, disabling LVI operation while the routine changes other settings.

LVISEL (LVIM.2) selects among voltage-comparison sources. The initialization program clears this bit to 0, which sets up a comparison of a particular tap of a VDD resistor divider with the reference voltage. The LVIS register selects the specific tap. Setting the LVISEL bit to 1 compares the EXLVI pin voltage with the 1.21V reference, triggering the detector if EXLVI falls below 1.21V. The demonstration program changes LVISEL to show both VDD and EXLVI triggering.

LVIMD (LVIM.1) controls the LVI operating mode. LVI_Init() clears LVIMD (LVIM.1) to 0. This setting selects interrupt operation, so that the LVI detector initiates the INTLVI interrupt. The program sets the LVIMD bit to 1 for reset mode, so that the LVI trigger resets the microcontroller. The demonstration program changes the setting of LVIMD to show both interrupt and reset modes.

LVI_Init() sets the LVIPR bit (PR0L.0) to 1, selecting low priority, and clears the LVIIF interrupt flag (IF0L.0).

As mentioned earlier, NEC Electronics microcontrollers allow you to assign either high or low (default) priority to each interrupt. Within these priority groups, each interrupt source has a relative priority. INTLVI has a higher priority than any other interrupt in its group. The demonstration uses low priority for all interrupts, thus giving INTLVI precedence over any other interrupt.

During the servicing of an interrupt, the interrupt-service routine normally clears the master-interrupt enable bit IE (PSW..7). However, this bit must be set for another interrupt (of a higher priority) to occur during the current interrupt-service routine. Thus, in the demonstration program, the interrupt-service routines for the lower-priority interrupts (INTTM000 for Timer 00 and INTTMH1 for the Time H1) set the IE bit to allow LVI interrupts to occur.

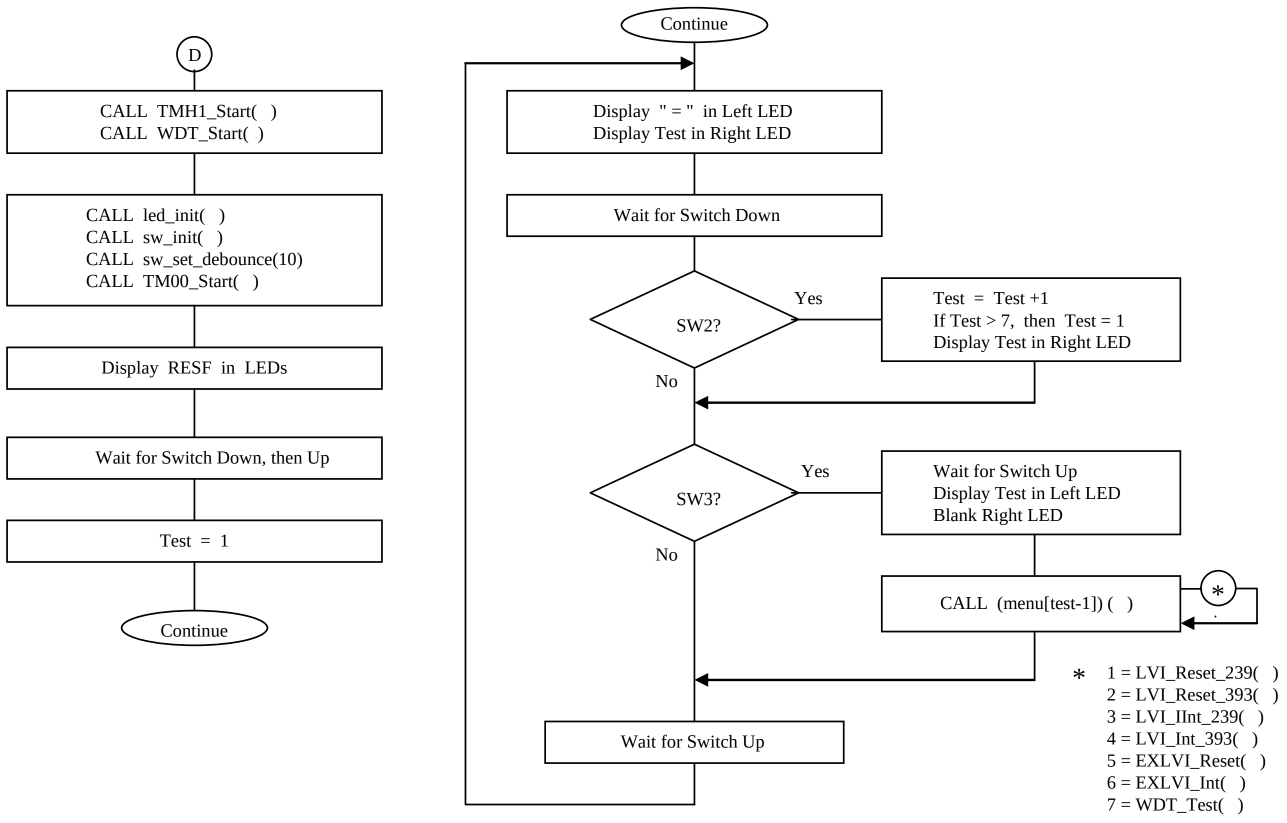
LVI_Init() sets the LVIS register to 0x02. This setting selects the 3.93V tap on the VDD resistor divider.

LVI_Init() sets the LVION bit (LVIM.7) to 1 to enable LVI operation. Because the LVI takes up to 10 microseconds to stabilize, the routine delays a minimum of 10 microseconds after setting LVION. If this delay takes longer than 69 milliseconds, the watchdog timer overflows, causing a reset. If there is the potential for a delay longer than 69 milliseconds in your application, insert a call to WDT_Restart().

After the 10-microsecond delay, LVI_Init() normally enables the LVI interrupt by clearing LVIMK (MK0L.0) to 0. The demonstration program comments this instruction out to keep the LVI interrupt disabled until it is enabled by one of the test routines.

2.3.6 Main() – Main Program – Safety Features Demonstration

Figure 13. Flowchart for Demonstrating Safety Features



Once the startup code completes hardware and software initialization, SystemInit( ) calls the main() routine. The main() routine calls TMH1_Start() to start the TMH1 timer and then calls WDT_Start() to clear the watchdog-timer overflow counter.

After this point, periodic INTTMH1 interrupts occur every 9.6 milliseconds. The MD_INTTMH1() interrupt-service routine calls WDT_Restart() to clear the watchdog-timer overflow counter. As long as the INTTMH1 interrupt occurs at this frequency, the watchdog timer does not reset the program.

In the case of the microcontroller used for the demonstration, the watchdog timer starts after reset by the option byte, so WDT_Start() just writes 0xAC to the WDTE register to clear the watchdog-timer overflow—the same action performed by WDT_Restart(). In the case of microcontrollers with watchdog timers whose start is register controlled, WDT_Start() actually starts the timer.

Note that the watchdog timer runs from the reset and is cleared for the first time when main() calls WDT_Start(). If the program startup takes more than 68 milliseconds, then the timer resets the microcontroller before the call to WDT_Restart(), and the program will never reach main(). If this is likely to be the case in your application, modify the startup code to insert instructions that reset the watchdog timer at strategic intervals. An alternative would be to stop the internal low-speed oscillator early in the startup code (if this is allowed by the option byte) and restart the oscillator once main() is reached.

After setting up to clear the watchdog timer periodically, the program initializes the LED display and switches, then sets the switch-debounce counter to 10 and calls TM00_Start() to start the TM00 timer. The TM00 timer now triggers an INTTM000 interrupt every millisecond, causing the MD_INTTM000() interrupt-service routine to check whether the switches are debounced.

Main() reads the RESF reset-flag register and displays the value stored there on the LED display as a hexadecimal number. The RESF register in the demonstration microcontroller uses two bits to indicate the reset type.

If a watchdog-timer counter overflow occurs, or another operation causes the timer to assert reset (such as writing a value to WDTE which is not 0xAC), the program sets RESF bit 4 (WDTRF), causing the LED display to show 10.

If the LVI detector causes a reset due to VDD falling below the set threshold level (in the VDD check mode), or a voltage on EXLVI falling below 1.21V (in the EXLVI check mode), the program sets RESF bit 0 (LVIRF) resulting in 01 on the LED display.

In the event of a low-going reset-input pin (indicating that you pushed the reset switch), or if the power-on-clear circuit causes a reset, the program clears both of the above bits (so RESF will be 0x00), and the LED display shows 00.

After displaying the RESF value, main() waits for you to press and release a switch. The main() routine then sets the **test** variable to 1, and then enters a test-selection loop. The left digit of the LED display shows “ = ”, and the right digit shows the current value of **test**. At startup, the display therefore shows “ =1”. Main() loops until you press a switch (INTTM000 and INTTMH1 interrupts occur periodically).

Once main() detects a debounced switch value, the routine tests to see which switch has been pressed. SW2 causes the routine to increment the **test** variable, whose value wraps back to 1 if it exceeds 7. Holding SW2 down therefore causes the display to increment **test** continuously. The display shows “=1”, “=2”, “=3”, up to “=7” and then back to “=1”.

If you press SW3, main() first waits for the switch to be up, then displays the value of **test** on the left LED, blanks the right LED, and calls a subroutine. Main() gets the address of the subroutine from the menu[ ] array, using **test – 1** as an index into the array. So if **test** is 1, main() calls the subroutine whose address is in **menu[0]** (in this case the LVI_Reset_239() routine).

This configuration allows you to run seven different tests. Table 3 shows the value of **test**, the LED display when you press SW3, the menu-array function accessed, and a description of each test (function operation).

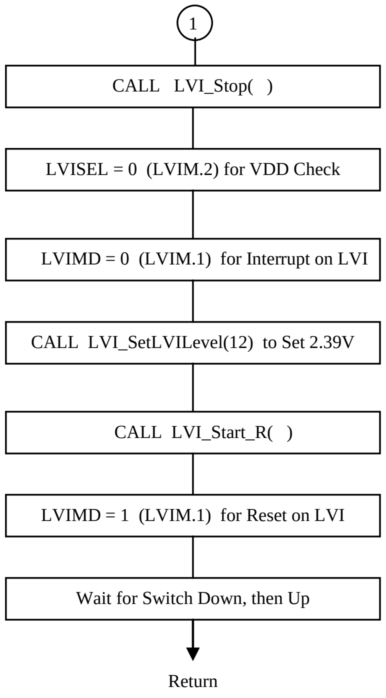
Table 3. Test Descriptions

Test	LED	menu[test - 1]	Function Operation
1	“1 ”	LVI_Reset_239()	Set LVI to reset on VDD < 2.39V
2	“2 ”	LVI_Reset_393()	Set LVI to reset on VDD < 3.93V
3	“3 ”	LVI_Int_239()	Set LVI to interrupt on VDD < 2.39V
4	“4 ”	LVI_int_393()	Set LVI to interrupt on VDD < 3.93V
5	“5 ”	EXLVI_Reset()	Set LVI to reset on EXLVI < 1.21V
6	“6 ”	EXLVI_Int()	Set LVI to interrupt on EXLVI < 1.21V
7	“7 ”	WDT_Demo()	Show reset on WDT counter overflow

When the selected test returns, main() waits until all switches are open, and then goes to the top of the loop to redisplay the equal sign and test number, and wait for the next switch closure.

2.3.7 LVI_Reset_239() – Set LVI for Reset at VDD Below 2.39V

Figure 14. Flowchart for LVI Reset at VDD Below 2.39V


1
CALL LVI_Stop( )
LVISEL = 0 (LVIM.2) for VDD Check
LVIMD = 0 (LVIM.1) for Interrupt on LVI
CALL LVI_SetLVILevel(12) to Set 2.39V
CALL LVI_Start_R( )
LVIMD = 1 (LVIM.1) for Reset on LVI
Wait for Switch Down, then Up
Return

When you select Test =1, main() calls LVI_Reset_239(). This routine sets the LVI unit to reset the microcontroller if VDD falls below 2.39V. This setting changes the default LVI behavior set in LVI_Init(), or by other LVI test routines.

LVI_Reset_239() calls LVI_Stop() to clear the LVION bit to 0, disabling the LVI detector. LVI_Stop() also sets LVIMK to 1 to disable the LVI interrupt.

LVI_Reset_239() sets the LVISEL bit (LVIM.2) to 0, selecting the LVI detector’s VDD-comparison mode. LVI_Reset_239() sets the LVIMD bit (LVIM.1) to 0, temporarily selecting the interrupt mode. The routine then calls LVI_SetLVILevel(12), which sets the LVIS register to 0x0C. This value sets the tap on the VDD resistor divider to trigger the LVI detector if VDD falls below 2.39V.

The routine then calls LVI_Start_R(), which sets LVIMK, clears LVIIF, and sets LVION to 1 to enable the LVI detector. The routine waits at least 10 microseconds, and then returns without clearing the LVIMK bit to enable the interrupt. This routine is a modified version of LVI_Start(), which sets up LVI to operate in reset mode. When LVI_Start_R() returns, the LVI detector is stable.

LVI_Reset_239() sets the LVIMD bit (LVIM.1) to 1, selecting the LVI-detector reset mode. In this mode, the setting of LVIMK has no effect on LVI operation. If VDD falls below the selected voltage, the low voltage triggers an immediate reset.

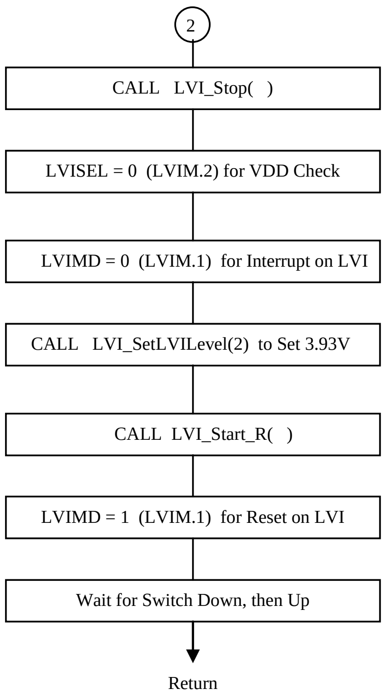
If there is no reset (VDD is above the selected voltage), the test routine waits for a switch to be pressed and released before returning to the main() program.

A reset occurs if you reduce VDD below the threshold while the routine is waiting for a switch (or later in main() when in this same LVI mode).

On reset, the program restarts, reinitializing the LVI detector in interrupt mode (generating an interrupt when VDD falls below 3.93V). The LED display shows 01 as the contents of the reset flag register RESF, indicating that the LVI detector caused the reset.

2.3.8 LVI_Reset_393() – Set LVI for Reset at VDD Below 3.93V

Figure 15. Flowchart for Setting LVI to Reset at VDD Below 3.93V


2
CALL LVI_Stop( )
LVISEL = 0 (LVIM.2) for VDD Check
LVIMD = 0 (LVIM.1) for Interrupt on LVI
CALL LVI_SetLVILevel(2) to Set 3.93V
CALL LVI_Start_R( )
LVIMD = 1 (LVIM.1) for Reset on LVI
Wait for Switch Down, then Up
Return

When Test =2, main() calls LVI_Reset_393(), which sets the LVI unit to reset the microcontroller if the VDD voltage falls below 3.93V. This setting changes the default LVI behavior set in LVI_Init() or by other LVI test routines.

LVI_Reset_393() calls LVI_Stop() to clear the LVION bit to 0, disabling the LVI detector, and to set LVIMK to 1 to disable the LVI interrupt.

LVI_Reset_393() sets the LVISEL bit (LVIM.2) to 0, selecting the VDD-comparison mode. The routine also sets the LVIMD bit (LVIM.1) to 0, temporarily selecting the LVI detector's interrupt mode.

The routine calls LVI_SetLVILevel(2), which sets the LVIS register to 0x02. This value sets the tap on the VDD resistor divider to trigger the LVI detector if VDD falls below 3.93V.

The routine then calls LVI_Start_R(), which sets LVIMK, clears LVIIF, sets LVION to 1 to enable the LVI detector, waits at least 10 microseconds, and then returns without clearing the LVIMK bit to enable the interrupt. This routine is a modified version of LVI_Start(), which sets the reset mode. When LVI_Start_R() returns, the LVI detector is stable.

LVI_Reset_393() sets the LVIMD bit (LVIM.1) to 1, selecting LVI detector reset mode. In reset mode, the setting of LVIMK has no effect on LVI operation; LVI resets immediately if VDD falls below the selected voltage.

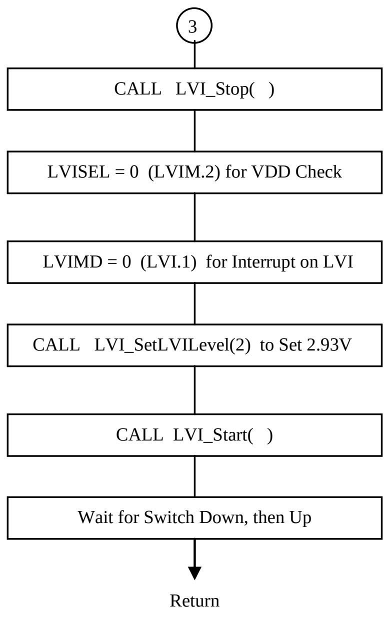
If there is no reset (VDD remains above the selected voltage), the test routine waits for you to press and release a switch before returning to the main() program.

A reset occurs if VDD drops below the threshold while the routine waits for a switch (or later in main() if the LVI is operating in this mode).

On reset, the program restarts, reinitializing the LVI detector to interrupt when VDD falls below 3.93V. The LED display shows 01 as the contents of the reset-flag register RESF, indicating that the LVI detector caused the reset.

2.3.9 LVI_Int_239() – Set LVI for Interrupt at VDD Below 2.39V

Figure 16. Flowchart for Setting Interrupt When VDD Falls Below 2.39V


3
CALL LVI_Stop( )
LVISEL = 0 (LVIM.2) for VDD Check
LVIMD = 0 (LVI.1) for Interrupt on LVI
CALL LVI_SetLVILevel(2) to Set 2.93V
CALL LVI_Start( )
Wait for Switch Down, then Up
Return

When Test = 3, main() calls LVI_Int_239(). This routine sets the LVI to interrupt the microcontroller if VDD falls below 2.39V. This setting changes the default LVI behavior set in LVI_Init() or by other LVI test routines.

LVI_Int_239() calls LVI_Stop() to clear the LVION bit to 0, disabling the LVI detector, and to set LVIMK to 1 to disable the LVI interrupt. The routine then clears the LVISEL bit (LVIM.2) to 0, selecting the LVI detector's VDD-comparison mode.

LVI_Int_239() sets the LVIMD bit (LVIM.1) to 0, selecting the interrupt mode.

The routine calls LVI_SetLVILevel(12), which sets the LVIS register to 0x0C. This value sets the tap on the VDD resistor divider to trigger the LVI detector if VDD falls below 2.39V.

The routine then calls LVI_Start(), which sets LVIMK, clears LVIIF, sets LVION to 1 (to enable the LVI detector), waits at least 10 microseconds, and then clears the LVIMK bit to enable the interrupt.

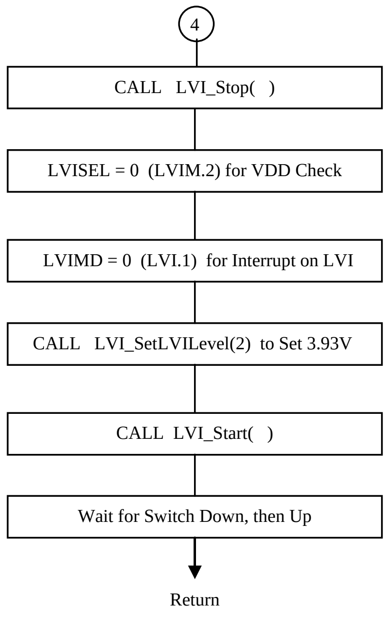
If VDD is below the selected voltage, clearing the LVIMK flag immediately triggers an INTLVI interrupt. If there is no INTLVI interrupt (VDD is above the selected voltage), the test routine waits for you to press and release a switch before returning to the main() program.

An INTLVI interrupt occurs if VDD decreases below the threshold while the routine waits for a switch (or later in main() if operating in this LVI mode).

On INTLVI, the MD_INTLVI() interrupt-service routine displays LI on the LEDs.

2.3.10 LVI_Int_393() – Set LVI for Interrupt at VDD Below 3.93V

Figure 17. Flowchart to Set LVI for Interrupt When VDD is Below 3.93V


4
CALL LVI_Stop( )
LVISEL = 0 (LVIM.2) for VDD Check
LVIMD = 0 (LVI.1) for Interrupt on LVI
CALL LVI_SetLVILevel(2) to Set 3.93V
CALL LVI_Start( )
Wait for Switch Down, then Up
Return

When Test=4, main() calls LVI_Int_393() to set the LVI unit to interrupt the microcontroller if VDD falls below 3.93V. Although this LVI behavior is the default set by LVI_Init(), other test routines might have changed this setting.

LVI_Int_393() calls LVI_Stop() to clear the LVION bit to 0, disabling the LVI detector, and to set LVIMK to 1 to disable the LVI interrupt.

The routine clears the LVISEL bit (LVIM.2) to 0, selecting the LVI detector's VDD comparison mode. The routine also sets the LVIMD bit (LVIM.1) to 0, selecting interrupt mode.

The routine calls LVI_SetLVILevel(2), which sets the LVIS register to 0x02. This value selects the tap on the VDD resistor divider to trigger the LVI detector if VDD falls below 3.93V.

The routine then calls LVI_Start(), which sets LVIMK, clears LVIIF, sets LVION to 1 to enable the LVI detector, waits at least 10 microseconds, and then clears the LVIMK bit to enable the interrupt.

If VDD is below the selected voltage, an INTLVI interrupt occurs immediately after clearing the LVIMK flag.

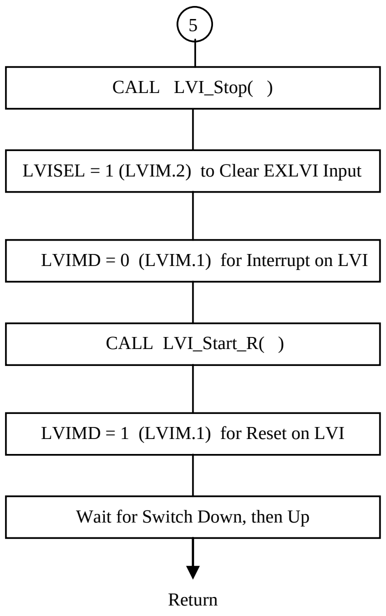
If an INTLVI interrupt does not occur (VDD is above the selected voltage), the routine waits for you to press and release a switch before returning to the main() program.

An INTLVI interrupt occurs if VDD drops below the threshold while the routine is waiting for a switch (or later in main() if operating in the same LVI mode).

On INTLVI, the MD_INTLVI() interrupt-service routine executes, showing LI in the LED display.

2.3.11 EXLVI_Reset() – Set LVI for Reset on EXLVI Low Voltage

Figure 18. Flowchart for Rest on EXLVI Low Voltage


5
CALL LVI_Stop( )
LVISEL = 1 (LVIM.2) to Clear EXLVI Input
LVIMD = 0 (LVIM.1) for Interrupt on LVI
CALL LVI_Start_R( )
LVIMD = 1 (LVIM.1) for Reset on LVI
Wait for Switch Down, then Up
Return

When Test=5, the main() routine calls EXLVI_Reset() to configure the LVI unit to reset the microcontroller if the EXLVI input voltage falls below 1.21V. This setting changes the default LVI behavior set in LVI_Init() or by other LVI test routines.

EXLVI_Reset() calls LVI_Stop() to clear the LVION bit to 0, disabling the LVI detector, and to set LVIMK to 1, disabling the LVI interrupt.

EXLVI_Reset() sets the LVISEL bit (LVIM.2) to 1, selecting the EXLVI comparison mode, and sets the LVIMD bit (LVIM.1) to 0, temporarily selecting the interrupt mode.

The routine then calls LVI_Start_R(), which sets LVIMK, clears LVIIF, sets LVION to 1 to enable the LVI detector, waits at least 10 microseconds, and then returns without clearing the LVIMK bit to enable the interrupt. This modified version of LVI_Start() establishes the reset mode. When LVI_Start_R() returns, the LVI detector is stable.

EXLVI_Reset() sets the LVIMD bit (LVIM.1) to 1, selecting the reset mode. In this mode, the setting of LVIMK has no effect on LVI operation. If EXLVI falls below 1.21V, the low voltage immediately resets the microcontroller.

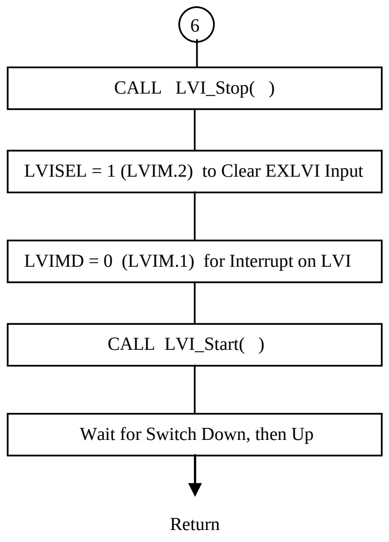
If a reset does not occur (EXLVI is above 1.21V), the test routine waits for you to press and release a switch, then returns to the main() program.

A reset occurs if EXLVI starts above 1.21V and decreases below this voltage while the routine is waiting for a switch (or later in main() if in the same LVI mode).

On reset, the program restarts, reinitializing the LVI detector to interrupt on VDD falling below 3.93V. The LED display shows 01 as the contents of the reset flag register RESF, indicating that the LVI detector caused the reset.

2.3.12 EXLVI_Int() – Set LVI for Interrupt on EXLVI Low Voltage

Figure 19. Flowchart for Setting LVI to Interrupt on EXLVI Low Voltage


6
CALL LVI_Stop( )
LVISEL = 1 (LVIM.2) to Clear EXLVI Input
LVIMD = 0 (LVIM.1) for Interrupt on LVI
CALL LVI_Start( )
Wait for Switch Down, then Up
Return

When Test =6, main() calls EXLVI_Int() to set the LVI unit to interrupt the microcontroller if the EXLVI input voltage falls below 1.21V. This setting changes the default LVI behavior set in LVI_Init() or by other LVI test routines.

EXLVI_Int() calls LVI_Stop() to clear the LVION bit to 0, disabling the LVI detector, and to set LVIMK to 1, disabling the LVI interrupt.

EXLVI_Int() sets the LVISEL bit (LVIM.2) to 1, selecting the EXLVI comparison mode. The routine also sets the LVIMD bit (LVIM.1) to 0, selecting interrupt mode.

The routine then calls LVI_Start(), which sets LVIMK, clears LVIIF, sets LVION to 1 to enable the LVI detector, waits at least 10 microseconds, and then clears the LVIMK bit to enable the interrupt.

If EXLVI is below 1.21V, an INTLVI interrupt occurs immediately after clearing the LVIMK flag.

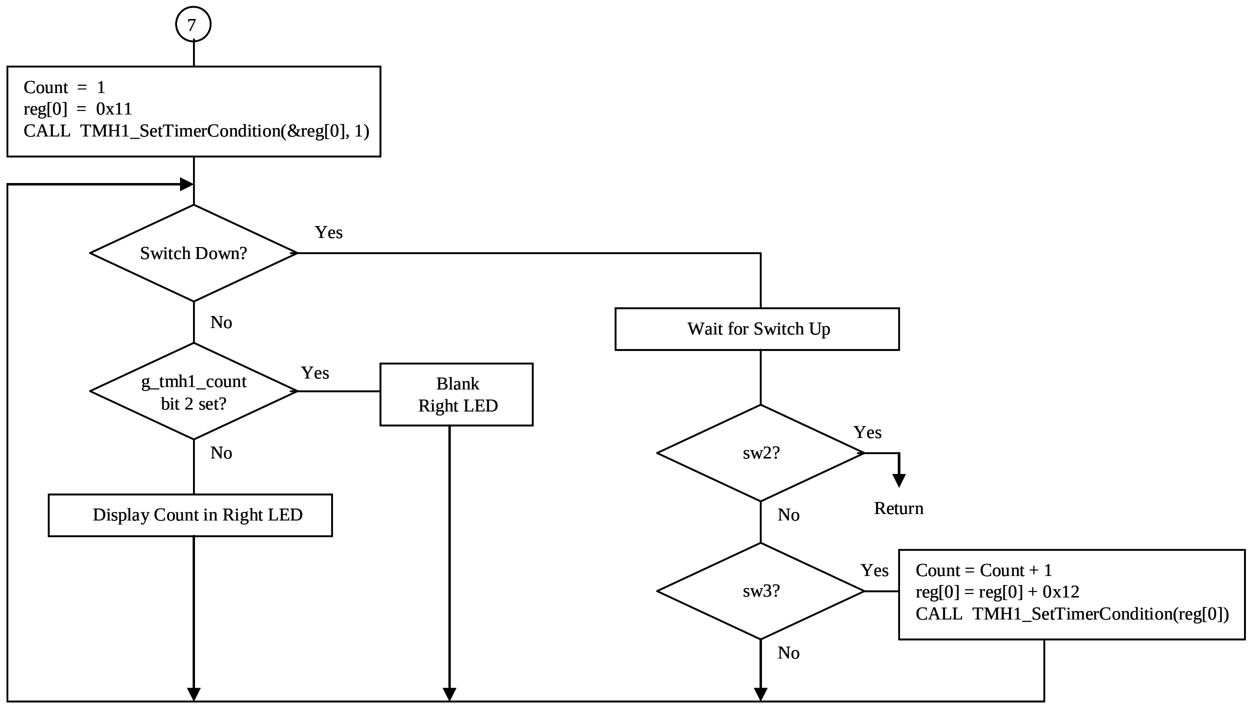
If an INTLVI interrupt does not occur (EXLVI is above 1.21V), the test routine waits for you to press and release a switch before returning to the main() program.

An INTLVI interrupt occurs if EXLVI is above 1.21V and drops below this voltage while the routine is waiting for a switch (or later in main() if in the same LVI mode).

On INTLVI, the MD_INTLVI() interrupt-service routine executes, showing LI in the LED display.

2.3.13 WDT_Demo() – Demonstrating Watchdog-Timer Overflow

Figure 20. Demonstrating Watchdog-Timer Overflow



When Test =7, main() calls WDT_Demo() to illustrate what happens when you do not clear the watchdog timer before it overflows — in this case, within 68 milliseconds.

The routine sets the **count** variable to 1. The routine also sets the interval for Timer H1 to 0x11 by placing this value in the **reg[0]** variable and calling TMH1_ChangeTimerCondition(). This change-timer-condition routine sets the value of the CMP01 register. When timer TMH1 counts up to the CMP01 value, the timer triggers an INTTMH1 interrupt. The MD_INTTMH1() interrupt-service routine calls WDT_Restart() to clear the watchdog timer. TMH1 also clears when the match occurs.

With CMP01 set to 0x11 (17 decimal), the interval between TMH1 and CMP01 matches is 18 times the TMH1 count-clock period. The count clock is 533.33 microseconds, so 18 * 533.33 = 9.6 milliseconds. When the routine starts, it clears the watchdog timer every 9.6 milliseconds, so the timer does not overflow. The MD_INTTMH1() interrupt-service routine increments the global variable **g_tmh1_count** every time the INTTMH1 interrupt occurs; this variable counts up at a frequency determined by the TMH1 interval.

The WDT_Demo() routine then enters a loop and waits for you to press a switch. If the routine does not see a switch pressed, the routine checks the global variable **g_tmh1_count**. Depending on the state of bit 2, the routine either blanks the right digit or displays the **count** variable. Since the interrupt increments **g_tmh1_count**, the right LED digit blinks at a rate proportional to the rate of INTTMH1 interrupts.

At the start of the WDT_Demo() routine, the left LED displays 7, as set by main(). Thus, when the demonstration runs, you see 71 on the display, with the 1 blinking on and off rapidly.

If you press SW2, WDT_Demo() returns to main(). If you press SW3, **count** increments, 0x12 (18 decimal) is added to **reg[0]**, and WDT_Demo() calls TMH1_ChangeTimerCondition() to change the TMH1 interval to the new value. Thus, every press of SW3 increases **count** (shown as 2, 3, 4, etc.) and changes the interval for TMH1 (**count** * 18 * 533.33 microseconds, or **count** * 9.6 milliseconds).

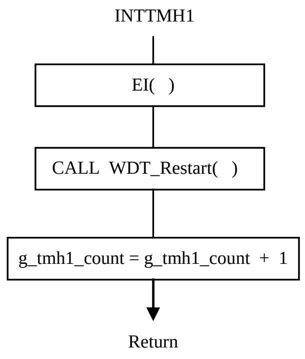
The increasing value of the TMH1 interval changes the time for clearing the watchdog timer. At first the routine clears the timer every 9.6 milliseconds, then every 19.2 milliseconds, then every 28.8 milliseconds, etc. As the values increase, the right LED digit blinks more and more slowly.

When **count** reaches 8, the value set in CMP01 is 18 *8 = 144, and the TMH1 interval is 8 * 9.6 milliseconds = 76.8 milliseconds. This value is longer than the watchdog-timer overflow time, so the watchdog timer overflows.

When the timer overflows, it resets the microcontroller. The program restarts, showing the RESF value in the display and setting RESF bit 4 (WDTRF), indicating that the reset was caused by the watchdog timer. The display shows 10.

2.3.14 MD_INTTMH1 – Timer H1 Interrupt-Service Routine

Figure 21. Timer H1 Interrupt-Service Routine

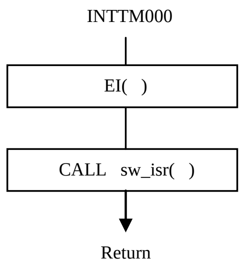


When timer-count register TMH1 matches the compare register CMP01 and asserts INTTMH1, the MD_INTTMH1() interrupt-service routine is invoked. MD_INTTMH1() calls WDT_Restart() to clear the watchdog-timer counter and increments the global variable **g_tmh1_count**.

Note that an EI() instruction executes at the start of this routine to allow interrupts in the same priority group and of higher priority to interrupt the microcontroller. In this case, the INTLVI interrupt can occur during this interrupt-service routine.

2.3.15 MD_INTTM000 – Timer 00 Interrupt-Service Routine

Figure 22. Timer 00 Interrupt-Service Routine

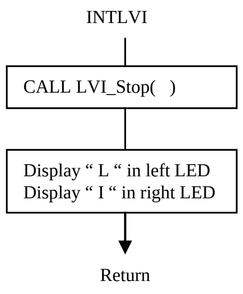


When timer-count register TM00 matches the compare register CMP00, the timer asserts INTTM000, which invokes the MD_INTTM000() interrupt-service routine. This event occurs once every 1 millisecond. The MD_INTTM000() routine calls sw_isr() to check and debounce the switches.

An EI() instruction executes at the start of this routine to allow interrupts in the same priority group and of higher priority to interrupt the microcontroller. As a result, the INTLVI or INTTMH1 interrupts can occur during this interrupt-service routine.

2.3.16 MD_INTLVI – LVI Interrupt Service Routine

Figure 23. MD_INTLVI – LVI Interrupt Service Routine



The INTLVI interrupt invokes the MD_INTLVI() interrupt-service routine. This interrupt occurs if the low-voltage detect circuit is set to interrupt mode (LVIMD=0) and either of two events occurs: if VDD falls below the level set by the LVIS register (in VDD checking mode), or if EXLVI falls below 1.21V (in EXLVI checking mode).

The MD_INTLVI() routine calls LVI_Stop() to disable the LVI detector and to display LI to indicate an INTLVI interrupt.

2.4 Applilet's Reference Driver

NEC Electronics' Applilet program generator automatically generates C or assembly-language source code to manage peripherals for the NEC Electronics microcontroller devices. Please see the Appendix for the version of the Applilet used.

The Applilet produces the program's basic initialization code and main function, driver code for the watchdog timer, LVI (low-voltage) detect, timers TM00 and TMH1, and I/O ports used for switch input and LED-display output. After the Applilet produces the basic code, you can add code to customize the program.

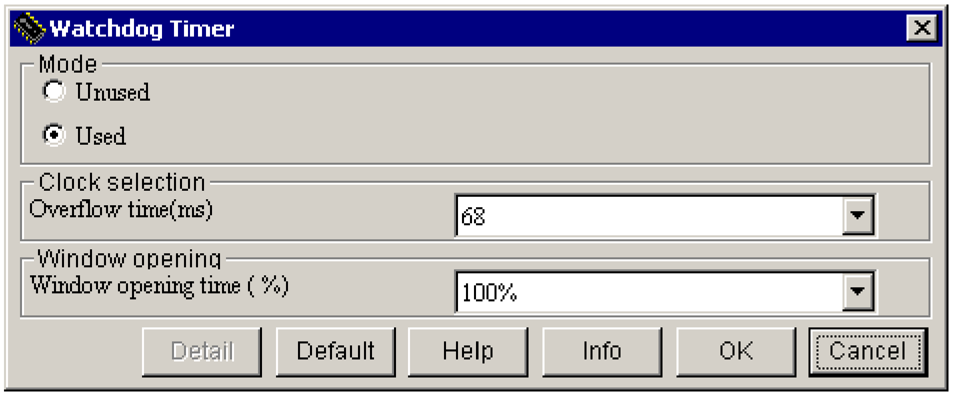
This section describes how to set up the Applilet to produce code for the watchdog timer, LVI, and timers.

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 the Applilet for Watchdog Timer (WDT)

Selecting **watchdog timer** brings up the watchdog-timer configuration dialog.

Figure 24. Watchdog Timer Configuration Dialog Box


Watchdog Timer

Mode
Unused
[✓ Used]

Clock selection
Overflow time(ms): 68

Window opening
Window opening time (%): 100%

Select **Mode** as **Used** to generate watchdog-timer code. Select the **Overflow time** (from the drop-down menu) as 68 milliseconds, and the **Window opening time** for 100%.

Based on these selections, the Applilet generates the appropriate value for storage in the option byte at 0080H and generates routines to control the watchdog timer.

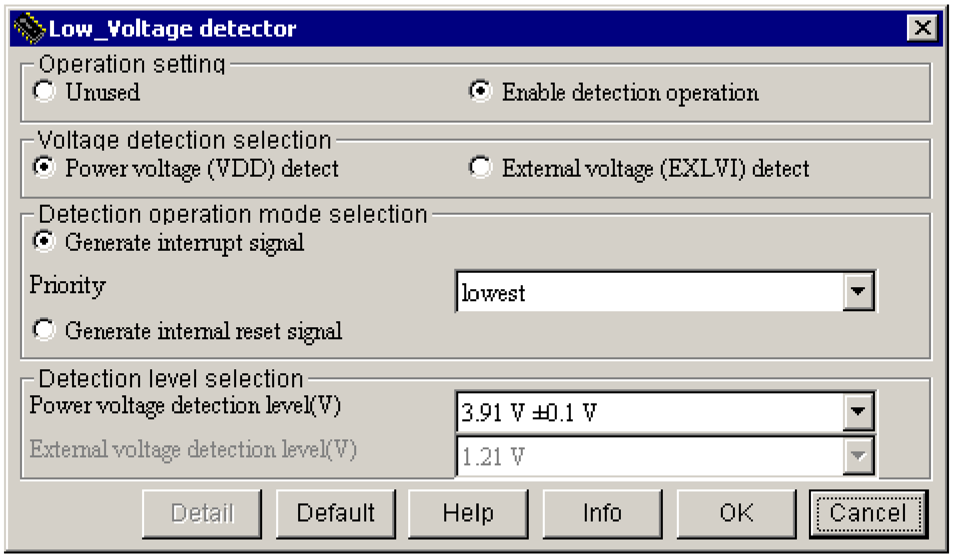
2.4.2 Configuring Applilet for Power-On Clear (POC)

The version of the Applilet used for this demonstration has no dialog to control power-on clear settings. However, the Applilet does produce a value for the POC81 (0081H) location in files option.inc and option.asm. You must edit these files to change the power-on clear settings.

2.4.3 Configuring Applilet for Low-Voltage Detector (LVI)

Selecting **Low-voltage detector** brings up the LVI configuration dialog.

Figure 25. Setting Up LVI Operation



Low_Voltage detector

Operation setting
( ) Unused [✓ Enable detection operation]

Voltage detection selection
[✓ Power voltage (VDD) detect] ( ) External voltage (EXLVI) detect

Detection operation mode selection
[✓ Generate interrupt signal]
Priority: lowest
( ) Generate internal reset signal

Detection level selection
Power voltage detection level(V): 3.91 V ±0.1 V
External voltage detection level(V): 1.21 V

Select **Enable detection operation** to have the Applilet produce code for the LVI detector.

For the demonstration program, you must modify the default LVI settings to demonstrate how the LVI operates in different modes. Under **Voltage-detection selection**, select **Power-voltage (VDD) detect** to initialize the LVI detector to check VDD. This selection controls the initial setting of the LVISEL bit.

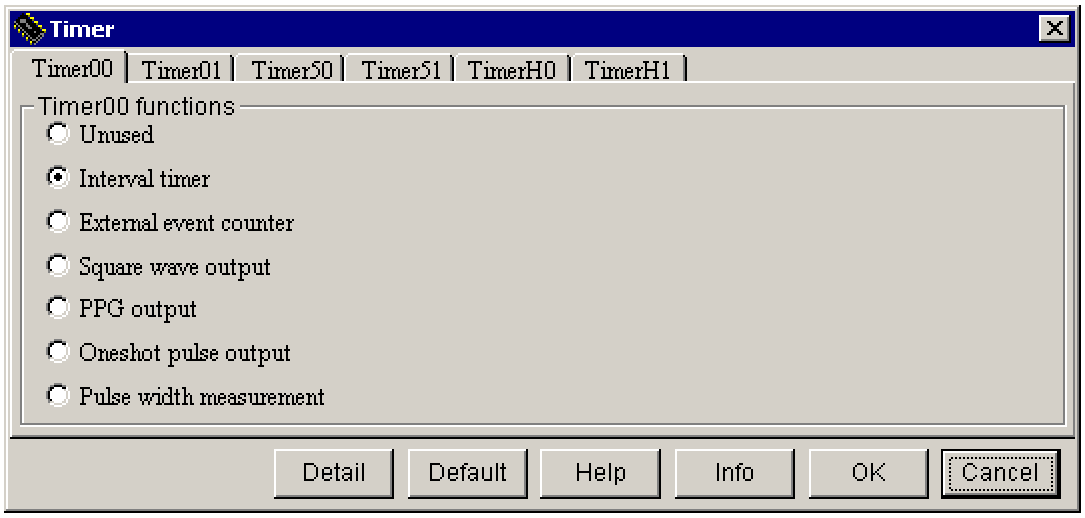
Under **Detection-operation mode selection** select **Generate interrupt signal** to cause the LVI to generate the INTLVI interrupt instead of a reset. This selection controls the initial setting of the LVIMD bit. Now choose the **lowest** priority.

Because you chose to detect VDD, under **Detection level selection** you need to select a voltage; pick 3.91V from the drop-down menu. This selection controls the initial value written to the LVIS register.

2.4.4 Configuring Applilet for Timer TM00 for 1-Millisecond Interval Interrupt

Selecting **Timer** brings up a dialog of various timer blocks. Select **Timer00** and click **Interval timer**.

Figure 26. The Applilet Timer-Selection Screen



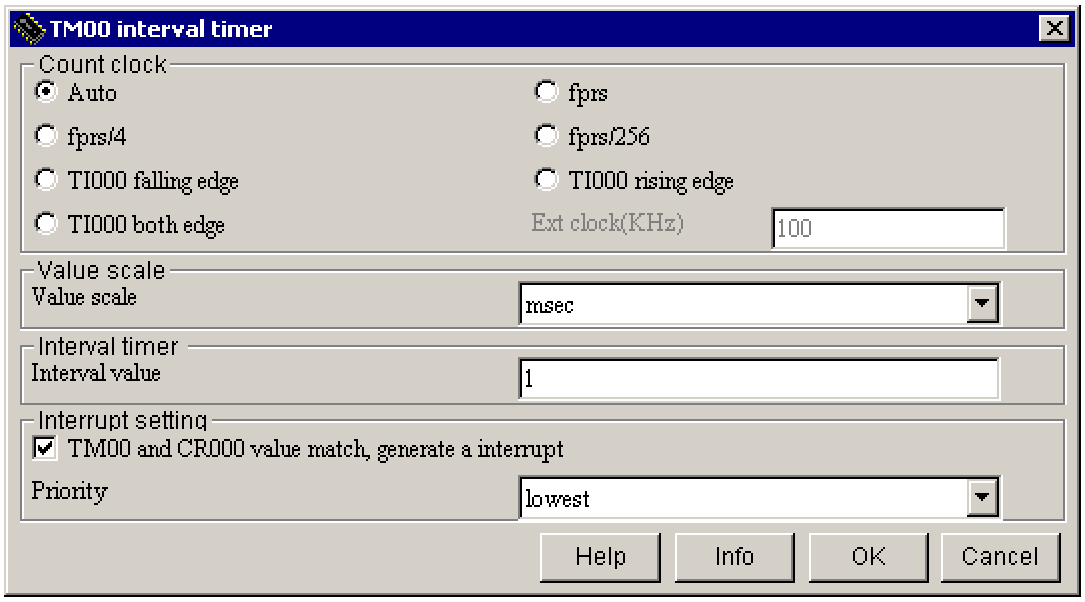
Timer

Timer00 | Timer01 | Timer50 | Timer51 | TimerH0 | TimerH1

Timer00 functions
- Unused
- [✓ Interval timer]
- External event counter
- Square wave output
- PPG output
- Oneshot pulse output
- Pulse width measurement

Now clicking **Detail** brings up the Timer 00 detail dialog for interval-timer settings.

Figure 27. Detail Dialog Box for Interval-Timer Settings



TM00 interval timer

Count clock
- [✓ Auto]
- fprs
- fprs/4
- fprs/256
- TI000 falling edge
- TI000 rising edge
- TI000 both edge

Ext clock(KHz): 100

Value scale
Value scale: msec

Interval timer
Interval value: 1

Interrupt setting
[✓] TM00 and CR000 value match, generate a interrupt
Priority: lowest

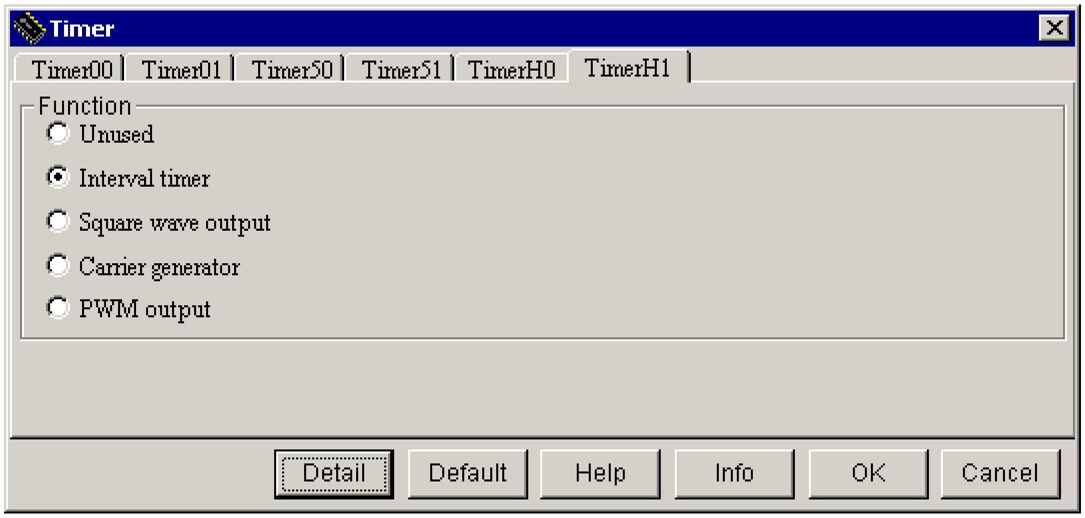
This timer should generate an interrupt every 1 millisecond, so set **Value scale** to **msec** (milliseconds) and **Interval value** to 1. Leave **Count clock** in **Auto** to have the Applilet select an appropriate setting for the timer-clock selection register.

Check **Interrupt setting** to generate an interrupt when TM00 (the timer count register) and CR000 (the timer compare register) match. Set **Priority** to **lowest**.

2.4.5 Configuring Applilet for Timer TMH1 for 10-Millisecond Interval Interrupt

On the timer-selection screen, choose **TimerH1** and click **Interval timer**.

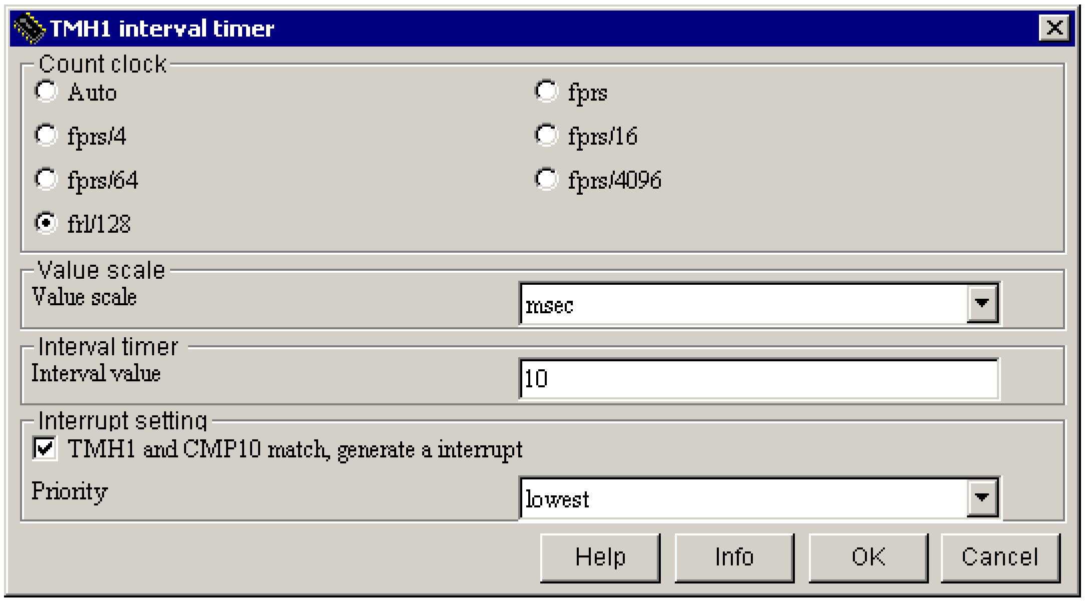
Figure 28. Configuring Applilet for TMH1



Function
( ) Unused
(●) Interval timer
( ) Square wave output
( ) Carrier generator
( ) PWM output

Now clicking **Detail** brings up the interval-timer detail dialog box.

Figure 29. Interval-Timer Detail Dialog Box



TMH1 interval timer

Count clock
- Auto
- fprs
- fprs/4
- fprs/16
- fprs/64
- fprs/4096
- [✓ frl/128]

Value scale
Value scale: msec

Interval timer
Interval value: 10

Interrupt setting
[✓] TMH1 and CMP10 match, generate a interrupt
Priority: lowest

This timer should run on the same clock as the watchdog timer (the internal low-speed oscillator), so select **frl/128** as the **Count clock**. Set **Value scale** to msec and the **Interval timer** value to **10** to generate an interrupt about every 10 milliseconds. The Applilet calculates an appropriate setting for the CMP01 timer comparison register. (The actual value provides a 9.6 millisecond interval — the closest available value to 10.)

Check **Interrupt setting** to have the timer generate an interrupt when TMH1 (the timer-count register) and CMP01 (the timer-compare register) match. (Note that the screen mistakenly shows the register as "CMP10".) Select the **lowest** priority for the interrupt.

2.4.6 Generating Code With Applilet, Selecting Optional Functions

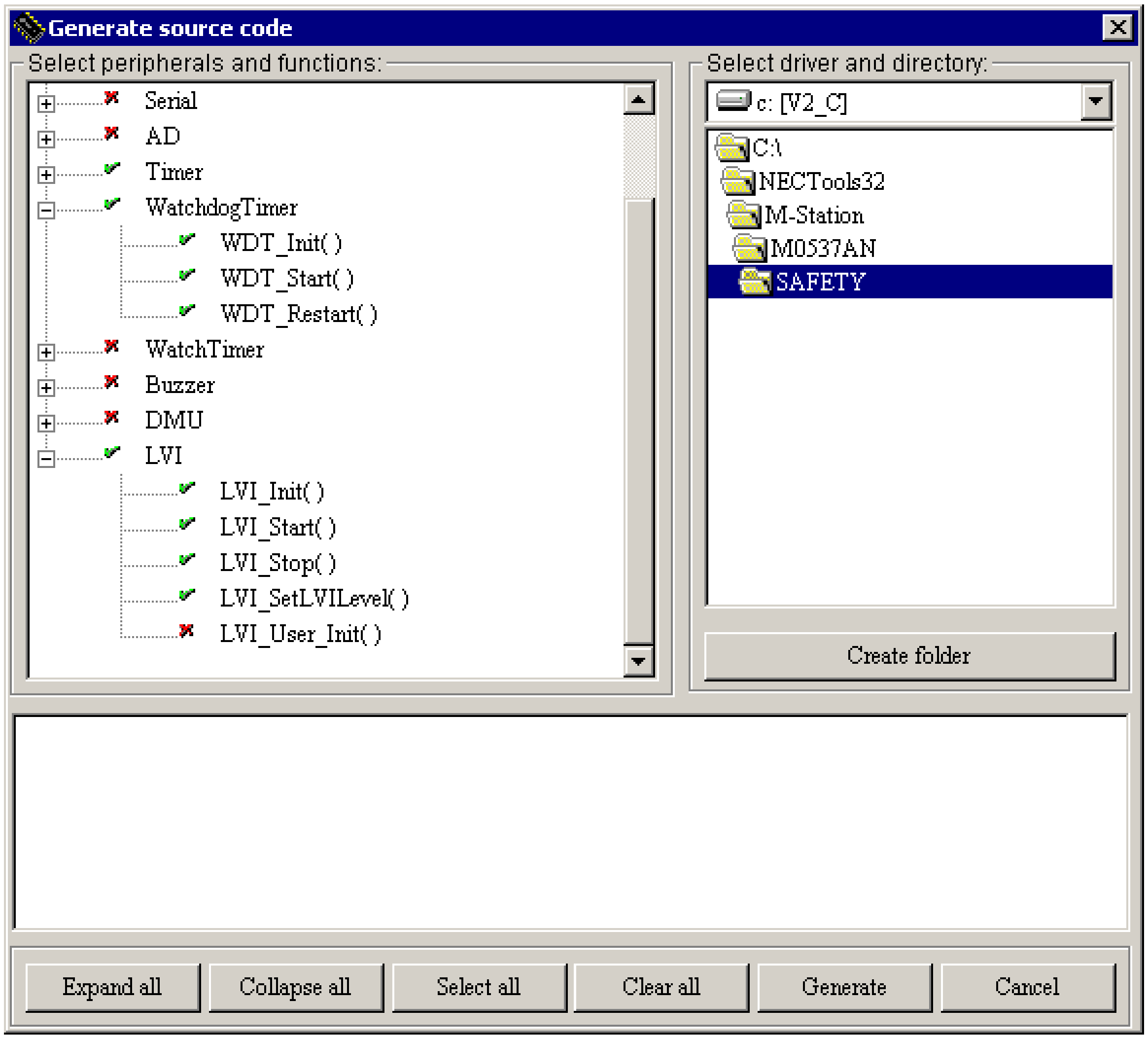
Once you have set up the various dialog boxes, click **Generate code**. The Applilet shows the peripherals and functions to be generated, and allows you to select a directory for the source code.

When you initialize the Applilet for a new project, you might select only some of the available functions. The function tree at the left of the dialog shows which functions are available for each peripheral. A check mark indicates a selected function; an "x" indicates a function not selected. You can change the options by clicking on the mark.

For the watchdog timer, only WDT_Init() may be selected. For the demonstration program you must add WDT_Start() and WDT_Restart().

For LVI, only LVI_Init() may be selected. For the demonstration program, you also select LVI_Start(), LVI_Stop(), and LVI_SetLVILevel().

Figure 30. Program Functions Available



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 the watchdog timer, the Applilet generates watchdogtimer.h, and watchdogtimer.c. The Applilet also generates the assembly language files option.inc and option.asm, which define the option byte area in flash memory.

To support the low-voltage detector, the Applilet generates lvi.h, lvi.c and lvi_user.c.

To support TM00 and TMH1, the Applilet generates timer.h, timer.c and timer_user.c.

To support the I/O ports for switch input and LED display output, the Applilet generates port.h and port.c

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

2.4.7 Applilet-Generated Files and Functions for Watchdog Timer (WDT) and POC

The Applilet stores code generated for watchdog-timer support in the files watchdogtimer.h and watchdogtimer.c. The Applilet stores the code controlling the option byte and the POC byte in the files option.inc and option.asm

2.4.7.1 Watchdogtimer.h

The header file watchdogtimer.h contains declarations for the functions controlling the watchdog timer.

2.4.7.2 Watchdogtimer.c

The source file watchdogtimer.c contains the following functions generated by the Applilet for the watchdog timer:

void WDT_Init(void)
The WDT_Init() initializes the watchdog timer. Since the option byte at 0080H controls watchdog timer settings, this function returns without any action.

void WDT_Start(void)
The WDT_Start() routine writes the value 0xAC to the WDTE register to clear the watchdog-timer overflow counter. This is the same code as WDT_Restart(), since the option byte already started the watchdog timer. For a microcontroller with registers controlling watchdog timer starting, this routine would actually start the watchdog timer.

void WDT_Restart(void)
The WDT_Restart() routine writes the value 0xAC to the WDTE register to clear the watchdog-timer overflow counter. Calling this routine throughout the program prevents overflow reset.

2.4.7.3 Option.inc

The source file option.inc is an assembly-language file included in option.asm. The file contains EQU statements, which define the values placed in the option byte at 0080H, the POC81 byte at 0081H, and other locations.

For the demonstration program, edit the option.inc file to produce two different files:

- Option.inc, which has the POC byte set to 0x00 for the 1.59V mode
- Option_POC27.inc, with the POC81 byte changed to 0x01 to set the 2.7V/1.59V POC mode

2.4.7.4 Option.asm

The assembly-language source file option.asm contains DB (define byte) statements, which allocate memory locations and assign values. The file also contains segment definition directives which specify the location of the memory. This file allocates locations for the option byte at 0080H, the POC81 byte at 0081H, and others, and assigns the values used in option.inc.

For the demonstration program, edit the option.asm file to produce two different files:

- Option.asm, which includes option.inc
- Option_POC27.asm, which includes the modified option_POC27.inc

2.4.8 Applilet-Generated Files and Functions for Low-Voltage Detector (LVI)

The files lvi.h, lvi.c, and lvi_user.c contain the code to support LVI operation.

2.4.8.1 Lvi.h

The header file lvi.h contains declarations for the functions used to initialize, start, stop, and configure the LVI detector.

You must add the declaration for the LVI_Start_R() routine to the Applilet-generated code.

2.4.8.2 Lvi.c

The source file lvi.c contains:

void LVI_Init(void)
The LVI_Init() routine initializes the LVI detector.

void LVI_Start(void)
The LVI_Start() routine starts the LVI detector operation for interrupt mode.

void LVI_Stop(void)
The LVI_Stop() routine clears the LVION bit to stop LVI detector operation, and sets the LVIMK bit to disable the INTLVI interrupt.

MD_STATUS LVI_SetLVILevel(enum LVILevel level)
The LVI_SetLVILevel(level) routine writes the voltage level parameter value to the LVIS register, to set the VDD detection voltage. Lvi.h defines symbolic values for the levels; LVILevel1 is 1, LVILevel2 is 2, etc.

2.4.8.3 Lvi_user.c

The source file lvi_user.c contains stub functions for user code. These functions are empty when generated, and you can add application-specific code.

__interrupt void MD_INTLVI(void)
This is the interrupt-service routine for the LVI-detector interrupt INTLVI, generated when the detected voltage (either VDD or the EXLVI input-pin voltage) falls below the threshold.

The Applilet generates a blank interrupt-service routine. You add code to stop the LVI detector by calling LVI_Stop() and code to display LI on the LED display to indicate an LVI interrupt.

void LVI_Start_R(void)
The Applilet does not generate the LVI_Start_R() routine. You modify the code in LVI_Start() to demonstrate how to start the LVI unit in reset mode. Please see the section below.

2.4.8.4 Applilet Code for LVI Operation in Interrupt and Reset Mode

The code generated by the Applilet for LVI-detector operation is correct for the specified interrupt mode. The sequence of operations in LVI_Init() is:

♦	LVION = 0	Disable LVI unit.
♦	LVISEL=x	Select VDD or EXLVI mode.
♦	LVIMD=0	Select interrupt mode.
♦	---	Set interrupt priority; set LVIS for VDD level if using.
♦	LVION=1	Turn on LVI unit.
♦	(delay)	Wait 10 microseconds minimum.
♦	LVIMK=0	Unmask the LVI interrupt.

LVI_Start() contains similar code; it masks the interrupt and clears the flag, sets LVION, delays, and unmasks the interrupt.

However, for the version of the Applilet used, if the reset mode of operation is selected for LVI, the LVI_Init() and LVI_Start() code contains a minor error. Specifically, both routines set the LVION bit immediately after setting the LVIMD bit to 1 (reset mode). This sequence could cause an immediate reset at an inappropriate level while the LVI-detector voltage stabilizes. For correct operation in the reset mode, first set the LVION bit, then delay 10 miscroseconds for the voltage to stabilize before setting the LVIMD bit. The proper sequence of operations in LVI_Init() is:

♦ LVION = 0	Disable LVI unit.
♦ LVISEL=x	Select VDD or EXLVI mode.
♦ LVIMD=0	Select interrupt mode temporarily.
♦ ---	Set interrupt priority; set LVIS for VDD level if using.
♦ LVION=1	Turn on LVI unit.
♦ (delay)	Wait 10 microseconds minimum.
♦ LVIMD=1	Set reset mode.

If using the Applilet for LVI control in reset mode, please check for proper sequences in the LVI_Init() and LVI_Start() routines.

2.4.9 Applilet-Generated Files and Functions for TM00 and TMH1

The files timer.h, timer.c and timer_user.c contain code for TM00 and TMH1 support.

2.4.9.1 Timer.h

The header file timer.h contains declarations for the functions controlling the timers, and definitions of values for timer 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.

To make the global variable **g_tmh1_count** available to other routines, add an external declaration of this variable to the Applilet-generated file.

2.4.9.2 Timer.c

The source file Timer.c contains the following functions:

void TM00_Init(void)
The TM00_Init() routine initializes Timer 00.

void TM00_Start(void)
The TM00_Start() routine starts Timer 00 operation, enabling the both the timer and interrupt INTTM000.

void TM00_Stop(void)
The TM00_Stop() routine stops Timer 00 operation by disabling the timer and the timer interrupt. The demonstration program does not use this routine.

MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg, USHORT array_num)
The TM00_ChangeTimerCondition() function changes the value in the Timer 00 compare registers, CR000 and CR010, and therefore changes the interval for Timer 00.

The array_reg parameter points to an array of values to be put in one or the other of the compare registers. The array_num parameter is either 1 (to select CR000) or 2 (to select both CR010 and CR000). The demonstration program does not use this routine.

void TMH1_Init(void)
The TMH1_Init() routine initializes the TMH1.

void TMH1_Start(void)
The TMH1_Start() routine starts TMH1 operation — enabling the timer and the interrupt INTTMH1.

void TMH1_Stop(void)
The TMH1_Stop() routine stops TM51 operation by disabling the timer and the interrupt. The demonstration program does not use this routine.

MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg, UCHAR array_num)
The TMH1_ChangeTimerCondition() function changes the value in the Timer H1 compare registers, CMP01 and CMP11, and therefore changes the interval for Timer H1.

The array_reg parameter points to an array of values to be put in one or the other of the compare registers. The array_num parameter is either 1 (to select CMP01) or 2 (to select both CMP01 and CMP11).

2.4.9.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. The Applilet generates empty function, and you can add application-specific code.

__interrupt void MD_INTTM000(void)
This is the interrupt-service routine for Timer 00 interrupt INTTM000, generated when TM00 and CR000 values match. Once started, the timer generates this interrupt once every 1 millisecond.

The Applilet generates this routine blank. To use the interval timer to debounce switches by checking their value every millisecond, you add code to MD_INTTM000() to call the sw_isr() function.

__interrupt void MD_INTTMH1(void)
This is the interrupt-service routine for the Timer H1 interrupt INTTMH1, generated when TMH1 and CMP01 values match. After initial timer set up, the timer generates this interrupt once every 9.6 milliseconds.

The Applilet generates this routine blank. You add code to call WDT_Restart() to reset the watchdog timer. You need to also add code to increment the global variable **g_tmh1_count—this,** which the watchdog-timer demo subroutine uses to indicate the rate of INTTMH1 occurrences.

2.4.10 Applilet-Generated Files and Functions for Port Initialization

The files port.h and port.c contain the code generated for I/O port support.

2.4.10.1 Port.h

The header file port.h contains declarations for the PORT_Init() function used to initialize the ports and definitions of values for initialization of port-output latches, port-mode registers, and port pull-up registers. For example, to initialize port P0, you write values to P0, PM0, and PU0. The file port.h defines these values.

To change the port initialization values, edit the definitions in port.h. Modifying the value of PORT_PU0 sets pull-up resistors on pins P04 and P05 for when they are not used as outputs in the key-scan routine.

2.4.10.2 Port.c

The source file port.c contains the following function for port initialization:

void PORT_Init(void)
The PORT_Init() routine initializes all of the device I/O ports by setting the port-output latch, port-mode register, and pull-up register for each of the ports. The file port.h defines the values.

2.4.11 Other Applilet-Generated Files

For the demonstration program, the Applilet generates several other source files.

Table 4. 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
System.h	Clock-related definitions
System.c	Clock_Init() function

2.4.12 Demonstration Program Files Not Generated by Applilet

The demonstration program includes the following files that the Applilet does not generate.

Table 5. Demonstration Program Files not generated by Applilet

File	Function
Sw_0537.h	Header file for push-button switch input
Sw_0537.c	Code to read and debounce pushbutton switches
Led_0537.h	Header file for seven-segment LED patterns and functions
Led_0537.c	Code to display data in seven-segment LED displays

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:

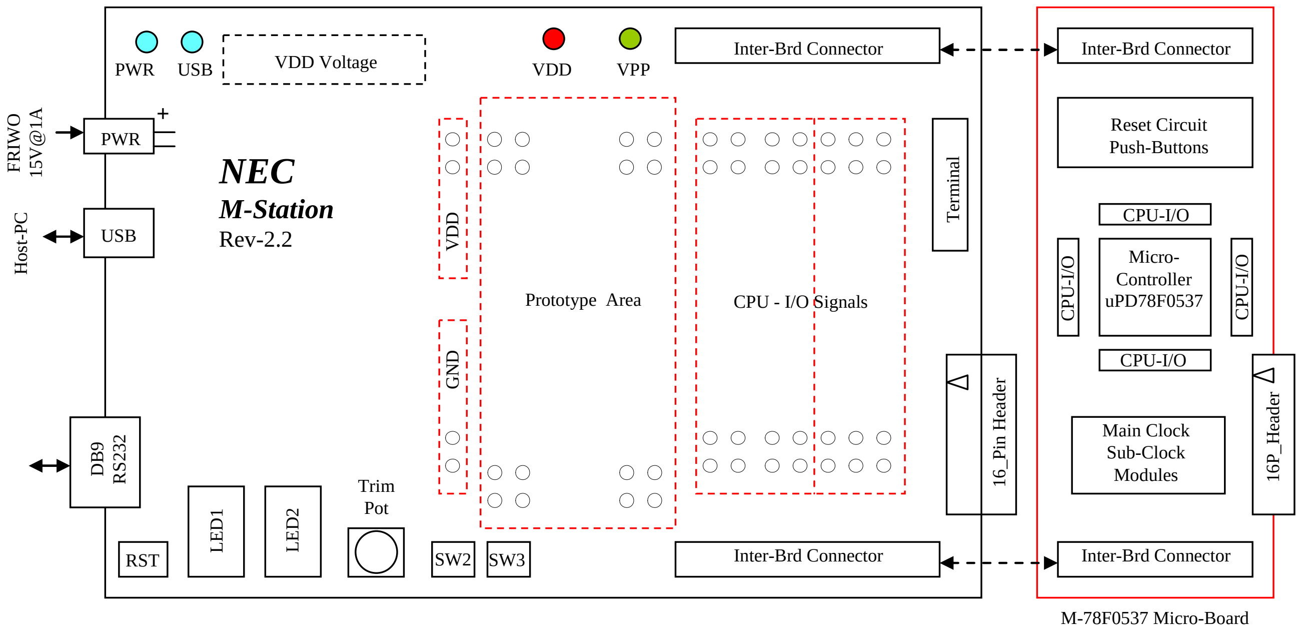
- ♦ M-78F0537 Micro-Board, with µPD78F0537 8-bit microcontroller mounted
- ♦ M-Station II Evaluation System, using M-Station II resources:
  - – RST switch to generate low-level reset
  - – 7-segment LED displays LED1 and LED2
  - – Pushbutton switches SW2 and SW3
  - – MH_009 standard M-Station-II potentiometer connected between VDD and GND

You need to add a wire from the MH_009 potentiometer’s tap to microcontroller pin P120/EXLVI (J1_B.17). (The tap is already connected to µPD78F0537 analog input P20/ANI0.)

You also need to add a 100-ohm potentiometer between VDD_FLASH (JP1.1) and GND, with the tap connected to VDD (JP1.2). This potentiometer allows you to vary VDD.

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

Figure 31. Demonstration Platform



2.5.2 Demonstration of Program

With the hardware configured and the µPD78F0397 microcontroller programmed with the demonstration code, the demonstration includes the following sequences.

2.5.2.1 External reset Demonstration

- ♦ Apply VDD power above Vpoc threshold; observe “00” on LED display.
- ♦ Press either SW2 or SW3; see “=1” (selection of test).
- ♦ Press RST switch to assert reset; on release, see “00” on LED display.

2.5.2.2 Power-On Clear (POC) Demonstration

The demonstration program has two versions of this demonstration. One version uses the option.inc and option.asm files, which set power-on clear to the 1.59V mode; on power-up, Vpoc is therefore 1.59V. The other version of the demonstration uses the option_POC27.inc and option_POC27.asm files, which set the POC mode to the 2.7/1.59V mode; on power-up, Vpoc is 2.7V and afterwards 1.59V. The sequence for both versions is as follows:

- Apply VDD power below Vpoc threshold; observe a blank LED display.
- Raise VDD above Vpoc threshold; the LED display shows “00”.
- Press either SW2 or SW3; see “=1” in LED display.
- Lower VDD below 1.59V; LED goes blank (reset asserted).
- Raise VDD above 1.59V; LED display shows “00”.

2.5.2.3 Low-Voltage Detect (LVI) Demonstration

After pressing SW2 or SW3 and seeing “=1” displayed, press SW2 to step the display to “=2”, “=3”, through “=7” and back to “=1”. These values represent seven tests; pressing SW3 executes a test. When you press SW3, the left digit shows the test number, and the right digit is blank. Tests 1 through 6 are LVI tests, and Test 7 is the watchdog-timer demonstration:

Table 6. Demonstration’s Seven Tests

Test	LED	Test Function	Function Operation
1	“1 ”	LVI_Reset_239()	Set LVI to reset on VDD < 2.39V
2	“2 ”	LVI_Reset_393()	Set LVI to reset on VDD < 3.93V
3	“3 ”	LVI_Int_239()	Set LVI to interrupt on VDD < 2.39V
4	“4 ”	LVI_int_393()	Set LVI to interrupt on VDD < 3.93V
5	“5 ”	EXLVI_Reset()	Set LVI to reset on EXLVI < 1.21V
6	“6 ”	EXLVI_Int()	Set LVI to interrupt on EXLVI < 1.21V
7	“7 ”	WDT_Demo()	Show reset on WDT counter overflow

Demonstrate LVI reset on VDD low voltage:

- Apply VDD power above 3.93V.
- Select test “=1” or “=2”, press SW3; see “1 ” or “2 ”.
- Reduce VDD below set voltage.
- See LED display blank at appropriate voltage.
- Raise VDD above set voltage.
- See LED display show “01” indicating LVI caused reset.

Demonstrate LVI interrupt on VDD low voltage:

- Apply VDD power above 3.93V.
- Select test “=3” or “=4”; press SW3; see “3 ” or “4 ”.
- Reduce VDD below set voltage.

- See LED display show “LI” at appropriate voltage.
- Press SW2 to return to test menu.

Demonstrate LVI reset on EXLVI low voltage:

- Apply VDD power.
- Set potentiometer for EXLVI above 1.21V.
- Select test “=5”; press SW3; see “5 ”
- Reduce EXLVI voltage below 1.21V.
- See LED display blank at appropriate voltage.
- Raise EXLVI voltage above 1.21V.
- See LED display show “01” indicating LVI caused reset.

Demonstrate LVI interrupt on EXLVI low voltage:

- Apply VDD power.
- Select test “=6”; press SW3; see “6 ”.
- Reduce EXLVI voltage below 1.21V.
- See LED display “LI” at appropriate voltage.
- Raise EXLVI voltage above 1.21V.
- Press SW2 to return to menu of tests.

2.5.2.4 Watchdog-Timer Demonstration

Initialization sets the watchdog timer to overflow after about 68 milliseconds. Initialization also sets Timer H1 to interrupt every 9.6 milliseconds and the interrupt-service routine for Timer H1 to clear the watchdog-timer counter. Use these steps to run the watchdog-timer demonstration:

- Select test “=7”; press SW3.
- See “71”, with right digit blinking rapidly.

At this point, Timer H1 interrupt INTTMH1 occurs once every 9.6 milliseconds.

- Press SW3 repeatedly.
- See “72”, “73”, etc., with right digit blinking more slowly.

Every press of SW3 adds another 9.6 milliseconds to the Timer H1 interval. A variable incremented in the INTTMH1 interrupt-service routine causes the blinking. As the interval increases, INTTMH1 occurs less frequently, so the blinking slows.

The table below relates the number of times you press SW3 to Timer H1's interval and the LED display.

Table 7. Watchdog Timer Interrupt Intervals

SW3 Presses	Timer H1 Interval	LED Display
1	9.6 msec	"71", 1 is blinking rapidly
2	19.2 msec	"72", 2 is blinking more slowly
3	28.8 msec	"73", 3 is blinking more slowly
4	38.4 msec	"74", 4 is blinking more slowly
5	48.0 msec	"75", 5 is blinking more slowly
6	57.6 msec	"76", 6 is blinking more slowly
7	67.2 msec	"77", 7 is blinking more slowly
8	76.8 msec	"10"; Watchdog Timer has caused reset

When the interval for Timer H1 interrupt exceeds 68 milliseconds, the watchdog timer overflows, causing a watchdog-timer reset. The program restarts and shows the value "10" for RESF, which indicates that the watchdog timer caused the reset.

The reset reinitializes the system, so the Timer H1 interval is again 9.6 milliseconds.

2.5.3 Demonstration Using M-Station VDD Selection Instead of Potentiometer

The M-Station-II offers selectable VDD levels, which you can use for a slightly limited version of this demonstration. The VDD SET switch allows you to turn VDD off or on. When off, you can select VDD as 2.0, 2.5, 3.0, 3.3, 4.0, 4.5, or 5.0V.

To demonstrate power-on-clear:

- ♦ Use the version of the demonstration program with POC mode 2.7/1.59V.
- ♦ Turn VDD off by holding down the VDD SET switch for 5 seconds.
- ♦ Set VDD selection to 2.5V.
- ♦ Turn VDD on by holding down the VDD SET switch for 5 seconds.
- ♦ Observe that the LED display is blank (VDD is below Vpoc threshold).
- ♦ Turn VDD off.
- ♦ Set VDD to 3.0V.
- ♦ Turn VDD on.
- ♦ Observe that the LED display is “00” (VDD is above Vpoc threshold).

Finally, reprogram the device with the version of the demonstration program that uses POC mode 1.59V. Observe that the program runs for all values of VDD from 2.0 to 5.0V.

Note: When changing the value of the POC81 byte at location 0081H, you must erase the µPD78F0537 fully (chip mode erase). Partial erasure (using block mode programming) or self-flash-programming will not change the value of the POC81 byte.

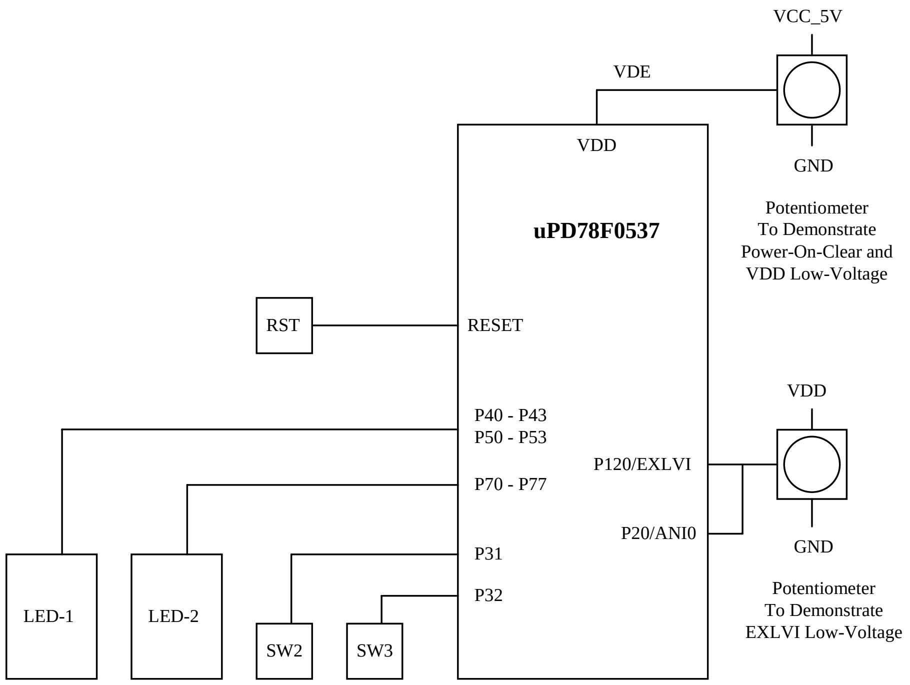
To demonstrate low-voltage detection:

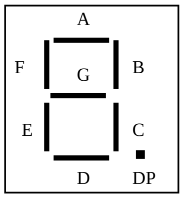
- ♦ Turn VDD on at 5.0V.
- ♦ Run tests =1, =2, =3, =4.
- ♦ Observe no LVI occurs (display does not blank or show “LI”).
- ♦ Turn VDD off, then on at 3.3V.
- ♦ Run tests =1, =2, =3, =4.
- ♦ Observe no LVI on tests =1 or =3 (VDD is above 2.39V).
- ♦ Observe LED display blank on test =2 (VDD is below 3.93V, reset).
- ♦ Observe LED display “LI” on test =4 (VDD is below 3.93V, interrupt).
- ♦ Make sure demonstration program with POC mode 1.59V is programmed.
- ♦ Turn VDD off, then on at 2.0V (display will be dim).

- Run tests =1, =2, =3, =4.
- Observe that the LED display blanks on tests =1 and =3 (VDD is below 2.39V, reset).
- Observe that the LED displays “LI” on tests =3 and =4 (VDD is below 2.39V, interrupt).

2.6 Hardware Block Diagram

Figure 32. Hardware Block Diagram


VCC_5V
VDE
VDD
GND
uPD78F0537
Potentiometer To Demonstrate Power-On-Clear and VDD Low-Voltage
RST
RESET
VDD
P40 - P43
P50 - P53
P120/EXLVI
P70 - P77
P20/ANI0
GND
P31
P32
Potentiometer To Demonstrate EXLVI Low-Voltage
LED-1
LED-2
SW2
SW3



LED-1 and LED-2

Segments	LED-1	LED-2
A	P40	P70
B	P41	P71
C	P42	P72
D	P43	P73
E	P50	P74
F	P51	P75
G	P52	P76
DP	P53	P77

2.7 Software Modules

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

The listings for these files are in the Appendix.

Table 8. 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
Lvi.h	LVI-related definitions	Yes	Yes[Note 1]
Lvi.c	LVI functions	Yes	Yes[Note 1]
Lvi_user.c	User code for LVI interrupt handling	Yes	Yes[Note 1]
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	Yes[Note 2]
Timer.c	Timer functions	Yes	No
Timer_user.c	User code for timer interrupt handling	Yes	Yes[Note 2]
Watchdogtimer.h	WDT-related definitions	Yes	No
Watchdogtimer.c	WDT functions	Yes	No
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No
Led_0537.h	LED display definitions	--	Yes
Led_0537.c	LED display functions	--	Yes
Sw_0537.h	Switch input definitions	--	Yes
Sw_0537.c	Switch input functions	--	Yes

Note 1: You must modify Lvi.h to add the declaration for the LVI_Start_R() routine and Lvi.c to comment-out the enabling of the LVI interrupt at the end of the LVI_Init() routine. Add code in Lvi_user.c to handle the INTLVI interrupt in the MD_INTLVI() routine, and to add the LVI_Start_R() routine.

Note 2: Add the declaration of the global variable **g_tmh1_count** in Timer.h. Add code to Timer_user.c to handle the INTTM000 interrupt in the MD_INTTM000() routine for debouncing switches, to handle the INTTMH1 interrupt in the MD_INTTMH1() routine for clearing the watchdog timer, and to add the global variable g_tmh1_count, used to track the rate of INTTMH1 interrupts.

The files listed in the following table implement a version of the program that sets the power-on clear function to 2.7V.

Table 9. Files for Setting Power-On Clear Function to 2.7V

File	Purpose	Generated By Applilet	Modified By User
Option_POC27.inc	Option-byte, POC, and security definitions	Yes	Yes[Note]
Option_POC27.asm	Option-byte, POC, and security data	Yes	Yes[Note]

Note: Modify Option_POC27.inc from option.inc to change the POC81 value from 00H to 01H, setting the POC mode to the 2.7V/1.59V mode. Modify Option_POC27.asm from option.asm to include the option_POC27.inc file instead of option.inc. Other than these changes, the files remain as the Applilet generated them.

3. Appendix A - Development Tools

This application uses the following software and hardware tools.

3.1 Software Tools

Table 10. Software Development Tools

Tool	Version	Comments
Applilet for 78K0KX2	V1.51	Source code generation tool for 78K0/KE2 devices
PM Plus	V5.20	Project Manager for program compilation and linking
CC78K0	V3.60	C Compiler for NEC Electronics 78K0 devices
RA78K0	V3.70	Assembler for NEC Electronics 78K0 devices
DF053764.78K	V2.00	Device file for µPD78F0537_64 device

3.2 Hardware Tools

Table 11. Hardware Development Tools

Tool	Version	Comments
M-Station 2	V2.1E	Base platform for NEC Electronics Micro-board demonstration
M-78F0537	V1.0	NEC Electronics Micro-board for µPD78F0537; CPU chip is µPD78F0537DGB

Modify the M-Station 2 by adding a potentiometer between JP1.1 (VDD_FLASH) and GND, with the tap of the potentiometer connected to JP1.2 (VDE). This modification lets you vary VDD to the µPD78F0537 microcontroller.

Make a connection between J1_B.17 (M-Station signal J1_043, M-78F0537 signal P120/EXLVI) and MH_009 (M-Station Potentiometer between VDD and GND). This connection lets you apply a variable voltage to the EXLVI input.

4. Appendix B – Software Listings

Note: Use the files option_POC27.inc and option_POC27.asm to build an alternate version of the demonstration program with the power-on-clear function set in 2.7V/1.59V mode.

4.1 Main.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 : 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 "port.h"
#include "timer.h"
#include "watchdogtimer.h"
#include "lvi.h"

/* add includes for non-Applilet functions */
#include "led_0537.h"
#include "sw_0537.h"

/*
******************************************************************************
** MacroDefine
******************************************************************************
*/
#define      TEST_MAX     7

/* test function definitions */
void LVI_Reset_239(void);        /* Set LVI for Reset on 2.39V */
void LVI_Reset_393(void);        /* Set LVI for Reset on 3.93V */
void LVI_Int_239(void);          /* Set LVI for Interupt at 2.39V */
void LVI_Int_393(void);          /* Set LVI for Interupt at 3.93V */
void EXLVI_Reset(void);          /* Set for EXLVI Reset at 1.51V */
void EXLVI_Int(void);            /* Set for EXLVI Interrupt at 1.51V */
void WDT_Demo(void);             /* Watchdog timer demo */

/* define PF_VF_RV as pointer to a function, void parameter, returning void */
typedef void (*PF_VF_RV)(void);

PF_VF_RV menu[TEST_MAX] = {
       LVI_Reset_239,
       LVI_Reset_393,
       LVI_Int_239,
       LVI_Int_393,
       EXLVI_Reset,
       EXLVI_Int,
       WDT_Demo };

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void main( void )
{
unsigned char resf_val;    /* value read from RESF */
unsigned char test;        /* selection of test to run */
unsigned char sw_val;      /* value of switches */

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;
       /* TODO. add user code */

       TMH1_Start();
       WDT_Start();

       led_init();         /* initialize LEDs */
       sw_init();          /* initialize switch variables */
       sw_set_debounce(10);      /* set debounce counter to 10 for 10 msec stable time */


       TM00_Start();/* start timer for switch debouncing */

       resf_val = RESF;    /* read reset flag register */
       led_dig(resf_val);  /* display the value */
       while (SW_LU_RU == sw_get())
              ;                          /* wait for switch down */
       while (SW_LU_RU != sw_get())
              ;                          /* wait for switch up */

       test = 1;                 /* set initial test number */
       while(1){
              led_out_left(LED_PAT_EQUAL);     /* display an equal sign */
```

```
             led_dig_right(test);                    /* display number of test to execute 
*/

             while (SW_LU_RU == sw_get())
                   ;                                             /* wait for switch 
pressed */
             sw_val = sw_get();                      /* get value of switch */

             if (sw_val == SW_LD_RU) {
                   /* SW2 (left) is down, right (SW3) is up */
                   test = test + 1;                  /* increment the test number */
                   if (test > TEST_MAX)
                         test = 1;                         /* wrap around if hit max */
                   led_dig_right(test);              /* update display right away */
             }

             if (sw_val == SW_LU_RD) {
                   /* SW2 up, SW3 down */
                   while (SW_LU_RU != sw_get())
                         ;                                       /* wait for switch up 
again */
                   led_dig_left(test);               /* show test in left digit */
                   led_out_right(LED_PAT_BLANK);     /* show blank in right digit */

                   (menu[test-1])();                       /* execute function */

             }

             while (SW_LU_RU != sw_get())
                   ;                                             /* wait for switch up */
       } /* end of while (1) loop */
} /* end of main() */

void LVI_Reset_239(void)
{
       /* Set LVI for Reset on 2.39V */
       LVI_Stop();          /* make sure we are stopped */
       LVISEL = 0;          /* select LVI checking VDD level as set by LVIS register */
       LVIMD = 0;           /* set LVI mode to interrupt temporarily */
       LVI_SetLVILevel(LVI_Level12);    /* set LVIS for level 12 (2.39V) */
       LVI_Start_R();       /* start the LVI detector without enabling interrupt */
       LVIMD = 1;           /* set mode to Reset on LVI */

       /* wait for switch press and release before returning */
       while (SW_LU_RU == sw_get())     ;
       while (SW_LU_RU != sw_get())     ;
}


void LVI_Reset_393(void)
{
       /* Set LVI for Reset on 3.93V */
       LVI_Stop();          /* make sure we are stopped */
       LVISEL = 0;          /* select LVI checking VDD level as set by LVIS register */
       LVIMD = 0;           /* set LVI mode to interrupt temporarily */
       LVI_SetLVILevel(LVI_Level2);     /* set LVIS for level 2 (3.93V) */
       LVI_Start_R();       /* start the LVI detector without enabling interrupt */
       LVIMD = 1;           /* set mode to Reset on LVI */

       /* wait for switch press and release before returning */
       while (SW_LU_RU == sw_get())     ;
       while (SW_LU_RU != sw_get())     ;
```

```
}


void LVI_Int_239(void)
{
      /* Set LVI for Interupt at 2.39V */
      LVI_Stop();         /* make sure we are stopped */
      LVISEL = 0;         /* select LVI checking VDD level as set by LVIS register */
      LVIMD = 0;          /* set mode to give interrupt on LVI */
      LVI_SetLVILevel(LVI_Level12);    /* set LVIS for level 12 (2.39V) */
      LVI_Start(); /* start LVI detector and enable the interrupt */

      /* wait for switch press and release before returning */
      while (SW_LU_RU == sw_get())     ;
      while (SW_LU_RU != sw_get())     ;
}


void LVI_Int_393(void)
{
      /* Set LVI for Interupt at 3.93V */
      LVI_Stop();         /* make sure we are stopped */
      LVISEL = 0;         /* select LVI checking VDD level as set by LVIS register */
      LVIMD = 0;          /* set mode to give interrupt on LVI */
      LVI_SetLVILevel(LVI_Level2);     /* set LVIS for level 2 (3.93V) */
      LVI_Start(); /* start LVI detector and enable the interrupt */

      /* wait for switch press and release before returning */
      while (SW_LU_RU == sw_get())     ;
      while (SW_LU_RU != sw_get())     ;
}


void EXLVI_Reset(void)
{
      /* Set for EXLVI Reset at 1.51V */
      LVI_Stop();         /* make sure we are stopped */
      LVISEL = 1;         /* select LVI checking EXLVI level above 1.21V */
      LVIMD = 0;          /* set LVI mode to interrupt temporarily */
      LVI_Start_R();      /* start the LVI detector without enabling interrupt */
      LVIMD = 1;          /* set mode to Reset on LVI */

      /* wait for switch press and release before returning */
      while (SW_LU_RU == sw_get())     ;
      while (SW_LU_RU != sw_get())     ;
}


void EXLVI_Int(void)
{
      /* Set for EXLVI Interrupt at 1.51V */
      LVI_Stop();         /* make sure we are stopped */
      LVISEL = 1;         /* select LVI checking EXLVI level above 1.21V */
      LVIMD = 0;          /* set mode to give interrupt on LVI */
      LVI_Start(); /* start LVI detector and enable the interrupt */

      /* wait for switch press and release before returning */
      while (SW_LU_RU == sw_get())     ;
      while (SW_LU_RU != sw_get())     ;
}
```

```
void WDT_Demo(void)
{
UCHAR count;
UCHAR reg[2];
UCHAR sw_val;
       /* Watchdog timer demo */
       count = 1;                                          /* number of timer increments
in timer */
       reg[0] = TM_TMH1_INTERVALVALUE;         /* set TMH1 for one increment */
       TMH1_ChangeTimerCondition(&reg[0],1);

       while (1) {
              while (SW_LU_RU == (sw_val = sw_get())) {
                     /* until switch down, blink count in right LED */
                     /* g_tmh1_count counting speed will decrease as */
                     /* timer count increases, and blinking will slow down */
                     if (g_tmh1_count & 0x04)
                            led_out_right(LED_PAT_BLANK);
                     else
                            led_dig_right(count);
              }
              /* switch has been pressed */
              while (SW_LU_RU != sw_get())
                     ;      /* wait for switch up */

              if (sw_val == SW_LU_RD) {
                     /* if right switch pressed, increment count, and add to interval */
                     count = count + 1;
                     reg[0] = reg[0] + (TM_TMH1_INTERVALVALUE + 1);
                     TMH1_ChangeTimerCondition(&reg[0],1);
                     /* now timer interval = (count) * (base interval) */
              }

              if (sw_val == SW_LD_RU) {
                     break; /* exit loop */
              }
       } /* end of while (1) loop */
}
```

4.2 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
```

4.3 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
```

4.4 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 "port.h"
#include "timer.h"
#include "watchdogtimer.h"
#include "lvi.h"
/*
```

```
*******************************************************************************
** MacroDefine
*******************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     Init every Macro
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* Clock generator initiate */
       Clock_Init();
       /* Port initiate */
       PORT_Init();
       /* WDT initiate */
       WDT_Init();
       /* TM00 initiate */
       TM00_Init();
       /* TMH1 initiate */
       TMH1_Init();
       /* LVI initiate */
       LVI_Init();
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     Init hardware setting
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void hdwinit( void )
{
       DI( );
       SystemInit( );
       EI( );
}
```

4.5 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 */
      ClrIORBit(OSCCTL, 0x10);
      SetIORBit(MOC, 0x80);                  /* stop X1 clock */
      PCC = CG_CPU_CLOCKSEL;
}
```

4.6 Lvi.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 : lvi.h
** Abstract : This file implements device driver for LVI module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
********************************************************************************
*/
#ifndef      _MDLVI_
#define      _MDLVI_
/*
********************************************************************************
** MacroDefine
********************************************************************************
*/
enum LVILevel {
       LVI_Level0, LVI_Level1, LVI_Level2, LVI_Level3,
       LVI_Level4, LVI_Level5, LVI_Level6, LVI_Level7,
       LVI_Level8, LVI_Level9, LVI_Level10, LVI_Level11,
       LVI_Level12, LVI_Level13, LVI_Level14,LVI_Level15
       };

void LVI_Init( void );
void LVI_Start( void );
MD_STATUS LVI_SetLVILevel( enum LVILevel level );
void LVI_Stop( void );
__interrupt void MD_INTLVI( void );

/* add special version of LVI_Start() for Reset mode for demo program */
void LVI_Start_R( void );

#endif
```

4.7 Lvi.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 : lvi.c
** Abstract : This file implements device driver for LVI module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
******************************************************************************
*/
/*
******************************************************************************
** Include files
******************************************************************************
*/
#include "macrodriver.h"
#include "lvi.h"

/*
**----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes the Low-Voltage Detector.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void LVI_Init( void )
{
      USHORT i;

      LVION = 0;                  /* stop LVI */
      LVISEL = 0;                 /* detect supply voltage (VDD) */
      LVIMD = 0;                  /* internal interrupt mode */
      LVIPR = 1;                  /* low priority level */
      LVIIF = 0;
```

```
        LVIS = LVI_Level2;          /* compare with 3.91V */

        LVION = 1;                  /* enable LVI */
        for(i=0; i<=100; i++){              /* wait 10 us */
                NOP();
        }
        /* for demonstration program, do not unmask LVI interrupt here */
#if 0
        LVIMK = 0;
#endif
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**      This function starts the Low-Voltage Detector.
**
** Parameters:
**      None
**
** Returns:
**      None
**
**-----------------------------------------------------------------------------
*/
void LVI_Start( void )
{
        USHORT i;
        LVIMK = 1;
        LVIIF = 0;

        LVION = 1;                  /* enable LVI */
        for(i=0; i<=100; i++){              /* wait 10 us */
                NOP();
        }
        LVIMK = 0;
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**      This function stops the Low-Voltage Detector.
**
** Parameters:
**      None
**
** Returns:
**      None
**
**-----------------------------------------------------------------------------
*/
void LVI_Stop( void )
{
        LVION = 0;                  /* stop LVI */
        LVIMK = 1;
}

/*
**-----------------------------------------------------------------------------
**
```

```
** Abstract:
**     This function set detection level.
**
** Parameters:
**     level :      fifteen detection level
**            ( LVI_Level0 - LVI_Level15 )
**
** Returns:
**     MD_OK
**     MD_ERROR
**
**-----------------------------------------------------------------------------
*/
MD_STATUS LVI_SetLVILevel( enum LVILevel level )
{
       if( LVISEL == 1 ){
              return MD_ERROR;
       }
       LVIS = level;
       return MD_OK;
}
```

4.8 Lvi_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 : lvi_user.c
** Abstract : This file implements device driver for LVI module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
*******************************************************************************
*/
#pragma       interrupt     INTLVI MD_INTLVI
/*
*******************************************************************************
** Include files
*******************************************************************************
*/
#include "macrodriver.h"
```

```
#include "lvi.h"
/* add include file for LEDs */
#include "led_0537.h"
/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     INTLVI Interrupt service routine.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTLVI( void )
{
       LVI_Stop();         /* mask LVI operation */
       led_out_left(0xC7);              /* show an 'L' in left LED */
       led_out_right(LED_PAT_1);  /* show 1 in right LED */
       /* and return */
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function starts the Low-Voltage Detector, for Reset Mode,
**            by not clearing the interrupt mask LVIMK
**            (this routine modified from LVI_Start() as generated by Applilet)
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void LVI_Start_R( void )
{
       USHORT i;
       LVIMK = 1;
       LVIIF = 0;

       LVION = 1;                /* enable LVI */
       for(i=0; i<=100; i++){           /* wait 10 us */
              NOP();
       }
/*     LVIMK = 0; */ /* comment out */
}
```

4.9 Port.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 :port.h
** Abstract :This file implements device driver for PORT module.
** APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler :NEC/CC78K0
**
********************************************************************************
*/
#ifndef      _MDPORT_
#define      _MDPORT_
/*
********************************************************************************
** MacroDefine
********************************************************************************
*/
#define      PORT_PM0     0xff
#define      PORT_PU0     0x0
#define      PORT_P0      0x0
#define      PORT_PM1     0xff
#define      PORT_PU1     0x0
#define      PORT_P1      0x0
#define      PORT_PM2     0xff
#define      PORT_P2      0x0
#define      PORT_PM3     0xff
#define      PORT_PU3     0x6
#define      PORT_P3      0x0
#define      PORT_PM4     0xf0
#define      PORT_PU4     0x0
#define      PORT_P4      0x0
#define      PORT_PM5     0xf0
#define      PORT_PU5     0x0
#define      PORT_P5      0x0
#define      PORT_PM6     0xff
#define      PORT_P6      0x0
#define      PORT_PM7     0x0
#define      PORT_PU7     0x0
#define      PORT_P7      0x0
#define      PORT_PM12    0xff
#define      PORT_PU12    0x0
#define      PORT_P12     0x0
#define      PORT_P13     0x0
#define      PORT_PM14    0xff
#define      PORT_PU14    0x0
#define      PORT_P14     0x0
#define      PORT_ADPC    0x0
```

```
void PORT_Init( void );

#endif
```

4.10 Port.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 : port.c
** Abstract : This file implements device driver for PORT module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
******************************************************************************
*/
/*
******************************************************************************
** Include files
******************************************************************************
*/
#include "macrodriver.h"
#include "port.h"
/*
******************************************************************************
** Constants
******************************************************************************
*/

/*
**----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes the I/O module.
**
** Parameters:
**     None
**
** Returns:
**     None
```

```
**
**-----------------------------------------------------------------------------
*/
void PORT_Init( void )
{
       /* initialize the port registers */
       P0 = PORT_P0;
       P1 = PORT_P1;
       P2 = PORT_P2;
       P3 = PORT_P3;
       P4 = PORT_P4;
       P5 = PORT_P5;
       P6 = PORT_P6;
       P7 = PORT_P7;
       P12 = PORT_P12;
       P13 = PORT_P13;
       P14 = PORT_P14;

       /* initialize the Pull-up resistor option registers */
       PU0 = PORT_PU0;
       PU1 = PORT_PU1;
       PU3 = PORT_PU3;
       PU4 = PORT_PU4;
       PU5 = PORT_PU5;
       PU7 = PORT_PU7;
       PU12 = PORT_PU12;
       PU14 = PORT_PU14;

       /* initialize the mode registers */
       PM0 = PORT_PM0;
       PM1 = PORT_PM1;
       PM2 = PORT_PM2;
       ADPC = PORT_ADPC;
       PM3 = PORT_PM3;
       PM4 = PORT_PM4;
       PM5 = PORT_PM5;
       PM6 = PORT_PM6;
       PM7 = PORT_PM7;
       PM12 = PORT_PM12;
       PM14 = PORT_PM14;
}
```

4.11 Timer.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 : timer.h
** Abstract : This file implements a device driver for the timer module
** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC/CC78K0
**
*******************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_
/*
*******************************************************************************
** MacroDefine
*******************************************************************************
*/
#define       REGVALUE_MAX  0xff
#define       TM_TM00_CLOCK 0x0
#define       TM_TM00_INTERVALVALUE      0x1f3f
#define       TM_TM00_SQUAREWIDTH 0x1f3f
#define       TM_TM00_PPGCYCLE    0x1f3f
#define       TM_TM00_PPGWIDTH    0x00
#define       TM_TM00_ONESHOTCYCLE       0x1f3f
#define       TM_TM00_ONEPULSEDELAY      0x00
#define       TM_TM01_CLOCK 0x0
#define       TM_TM01_INTERVALVALUE      0x00
#define       TM_TM01_SQUAREWIDTH 0x00
#define       TM_TM01_PPGCYCLE    0x00
#define       TM_TM01_PPGWIDTH    0x00
#define       TM_TM01_ONESHOTCYCLE       0x00
#define       TM_TM01_ONEPULSEDELAY      0x00
#define       TM_TM50_CLOCK 0x2
#define       TM_TM50_INTERVALVALUE      0x00
#define       TM_TM50_SQUAREWIDTH 0x00
#define       TM_TM50_PWMACTIVEVALUE     0x00
#define       TM_TM51_CLOCK 0x2
#define       TM_TM51_INTERVALVALUE      0x00
#define       TM_TM51_SQUAREWIDTH 0x00
#define       TM_TM51_PWMACTIVEVALUE     0x00
#define       TM_TMH0_CLOCK 0x0
#define       TM_TMH0_INTERVALVALUE      0x00
#define       TM_TMH0_SQUAREWIDTH 0x00
#define       TM_TMH0_PWMCYCLE    0x00
#define       TM_TMH0_PWMDELAY    0x00
#define       TM_TMH1_CLOCK 0x5
#define       TM_TMH1_INTERVALVALUE      0x11
#define       TM_TMH1_SQUAREWIDTH 0x11
#define       TM_TMH1_PWMCYCLE    0x11
#define       TM_TMH1_PWMDELAY    0x00
#define       TM_TMH1_CARRIERDELAY       0x11
#define       TM_TMH1_CARRIERWIDTH       0x00


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM00_Init(void);
void TMH1_Init(void);
```

```
/*timer start*/
void TM00_Start(void);
void TMH1_Start(void);

/*timer stop*/
void TM00_Stop(void);
void TMH1_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num);
__interrupt void MD_INTTM000(void);
__interrupt void MD_INTTMH1(void);

/* add declaration of g_tmh1_count */
extern unsigned char g_tmh1_count;

#endif       /* _MDTIMER_*/
```

4.12 Timer.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 : timer.c
** Abstract : This file implements a device driver for the timer module
** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC/CC78K0
**
*******************************************************************************
*/

/*
*******************************************************************************
** Include files
*******************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"


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

```
** MacroDefine
******************************************************************************
*/
/*TM00 pulse width measure*/

/*TM01 pulse width measure*/

/*
**----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes TM00_module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void TM00_Init( )
{
       TMC00=0x00;
                           /* internal count clock */
       PRM00 |= TM_TM00_CLOCK;
       SetIORBit(PR0H, 0x40);                 /* low priority level */
       ClrIORBit(IF0H, 0x40);
       /* TM00 interval */
       ClrIORBit(CRC00,0x01);
       CR000 = TM_TM00_INTERVALVALUE;
}
/*
**----------------------------------------------------------------------------
**
** Abstract:
**     This function starts the TM00 counter.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void TM00_Start()
{
       TMC00 = 0x0c;                   /* interval timer start */
       ClrIORBit(MK0H, 0x40);                 /* INTTM000 enable */
}

/*
**----------------------------------------------------------------------------
**
** Abstract:
**     This function stops the TM00 counter and clear the count register.
**
** Parameters:
**     None
**
** Returns:
```

```
**      None
**
**---------------------------------------------------------------------------------
*/
void TM00_Stop()
{
      TMC00=0x0;
      SetIORBit(MK0H, 0x40);                  /* INTTM000 stop */
}

/*
**---------------------------------------------------------------------------------
**
** Abstract:
**      This function changes TM00 condition.
**
** Parameters:
**      USHORT* :     array_reg
**      USHORT :      array_num
** Returns:
**      MD_OK
**      MD_ERROR
**
**---------------------------------------------------------------------------------
*/
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
 switch (array_num){
 case 2:
 CR010=*(array_reg + 1);
 case 1:
 CR000=*(array_reg + 0);
 break;
 default:
 return MD_ERROR;
 }
       return MD_OK;
}

/*
**---------------------------------------------------------------------------------
**
** Abstract:
**      This function initializes TMH1_module.
**
** Parameters:
**      None
**
** Returns:
**      None
**
**---------------------------------------------------------------------------------
*/
void TMH1_Init( void )
{
      ClrIORBit(TMHMD1, 0x80);
                  /* countclock=fr/128 */
      SetIORBit(TMHMD1, 0x50);
      SetIORBit(PR0H, 0x08);                  /* low priority level */
      ClrIORBit(IF0H, 0x08);
      /* TMH1 interval timer */
      ClrIORBit(TMHMD1, 0x8c);
```

```
        CMP01 = TM_TMH1_INTERVALVALUE;
}

/*
**------------------------------------------------------------------------------
**
** Abstract:
**      This function can start the TMH1 counter.
**
** Parameters:
**      None
**
** Returns:
**      None
**
**------------------------------------------------------------------------------
*/
void TMH1_Start( void )
{
        /* TMH1 interval timer */
        SetIORBit(TMHMD1, 0x80);
        ClrIORBit(MK0H, 0x08);                  /* INTTMH1 enable */
}

/*
**------------------------------------------------------------------------------
**
** Abstract:
**      This function can stop the TMH1 counter operation.
**
** Parameters:
**      None
**
** Returns:
**      None
**
**------------------------------------------------------------------------------
*/
void TMH1_Stop( void )
{
        ClrIORBit(TMHMD1, 0x80);
        SetIORBit(MK0H, 0x08);                  /* INTTMH1 disable */
}

/*
**------------------------------------------------------------------------------
**
** Abstract:
**      This function can change TMH1 condition.
**
** Parameters:
**      UCHAR* :        array_reg
**      UCHAR :         array_num
** Returns:
**      MD_OK
**      MD_ERROR
**
**------------------------------------------------------------------------------
*/
MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num)
{
 switch (array_num){
```

```
 case 2:
 CMP11=*(array_reg + 1);
 case 1:
 CMP01=*(array_reg + 0);
 break;
 default:
 return MD_ERROR;
 }
        return MD_OK;
}
```

4.13 Timer_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 : timer_user.c
** Abstract : This file implements a device driver for the timer module
** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler: NEC/CC78K0
**
******************************************************************************
*/

#pragma sfr
#pragma interrupt INTTM000 MD_INTTM000
#pragma interrupt INTTMH1 MD_INTTMH1
/*
******************************************************************************
** Include files
******************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

/* add include for switch routines */
#include "sw_0537.h"
/* add include for watchdog timer routines */
#include "watchdogtimer.h"

/*
```

```
******************************************************************************
** MacroDefine
******************************************************************************
*/

/* Timer00, Timer01 pulse width measure */
/*
**----------------------------------------------------------------------------
**
** Abstract:
**     TM00 INTTM000 interrupt service routine
**
** Parameters:
**     None
**
** Returns:
**     None
**
**----------------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
      EI();  /* allow higher priority interrupts (INTTMH1, INTLVI) to occur */
      /* call switch ISR routine for debouncing switches */
      sw_isr();
}
/*
**----------------------------------------------------------------------------
**
** Abstract:
**     TMH1 INTTMH1 interrupt service routine
**
** Parameters:
**     None
**
** Returns:
**     None
**
**----------------------------------------------------------------------------
*/
/* global variable incremented by TMH1 interrupt */
unsigned char g_tmh1_count;

__interrupt void MD_INTTMH1( )
{
      EI();  /* allow higher priority interrupts (INTLVI) to occur */
      WDT_Restart();
      g_tmh1_count = g_tmh1_count + 1;
}
```

4.14 Watchdogtimer.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 : watchdogtimer.h
** Abstract : This file implements device driver for WDT module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
********************************************************************************
*/

#ifndef       _MDWTACHDOGTIMER_
#define       _MDWTACHDOGTIMER_
/*
********************************************************************************
** MacroDefine
********************************************************************************
*/
void WDT_Init( void );
void WDT_Start( void );
void WDT_Restart( void );

#endif
```

4.15 Watchdogtimer.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 : watchdogtimer.c
** Abstract : This file implements device driver for WDT module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
```

```
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
*******************************************************************************
*/

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

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes the Watchdog timer module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void WDT_Init( void )
{
       /* overflow time: 2^14/frl (62.06ms) */
       /* window open period: 100% (before overflow time) */
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function starts watchdog timer.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void WDT_Start( void )
{
       WDTE = 0xac;
}

/*
**-----------------------------------------------------------------------------
**
```

```
** Abstract:
**      This function clears and starts watchdog timer again.
**
** Parameters:
**      None
**
** Returns:
**      None
**
**-----------------------------------------------------------------------------
*/
void WDT_Restart( void )
{
        WDTE = 0xac;
}
```

4.16 Led_0537.h

```
/* led_0537.h                                                                    */
/*      header for M-78F0537 CPU board for LED digit display */
/* Version 1.1      01-13-2006                                                   */

#ifndef _LED_0537_H
#define _LED_0537_H

/*******************************************************************/
/* Define definitions */
/*******************************************************************/

/* LED Patterns for decimal and hex digits, characters */
/* for individual bits,     ---A---      */
/* 0=on 1=off                     | |    */
/* bit 0 = segment A              F B    */
/* bit 1 = segment B              | |    */
/* bit 2 = segment C               ---G---      */
/* bit 3 = segment D              | |    */
/* bit 4 = segment E              E C    */
/* bit 5 = segment F              | |    */
/* bit 6 = segment G               ---D--- DP   */
/* bit 7 = decimal point                        */

#define LED_PAT_0 0xC0
#define LED_PAT_1 0xF9
#define LED_PAT_2 0xA4
#define LED_PAT_3 0xB0
#define LED_PAT_4 0x99
#define LED_PAT_5 0x92
#define LED_PAT_6 0x82
#define LED_PAT_7 0xF8
#define LED_PAT_8 0x80
#define LED_PAT_9 0x98
#define LED_PAT_A 0x88
#define LED_PAT_B 0x83
#define LED_PAT_C 0xC6
```

```
#define LED_PAT_D 0xA1
#define LED_PAT_E 0x86
#define LED_PAT_F 0x8E
#define LED_PAT_BLANK      0xFF
#define LED_PAT_DP         0x7F
#define LED_PAT_DASH       0xBF
#define LED_PAT_ULINE      0xF7
#define LED_PAT_OLINE      0xFE
#define LED_PAT_EQUAL      0xB7

/**********************************************************************/
/* Export functions */
/**********************************************************************/
extern void led_init(void);                                           /* init ports for LED
output */
extern void led_out_right(unsigned char val);  /* output value to right LED     */
extern void led_out_left(unsigned char val);   /* output value to left LED             */
extern void led_dig_right(unsigned char num);  /* display number in right LED   */
extern void led_dig_left(unsigned char num);   /* display number in left LED    */
extern void led_dig(unsigned char num);              /* display number as hex          */
extern void led_dig_bcd(unsigned char bcdnum); /* display number as BCD         */

#endif /* _LED_0537_H */
```

4.17 Led_0537.c

```
/*      led_0537.c - routines for LED display                         */
/*             for M-78F0537 CPU board on M-Station base board        */
/* Version:   1.1    01-13-2006                                       */

/*             P70-P77 = output to right digit (LED2)                 */
/*             P40-P43 = output to left digit (LED1) bits 0-3 */
/*             P50-P53 = output to left digit (LED1) bits 4-7 */

/*      To connect ports to LEDs on M-Station 1.1, make the
        following jumper connections between ROW1 and ROW2.
        To connect ports to LEDs on M-Station 2, make sure
        the default SBxx connections are inserted.
        Port   LED          M-Station 1.1 M-Station 2.2
        ----   ---          ------------- ------------
        P70          2-A          R1.25 - R2.25 SB27
        P71          2-B          R1.26 - R2.26 SB28
        P72          2-C          R1.27 - R2.27 SB29
        P73          2-D          R1.28 - R2.28 SB30
        P74          2-E          R1.29 - R2.29 SB31
        P75          2-F          R1.30 - R2.30 SB32
        P76          2-G          R1.31 - R2.31 SB33
        P77          2-DP   R1.32 - R2.32 SB34

        P40          1-A          R1.17 - R2.17 SB35
        P41          1-B          R1.18 - R2.18 SB36
        P42          1-C          R1.19 - R2.19 SB37
        P43          1-D          R1.20 - R2.20 SB38
```

```
       P50           1-E           R1.21 - R2.21 SB39
       P51           1-F           R1.22 - R2.22 SB40
       P52           1-G           R1.23 - R2.23 SB41
       P53           1-DP   R1.24 - R2.24 SB42
*/

/* NOTE: on M-Station Base V1.0 prototype, P40-P53 are      */
/* located at ROW4.1-8, and need to be wirewrapped to */
/* connect to ROW2.17-24 to drive LED1.                     */

/* need pragma declaration to access SFR's in C      */
#pragma sfr

#include "led_0537.h"

/* table of bit patterns for seven-segment digits */
static unsigned char dig_tab[] = {
 LED_PAT_0,  /* 0 */
 LED_PAT_1,  /* 1 */
 LED_PAT_2,  /* 2 */
 LED_PAT_3,  /* 3 */
 LED_PAT_4,  /* 4 */
 LED_PAT_5,  /* 5 */
 LED_PAT_6,  /* 6 */
 LED_PAT_7,  /* 7 */
 LED_PAT_8,  /* 8 */
 LED_PAT_9,  /* 9 */
 LED_PAT_A,  /* A */
 LED_PAT_B,  /* B */
 LED_PAT_C,  /* C */
 LED_PAT_D,  /* D */
 LED_PAT_E,  /* E */
 LED_PAT_F   /* F */
};

/*     void led_init(void) */
/*             set up ports for display of LED digits */
void led_init(void)
{
#if 0  /* ports initialized in Port_Init() by Applilet */
       PM7 = 0x00;         /* set all port 7 to output */
       PM4 = 0x00;         /* set all port 4 to output */
       PM5 = 0x00;         /* set all port 5 to output */
#endif
}

/* void led_out_right(unsigned char val) */
/*             output raw data to right LED */
void led_out_right(unsigned char val)
{
       P7 = val;
}

/* void led_out_left(unsigned char val) */
/*             output raw data to left LED */
void led_out_left(unsigned char val)
{
       P4 = val & 0x0F;
       P5 = (val >> 4) & 0x0F;
}

/* void led_dig_right(unsigned char num) */
/*     display number in right LED */
void led_dig_right(unsigned char num)
{
       if (num > 0x0F) {
              led_out_right(LED_PAT_BLANK);
              return;
       }
       led_out_right(dig_tab[num]);
}

/* void led_dig_left(unsigned char num) */
/*     display number in left LED */
void led_dig_left(unsigned char num)
{
       if (num > 0x0F) {
              led_out_left(LED_PAT_BLANK);
              return;
       }
       led_out_left(dig_tab[num]);
}

/* void led_dig(unsigned char num) */
/*            display number as hex digits */
/*            num - number to display */
/*                  bits 0-3 in right digit */
/*                  bits 4-7 in left digit */
void led_dig(unsigned char num)
{
       led_out_right(dig_tab[num & 0x0F]);
       led_out_left(dig_tab[(num >> 4) & 0x0F]);
}

/* void led_dig_bcd(unsigned char bcdnum) */
/*            display two digits of BCD coded bcdnum */
/*            bcdnum - number to display in BCD */
/*                  0 - 9 displayed as right decimal digit, left blank */
/*                  10 - 99 displayed as two decimal digits */
/*                  100 - 255 displayed as blank */
void led_dig_bcd(unsigned char bcdnum)
{
unsigned char tens_dig;

       if (bcdnum > 99) {
              led_out_right(LED_PAT_BLANK);    /* display both digits blank */
              led_out_left(LED_PAT_BLANK);
              return;
       }

       if (bcdnum < 10) {
              led_out_right(dig_tab[bcdnum]);  /* just display right LED */
              led_out_left(LED_PAT_BLANK);            /* blank left LED */
              return;
       }

       /* 10 <= bcdnum <= 99 */
       tens_dig = 0;
       do {                        /* calculate ten's place and remainder */
              bcdnum -= 10;/* by multiple subtractions of 10 */
              tens_dig++;          /* while counting up the tens digit */
       } while (bcdnum >= 10);
       /* now tens_dig has ten's place */
       /* and bcdnum has remainder */
       led_out_right(dig_tab[bcdnum]);
       led_out_left(dig_tab[tens_dig]);
}
```

4.18 Sw_0537.h

```
/* sw_0537.h */
/*     header for M-78F0537 CPU board for base board switch reading */
/* Version:  1.1    01-13-2006    */

#ifndef _SW_0537_H
#define _SW_0537_H

/*********************************************************************/
/* Define definitions */
/*********************************************************************/

/* symbolic definitions for switch inputs */
/* SW2 = left switch = P31 */
/* SW3 = right switch = P32 */
/*                                                                   P32
             P31 */
#define      SW_LU_RU     0x06   /* left up, right up       1          1     */
#define SW_LD_RU    0x04   /* left down, right up     1          0     */
#define SW_LU_RD    0x02   /* left up, right down     0          1     */
#define SW_LD_RD    0x00   /* left down, right down 0       0     */

#define      SW_DEF_DEB_COUNT    8     /* default debounce counter      */

/*********************************************************************/
/* Export functions */
/*********************************************************************/
extern void sw_init(void);             /* init ports and variables for switch input */
extern unsigned char sw_chk(void);     /* get undebounced switch input */
extern unsigned char sw_get(void);     /* get debounced switch input */
extern void sw_set_debounce(unsigned char count);    /* set deboune cound */
extern void sw_isr(void);              /* debounce routine, called by timer ISR */

#endif /* _SW_0537_H */
```

4.19 Sw_0537.c

```
/*     sw_0537.c - routines for switch input                     */
/*           for M-78F0537 CPU board on M-Station base board      */
/* Version:  1.1    01-13-2006                                    */
/*          P31 = input for left switch (SW2)                      */
/*          P32 = input for right switch (SW3)                            */

/*    To connect ports to switches on M-Station 1.1, make the
      following jumper connections between ROW1 and ROW2.
      To connect ports to swtiches on M-Station 2, make sure
      the default SBxx connections are inserted.

      Port   Switch M-Station 1.1 M-Station 2.2
      ----   ---          -------------------------
      P31          SW2          R1.5 - R2.5         SB7
      P32          SW3          R1.6 - R2.6         SB8
*/

/* need pragma declaration to access SFR's in C      */
#pragma sfr

#include "sw_0537.h"

/* local variables for switch handling */
static unsigned char sw_last;           /* last debounced switch value */
static unsigned char sw_new;            /* new value being debounced */
static unsigned char sw_deb_value;      /* value of debounce counter */
static unsigned char sw_deb_count;      /* debounce counter */

/*    void sw_init(void) */
/*          set up ports for switch input */
void sw_init(void)
{
#if 0  /* initialization done in Port_Init() by Applilet */
      /* set P31 and P32 to inputs */
      PM3.1 = 1;
      PM3.2 = 1;
      /* set pullups on P31 and P32 */
      PU3.1 = 1;
      PU3.2 = 1;
#endif
      /* set static variables */
      sw_last = SW_LU_RU; /* default is right up, left up (no switch pressed) */
      sw_deb_value = SW_DEF_DEB_COUNT; /* set default debounce counter value */
      sw_deb_count = SW_DEF_DEB_COUNT; /* set counter to max */
}

/* unsigned char sw_chk(void) */
/*    return input from switches, undebounced */
unsigned char sw_chk(void)
{
      return P3 & 0x06;
}

/* void sw_set_debounce(unsigned char count) */
/*    set the debounce counter value */
void sw_set_debounce(unsigned char count)
{
      sw_deb_value = count;     /* set new debounce counter value */
      sw_deb_count = count;     /* set counter to max */
}

/* unsigned char sw_get(void) */
/*    return debounced switch input */
```

```
unsigned char sw_get(void)
{
      return sw_last;
}


/* void sw_isr( void ) */
/* this routine called by periodic timer interrupt to poll and debounce switches */
/* after a new value has been seen steadily for sw_deb_value times, sw_last is updated */
void sw_isr( void )
{
unsigned char val;

      val = sw_chk();            /* get current value */
      /* if value is the same as before, no change; reset debounce and return */
      if (val == sw_last) {
            sw_deb_count = sw_deb_value;     /* reset debounce counter to max */
            return;
      }

      /* val != sw_last, there is a new input */
      /* if it's not the same as the previous new one, */
      /* set the NEW new one, reset the debounce counter and return */
      if (val != sw_new) {
            sw_new = val;
            sw_deb_count = sw_deb_value;
            return;
      }

      /* val != sw_last, val == sw_new */
      /* count down the debounce counter */
      sw_deb_count--;

      /* if we have counted down to zero, we have seen the same sw_new */
      /* for debounce count times, it is now the debounced switch value */
      if (sw_deb_count == 0) {
            sw_last = val;
            sw_deb_count = sw_deb_value;
            return;
      }

      /* if still debouncing, just return */
      return;
}
```

4.20 Option.inc

Note: this version is for 1.59V POC mode.

```
;*****************************************************************************
```

```
;**
;** 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
OPTION_BYTE_WDT     EQU    078H
```

4.21 Option.asm

Note: this version is for 1.59V POC mode.

```
;*******************************************************************************
;**
```

```
;** 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 + OPTION_BYTE_WDT;
       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
```

4.22 Option_POC27.inc

Note: this version is for 2.7V1.59V POC mode.

```
;*******************************************************************************
;**
;** 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
; original POC81 byte as generated by Applilet
; had POC81.0 = 0 (POCMODE) to select 1.59V POC mode
; for SAFETY_POC27 project, changed value of POC81 to 01H
; so POC81.0 = 1 (POCMODE) to select 2.7V/1.59V POC mode
;POC81 EQU    00H
POC81  EQU    01H
POC82  EQU    01H
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
OPTION_BYTE_WDT     EQU    078H
```

4.23 Option_POC27.asm

Note: this version is for 2.7V1.59V POC mode.

```
;*******************************************************************************
;**
;** 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
;*******************************************************************************
; changed option.asm as originally generated by Applilet
; to option_POC27.asm, to include option_POC27.inc
; for SAFETY_POC27 project
;$ INCLUDE (option.inc)
$ INCLUDE (option_POC27.inc)
      OPT_SET CSEG AT 80H
OPTION:       DB     OPTION_BYTE + OPTION_BYTE_WDT;
      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
```