NEC

Application Note

Safety Features

For NEC Electronics 32-Bit Microcontrollers

Document No. U19024EU1V0AN00
©November 2007. NEC Electronics America, Inc.
All rights reserved.

The information in this document is current as of November 2007. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.

No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such NEC Electronics products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC Electronics no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

NEC Electronics products are classified into the following three quality grades: “Standard”, “Special” and “Specific”.

The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated “quality assurance program” for a specific application. The recommended applications of NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.

"Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

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

Notes:

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

M8E 02.10

Revision History

Date	Revision	Section	Description
November 2007	—	—	First release

Contents

1. Introduction 7

1.1 Overview of Safety Features 7

2. Reset Sources 8

2.1.1 Power-On Clear (POC) 9
2.1.2 Low-Voltage Detect (LVI) 10
2.1.3 Watchdog Timer 12
2.1.4 Clock Monitor 14

3. Program Description and Specification 15

3.1.1 External Reset 16
3.1.2 Power-On Clear 16
3.1.3 Low-Voltage Detect 16
3.1.4 Watchdog Timer 16
3.1.5 Clock Monitor 16
3.2 Software Flow Charts 17
3.3 Applilet's Reference Drivers 17
3.4 Demonstration Platform 17
3.4.1 Resources 17
3.4.2 Demonstration Program 18
3.5 Hardware Block Diagram 22
3.6 Software Modules 22

4. Appendix A — Software Flowcharts 24

4.1 Main() 24
4.2 Voltage Reset 27
4.3 Voltage Interrupt 28
4.4 LVD monitor 29
4.5 Watchdog Timer Reset 30
4.6 Watchdog Timer Interrupt 31
4.7 Watchdog Timer Monitor 32
4.8 Clock Monitor 34
4.9 System Initialize 35
4.10 System Initialize Hardware 36
4.11 Port Initialize 37
4.12 Port User 38
4.13 Port User LED 41
4.14 Watchtimer Initialize 47
4.15 Watchtimer User 49
4.16 Watchdog 1 Initialize 50
4.17 Watchdog 1 User 51
4.18 LVI Initialize 52
4.19 LVI User 54
4.20 Write Special 55
4.21 Pre-Processing 58

5. Appendix B – Applilet Tool 61

5.1 System Memory 61
5.2 System Peripherals 62
5.3 System 63
5.3.1 System Foundation Settings 63
5.3.2 System Startup Settings 64
5.3.3 System Watchdog Settings 65
5.4 Port Subsystem 65
5.4.1 Port 9-1 Selections 66
5.4.2 Port DH Selections 67
5.4.3 Port DL-2 Selections 68
5.5 Watchdog Timer 1 Selections 69
5.6 Low-Voltage Detect Selections 69
5.7 Timer Subsystem 70
5.7.1 Timer TMP0 Selections 71
5.8 Watch Timer Selections 72
5.9 Interrupt Selections 73
5.10 Generation of Source files 73
5.10.1 Interrupt and Port Functions 74
5.10.2 Timer Functions 75
5.10.3 Watchdog, Watch Timer and LVI Functions 76
5.10.4 Source File List 77

6. Appendix C – Source Code Listings 78

6.1 crtE.s 78
6.2 systeminit.c 82
6.3 main.c 84
6.4 int.c 92
6.5 int_user.c 94
6.6 LVI.s 96
6.7 LVI_user.c 99
6.8 port.c 101
6.9 port_user.c 104
6.10 timer.c 111
6.11 timer_user.c 116
6.12 watchdogtimer.c 120
6.13 watchdogtimer_user.c 122
6.14 watchtimer.c 124
6.15 watchtimer_user.c 127
6.16 write_special.s 130
6.17 int.h 134
6.18 lvi.inc 135
6.19 macrodriver.h 136
6.20 port.h 138
6.21 timer.h 142
6.22 watchdogtimer.h 145
6.23 watchtimer.h 146

1. Introduction

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

This document includes the following information:

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

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

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

1.1 Overview of Safety Features

The safety features built into NEC Electronics microcontrollers enable you to reset the system at the start of operation or restart from a reset condition and find out the cause of the reset. When the supply voltage sags below a specified level, for example, the microcontroller's low-voltage detect circuit detects the condition and interrupts the system so that the system's state can be saved. If a program goes into an infinite loop, the microcontroller's watchdog timer resets the system. If the system clock oscillator stops, a backup oscillator can take over. These features are crucial for keeping the system running in a safe condition.

This application note covers the following safety features:

- Reset by external reset pin
- Power-on-clear reset (POC)
- Low-voltage detect (LVI) reset or interrupt
- Clock monitor (CLM)
- Watchdog timer (WDT) reset

This document provides a short overview of these features and demonstrates their use, using the NEC Electronics V850ES-KJ1+™ microcontroller as an example.

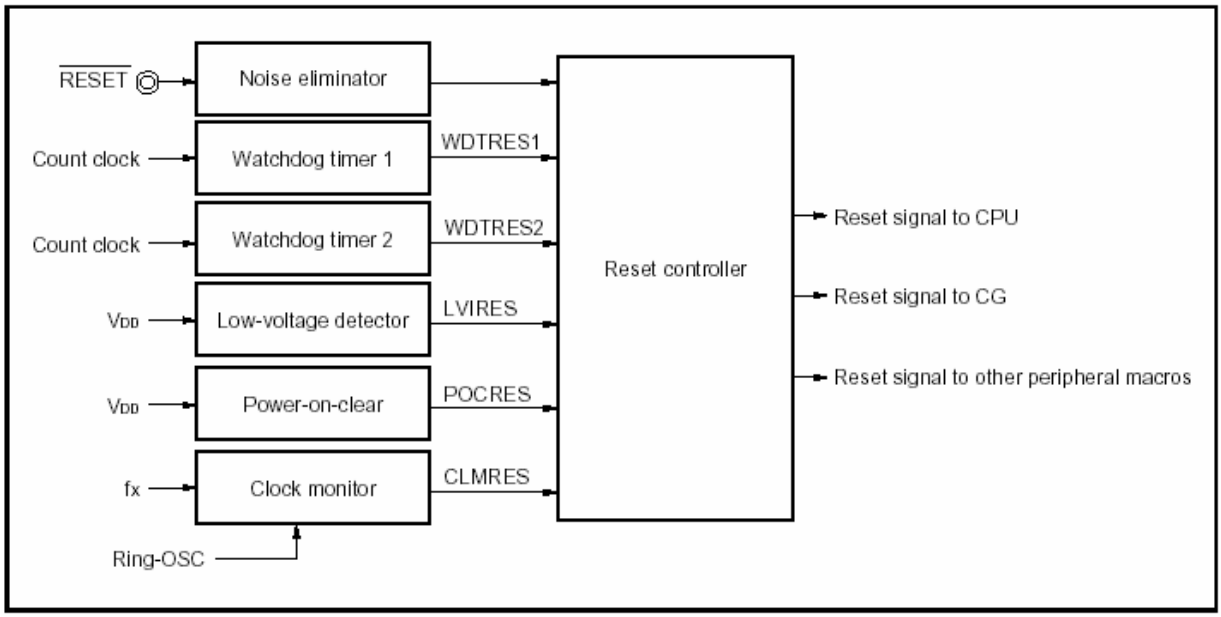
2. Reset Sources

The following hardware resources generate a reset signal:

- External reset input pin
- Internal reset by watchdog timer overflow
- Internal reset by power-on reset signal
- Internal reset by low-voltage detection circuit
- Internal clock monitor

The external and internal reset signals have no functional differences. In both cases, when the reset signal is asserted, program execution starts at the address specified by the reset vector (which resides at memory location 00000000H). A reset assertion sets the I/O registers and port pins to their default values to ensure a known state of operation.

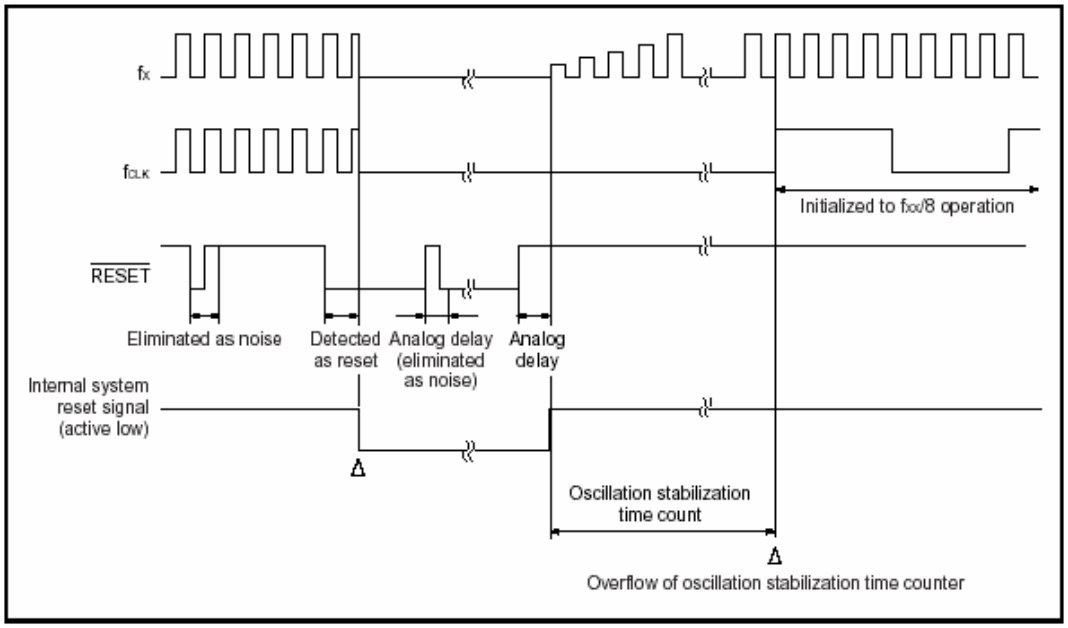
Figure 1. Reset Sources



The reset source flag register (RESF) indicates the source from which the reset signal is generated. This register can be read or written in 8- or 1-bit units; only 0 can be written to this register. A RESET_B input or power-on clear (POCRES) clears this register to 00h.

The diagram below shows the timing of the external reset pin and internal reset signal. When reset is asserted or released, there is a delay to avoid repeated internal resets due to noise on the reset pin. Once the internal reset is released, the CPU starts program execution after the main clock oscillator stabilization time is due.

Figure 2. External Reset Timing



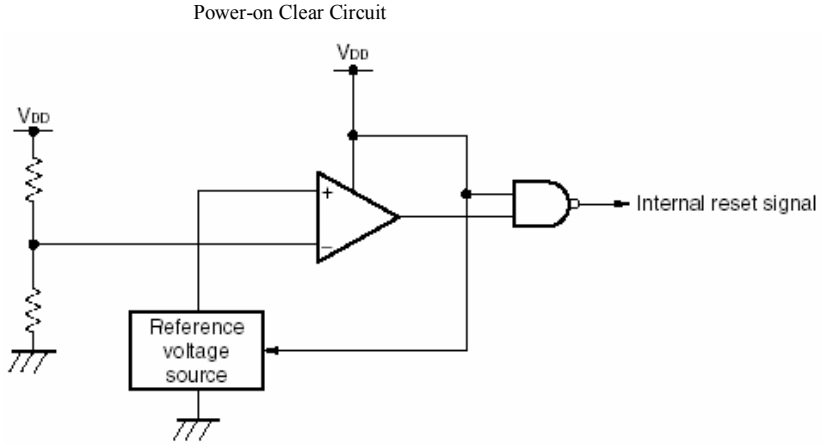
2.1.1 Power-On Clear (POC) Circuit

The power-on clear function offers the following features:

- Holds device in reset while supply voltage V_{DD} is stabilizing on power up to POC voltage Vpoc = 2.6V ± 0.1V
- Resets device if supply voltage V_{DD} falls to a level lower than Vpoc which may cause unstable operation

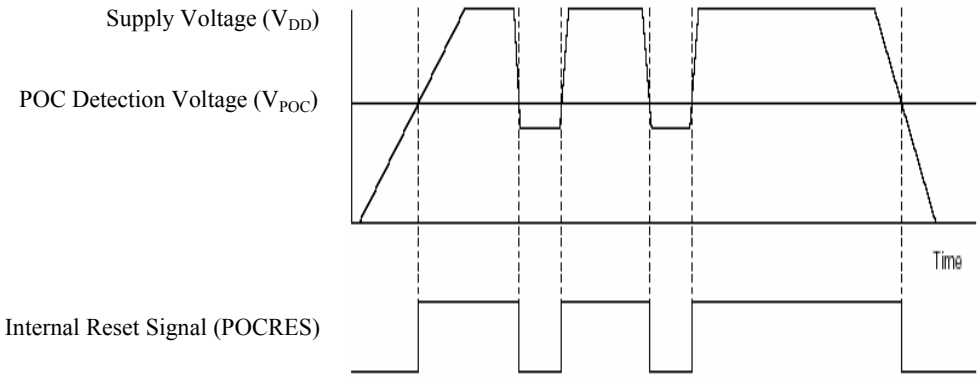
The power-on clear circuit generates an internal reset (POCRES) at power-up of V_{DD}. The internal reset signal is released when the power supply voltage exceeds V_{DD} = 2.6V ± 0.1V.

Figure 3. Power-On Clear Circuit



In the power-on clear circuit, the supply voltage (V_{DD}) and the detection voltage (Vpoc) are compared. An internal reset signal is generated when V_{DD} falls below the detection voltage. The reset is released when the supply voltage rises above the detection voltage.

Figure 4. Reset vs Voltage Detection



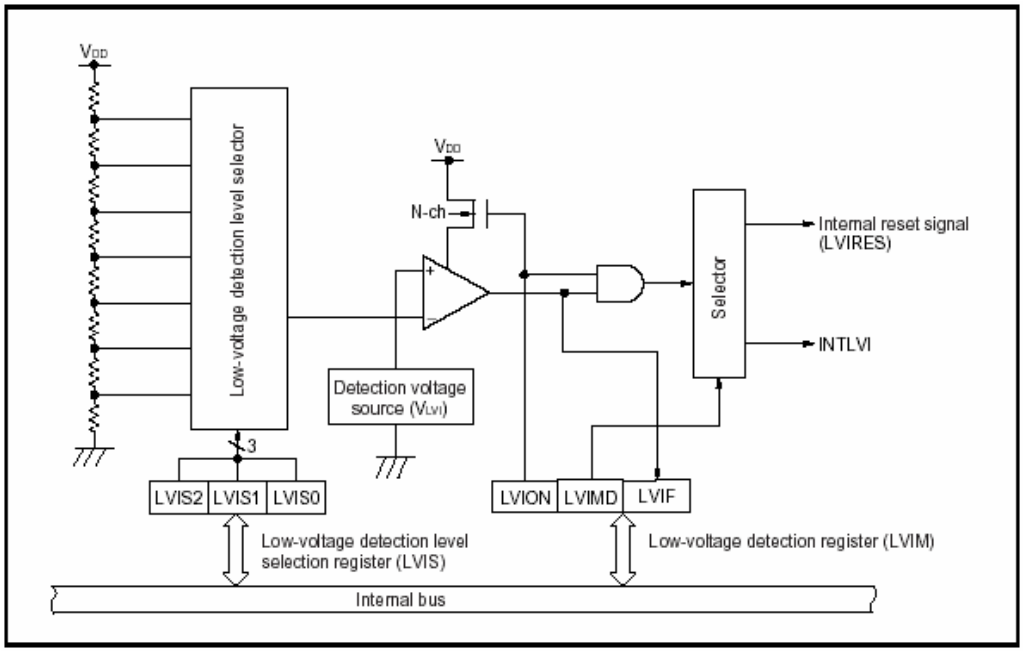
2.1.2 Low-Voltage Detector (LVI)

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

- Selectable reset or interrupt operation for low-voltage condition
- Can check V_{DD} voltage
- For V_{DD} voltage detection, one of seven voltage levels can be set

The low-voltage detector circuit compares a voltage against an internal reference and generates an internal interrupt signal or internal reset signal when the voltage is lower than the set low-voltage range. The V_{DD} voltage is compared by use of a tap on a resistor divider set so a selectable voltage level can be detected.

Figure 5. Low-Voltage Detection



The table below lists the registers controlling low-voltage detection.

Table 1. Low-Voltage Detect Registers

Registers	Symbol	Description of Functions
Low-Voltage Detection Register	LVIM	Sets Low-Voltage Detection and Operation Mode
		Enable/Disable Low-Voltage Detection
		Set Low-Voltage Operation Mode and Flag
Low-Voltage Detection Level Selection Register	LVIS	Selects Low-Voltage Detection Level up to 7 levels

To configure the low-voltage detector function for operation in reset mode, use the following steps:

- Set the LVIMK bit in the LVIIC register to mask the LVI interrupt.
- Clear the LVIIF interrupt flag in the LVIIC register.
- Disable the LVI detector by clearing the LVION bit in the LVIM register.
- If using V_{DD} check mode, set the voltage level in the LVIS register.
- Set the LVIMD bit to zero to select interrupt mode temporarily to avoid triggering reset.
- Turn on the LVION bit to enable the detector.
- Wait a minimum of 200 microseconds for the detector to stabilize.
- Set the LVIMD bit to 1 to select reset mode.

When using interrupt mode rather than reset mode, use these steps:

- Set the LVIMK bit in the LVIIC register to mask the LVI interrupt.
- Clear the LVIIF interrupt flag in the LVIIC register.

- Disable the LVI detector by clearing the LVION bit in the LVIM register.
- If using V_{DD} check mode, set the voltage level in the LVIS register.
- Set the LVIMD bit to zero 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 200 microseconds for the detector to stabilize.
- Clear the LVIMK bit to zero to enable the LVI interrupt.

2.1.3 Watchdog Timer

The watchdog timer (WDT) function offers the following features:

- Reset of the processor for runaway program loops
- Operation on internal oscillator to function even if 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
- Reset on fetch or read/write of illegal memory areas

When enabled and running, the watchdog timer counter must be cleared periodically. Otherwise, a counter overflows and generates an internal reset signal or a non maskable interrupt.

Some V850ES Series microcontrollers (including the V850ES-KJ1+) incorporate two watchdog timers. One watchdog timer (Watchdog Timer 1) operates with the system clock selected by the watchdog timer clock select (WDCS) register. Watchdog Timer 1 operates as either a watchdog timer or interval timer. It generates the following signals upon overflow of the watchdog timer:

- A non-maskable interrupt request signal (INTWDT1)
- A maskable interrupt signal (INTWDTM1)
- System reset request signal (WDTRES1)

The second watchdog timer (Watchdog Timer 2) operates from the microcontroller's ring oscillator or subclock. Upon overflow of the watchdog timer, Watchdog Timer 2 generates the following signals:

- System reset request (WDTRES2)
- A non-maskable interrupt request (INTWDT2)

Note that a system reset request (WDTRES2) is the default starting state, and that this state is operating on start up. When you are not using watchdog timer 2, either stop its operation before it initiates a reset, or clear it once and stop it within the next interval time.

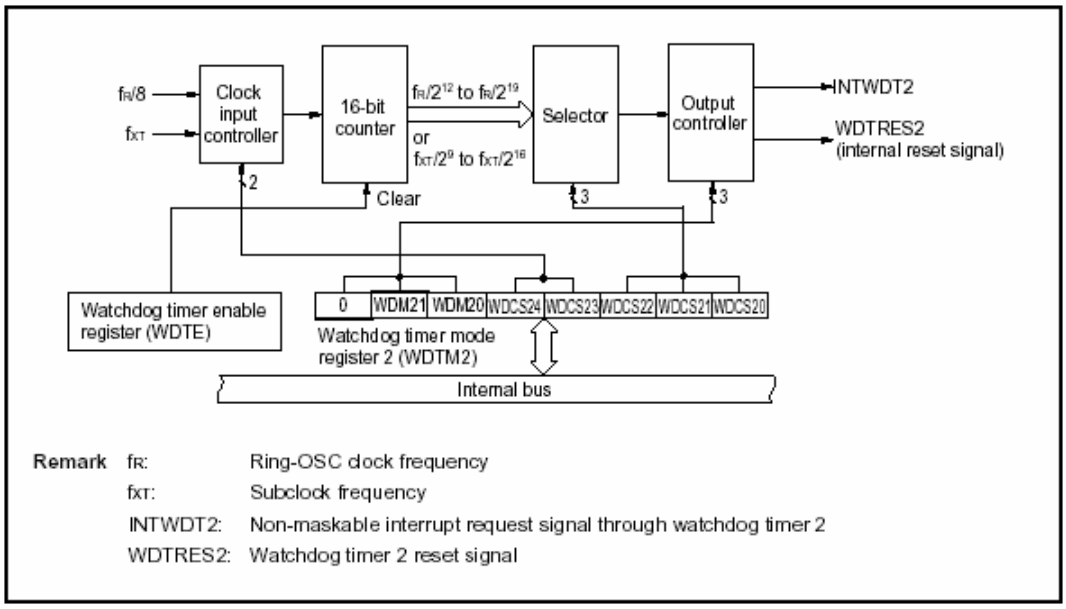
The registers controlling Watchdog Timer 2 are:

- Watchdog timer mode register 2 (WDTM2)
- Watchdog timer enable register (WDTE)

The watchdog timer mode register (WDTM2) selects the operation clock and sets the overflow time. You can read or write the WDTM2 in 8-bit units.

When using the watchdog timer, your program must clear the watchdog timer counter by writing a special value (hexadecimal value AC) to the watchdog timer enable register (WDTE). This write must be done periodically in the main loop of a program or in any portion of code before the execution time exceeds the overflow time of the watchdog timer.

Figure 6. Watchdog Timer 2



Depending on the watchdog timer, you start the watchdog timer by writing to a register or setting a bit in a register. Once the watchdog timer is running, it cannot be disabled, thus preventing a runaway program from accidentally disabling the timer. In addition, you can select a mode by using a flash memory OPTION byte in which software cannot disable the ring-oscillator, thus preventing the source of the Watchdog Timer 2 clock from being stopped.

You clear the Watchdog Timer 2 counter by writing a specific value to the WDTE register. In the case of the V850ES Series microcontrollers, this value is ACh. If any other value is written to WDTE, a reset occurs, protecting against a program loop which writes the wrong value to the WDTE register.

After the count operation starts, the program must write ACh to the WDTE register within the set program loop detection time interval. If the program loop detection time is exceeded without ACh being written to

the WDTE register, the microcontroller generates a system reset request (WDTRES2) or a non-maskable interrupt request (INTWDT2), depending on the set value of the watchdog timer mode register (WDTM2).

2.1.4 Clock Monitor

The clock monitor samples the main clock, using the on-chip ring-oscillator clock as a reference. The monitor generates a reset signal (CLMRES) when the oscillation of the main clock stops. The CPU then operates on the ring-oscillator clock after reset is released. Once the clock monitor has been started (enabled by the CLME bit in the CLM Register), only a reset can stop the monitor. However, the clock monitor automatically stops under the following conditions:

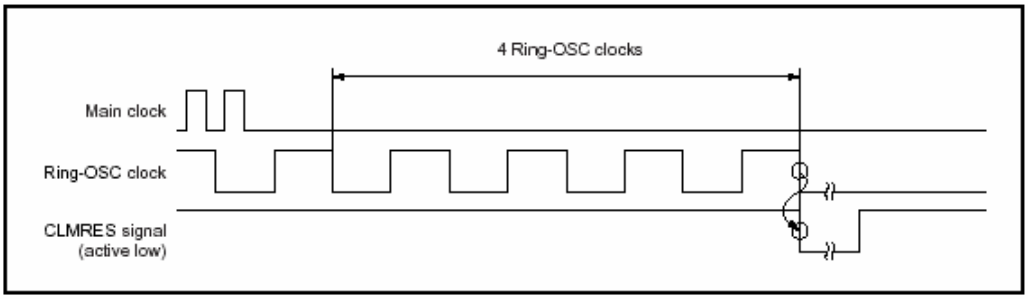
- When the oscillation stabilization time is counted after STOP mode is released
- When the processor clock control (PCC) register stops the main clock and the CPU is running with the subclock
- When the sampling clock ring oscillator stops
- When the CPU operates on the ring oscillator

The following registers control the clock monitor:

- Clock monitor mode (CLM) register
  - Enables or disables the clock monitor
  - Can be read or written in 8- or 1-bit units
- ring-oscillator mode register
  - Sets the operation mode of the ring oscillator
  - Can be read or written in 8- or 1-bit units

If the oscillation of the main clock stops while the clock monitor is enabled, the microcontroller generates a system reset request signal (CLMRES), as shown below.

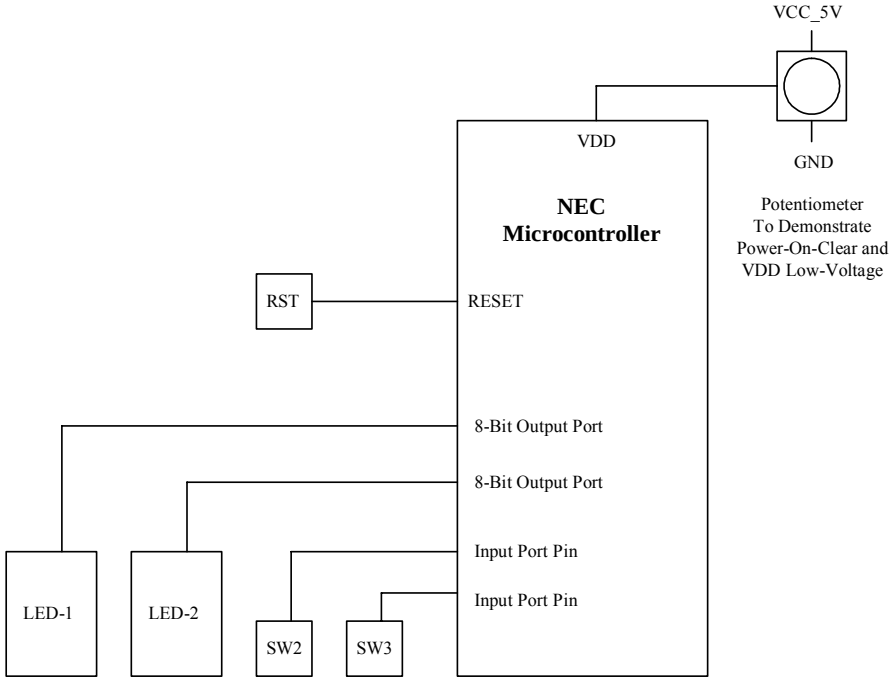
Figure 7. Clock Monitor Reset



3. Program Description and Specification

To demonstrate reset, power-on clear, low-voltage detect, and watchdog timer operation, connect an NEC microcontroller to a reset (RST) switch, other switches for input, a 2-digit 7-segment LED display for output, and variable voltage controls for VDD. The adjustable VDD voltage is implemented by connecting the tap of a potentiometer to the microcontroller's VDD power. As you turn the potentiometer, the voltage at the tap varies between 0V and 5V, allowing you to check low-voltage operation. A block diagram of the demonstration hardware appears below.

Figure 8. Demonstration Hardware Block Diagram



On start up, the demonstration program reads the RESF register and displays the value on the LEDs. This arrangement allows you to see the cause of the previous reset.

The specifications for the demonstration are as follows:

- The POC voltage (Vpoc) is set to 2.6V.
- LVI is set initially for interrupt on VDD below 3.93V.
- The demonstration shows both LVI reset and LVI interrupt operation.
- The watchdog timer is set for 68 millisecond overflow time.
- Timer H1 is used for periodic interrupt for watchdog timer clear, with variable interval.
- Input switches are debounced using Timer 00 and considered stable after 10 milliseconds.

3.1.1 External Reset

To demonstrate external reset, press the reset switch, which grounds the microcontroller's reset input.

3.1.2 Power-On Clear

To demonstrate power-on clear, set the potentiometer so that V_{DD} is below V_{POC} (2.6V). When you apply power, the device stays in reset, and the LEDs are blank because they are not driven. Turning the potentiometer to raise V_{DD} above V_{POC} brings the processor out of reset, and the LEDs show the RESF flag value; this value is 00 for a power-on-clear reset.

Reducing V_{DD} below V_{POC} causes a power-on clear, and the LEDs are again blank. Increasing V_{DD} back above V_{POC} releases the power-on-clear reset, and the LEDs again display the RESF value 00.

3.1.3 Low-Voltage Detector

To demonstrate low-voltage detector (LVI), use the potentiometer to adjust V_{DD}. After the LEDs show the RESF value, you can select one of several tests showing LVI operation by pressing SW2 to select a test, then SW3 to execute the test.

To demonstrate V_{DD} checking, the program sets one of two V_{DD} voltage levels, and sets up the LVI unit for reset or interrupt operation. Reducing V_{DD} below the V_{DD} set point causes a reset (blanking the display) or an interrupt (displaying "LI"). When you increase V_{DD}, the reset clears, and the RESF flag display shows LVI as the cause of the reset.

3.1.4 Watchdog Timer

To demonstrate the watchdog timer, the program sets up a timer for a periodic interrupt to clear the watchdog timer before overflow. Select the watchdog timer demo test, and the LEDs show a blinking value. The speed of blinking and the value shown represent the frequency of the interrupt.

Repeatedly pressing SW3 increases the interval of the timer interrupt, causing the LEDs to display a higher value and blink at a slower rate. When the interval exceeds the watchdog timer overflow time, the microcontroller resets and shows the RESF value indicating a watchdog timer reset.

3.1.5 Clock Monitor

To demonstrate the clock monitor function, run a program displaying a value on the LEDs—"CL," for example. The LEDs continue to display "CL" while the CPU runs on the main clock.

With the clock monitor set and the CPU running on the main clock, pull the clock jumper (JP4) on the V850ES-KJ1+ Micro-Board to disconnect the X1 clock from the on-board crystal oscillator.

When the clock monitor detects that the main clock is not oscillating, the microcontroller switches to the internal ring-oscillator clock. At this time, the LEDs display "CL" in blinking mode. The blinking is rather slow, since the processor is running at about 240 KHz.

3.2 Software Flow Charts

Reference Appendix A for the flow chart of the demonstration program.

3.3 Applilet's Reference Drivers

Reference Appendix B for screen captures and descriptions of the Applilet selections made for this demonstration.

3.4 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.

3.4.1 Resources

The demonstration uses the following resources:

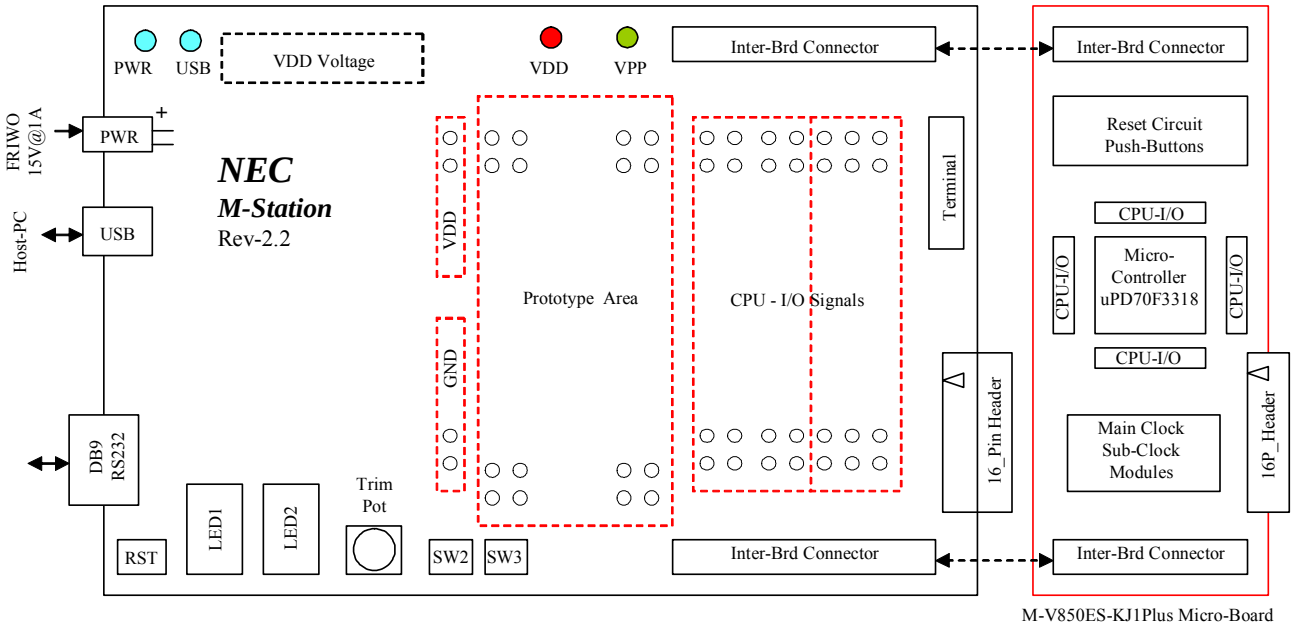
- M-V850ES-KJ1+ Micro-Board, with uPD70F3318 32-bit microcontroller mounted
- M-Station II Evaluation System, using M-Station II resources:
  - RST switch generating low-level reset
  - 7-segment LED displays LED1 and LED2
  - Pushbutton switches SW2 and SW3
  - Potentiometer for VDD-GND connection
  - DPDT switch for clock monitor demonstration

Add a 100Ω potentiometer between VDD_FLASH (JP1.1) and GND. With the tap of the potentiometer connected to VDD (JP1.2), this setup allow you to vary V_{DD}.

The standard M-Station-II potentiometer is connected between V_{DD} and GND, connected by default to uPD70F3318 analog input P70/ANI0.

For details on the hardware listed above, please refer to the appropriate User's Manual, available from NEC Electronics upon request.

Figure 9. M-Station II Evaluation System



3.4.2 Demonstration Program

With the hardware configured and the uPD70F3318 microcontroller programmed with the demonstration code or run with ID850QB-MST debug monitor program, the demonstration operates as follows.

3.4.2.1 External Reset Demonstration

To demonstrate an external reset, press the RST switch to ground the microcontroller's reset input. Upon receiving this input to external input pin RESET_B, the microcontroller resets and initializes each hardware unit. The reset source flag register (RESF) is set to 00h, if the reset source is RESET_B or a power-on clear reset (POCRES). The RESF is a special register that indicates the source of a reset signal.

To demonstrate external reset, set V_{DD} above Vpoc (2.6V) using the potentiometer on the V_{DD} pin. As V_{DD} rises above Vpoc, the processor comes out of the reset condition. Press SW2 or SW3 to read the RESF register. Then press the RST switch to reset the processor. The RESF display clears because the reset source is external reset. Pressing SW2 or SW3 prompts the program to read the RESF register and display its value, which is 00h.

These are the steps in the external reset demonstration:

- Apply VDD power above the Vpoc threshold and observe "00" on LEDs.
- Press either SW2 or SW3 and observe "=1" (selection of test).
- Press RST switch to assert reset. On release, observe "00" on LEDs.

3.4.2.2 Power-on Clear (POC) Demonstration

The V850ES-KJ1+ microcontroller has its power-on clear voltage (Vpoc) set at VDD=2.6 plus/minus 0.1V. The microcontroller's internal detector compares the VDD level with the preset Vpoc. When VDD falls below Vpoc, the microcontroller resets and the display goes blank. When you increase VDD above Vpoc by turning the potentiometer, the microcontroller comes out of reset. The RESF is now set to 00h, because the source of the reset is power-on clear. Press SW2 or SW3 to read and observe 00h on the LEDs.

These are the steps in the power-on clear demonstration:

- Apply VDD below the Vpoc threshold and observe that the LEDs are blank.
- Increase VDD above the Vpoc threshold and observe that the LEDs show "00".
- Press SW2 or SW3 and observe "=1" on the LEDs.
- Reduce VDD below the Vpoc threshold and observe that the LEDs are again blank (reset asserted).
- Increase VDD above the Vpoc threshold and observe that the LEDs again show "00".

3.4.2.3 Low-Voltage Detect (LVI) Demonstration

After you press SW2 or SW3 and see "=1" displayed, pressing SW2 changes the display to "=2", "=3", through "=7" and back to "=1". In this way, SW2 selects one of seven tests. Press SW3 to execute a test. When you press SW3, the left LED digit shows the count-up number, and the right digit is blank. Tests 1 through 4 are LVI demonstration tests, as listed below.

Table 2. LVI Demonstration Tests

Test	LED	Test Function	Function Operation
1	"1 "	voltage_reset(0)	Set LVI to RESET on VDD < 4.3V
2	"2 "	voltage_reset(3)	Set LVI to RESET on VDD < 3.7V
3	"3 "	voltage_Interrupt(0)	Set LVI to Interrupt on VDD < 4.3V
4	"4 "	voltage_interrupt(3)	Set LVI to Interrupt on VDD < 3.7V

To demonstrate an LVI reset, follow these steps:

- Apply VDD power above 3.93V.
- Using SW2, select test "=1" or "=2".
- Press SW3 and observe mode "1 " or "2 "
- Reduce VDD below the set voltage.
- Observe that the LEDs blank at the appropriate voltage.
- Increase VDD above the set voltage.
- Observe that the LEDs display "Lr" indicating that LVI caused the reset.

If you try to start up with the voltage still below the required operating voltage, the LEDs display the center bar.

To demonstrate an LVI interrupt, follow these steps:

- Apply VDD power above 3.93V.
- Use SW2 to select test "=3" or "=4".
- Press SW3 and observe mode "3 " or "4 ".
- Reduce VDD below set voltage.
- Observe LED display of "LI" at appropriate voltage.
- Press SW2 to return to menu of tests.

3.4.2.4 Watchdog Timer Demonstration

On initialization, the watchdog timer is set so that the overflow time is about 68 milliseconds. Timer TM0 is initialized to interrupt every 9.6 milliseconds, and the interrupt service routine for Timer TM0 clears the watchdog timer counter.

Table 3. Watchdog Timer Tests

Test	LED	Test Function	Function Operation
5	"5 "	watchdog_timer_reset()	Watchdog timer reset
6	"6 "	watchdog_timer_interrupt()	Watchdog non-maskable interrupt

To run the watchdog timer demonstration, follow these steps:

- Use SW2 to select test "=5".
- Press SW3.
- Observe "51" displayed by the LEDs with the right digit blinking rapidly. At this point, the Timer TM0 interrupt INTTM0011 is occurring once every 9.6 milliseconds and clearing the watchdog timer.
- Press SW3 repeatedly.
- Observe "52", "53", etc., on the LEDs with the right digit blinking more slowly.

Note: Repeat this same sequence for test "=6", and observe "62", "63", etc.

For every press of SW3, the demonstration program adds another 9.6 milliseconds to the Timer TM0 interval. To generate the blinking, the program increments a variable in the interrupt service routine for INTTM001. As the interval increases, the INTTM001 interrupt happens less frequently, so the blinking rate slows.

The table below shows the number of presses on SW3, the Timer TM0 interval set, and the LED display. The timer and watchdog have not been precisely calibrated.

Table 4. Watchdog Timer Test Intervals

SW3 Presses	Timer H1 Interval	LED Display
1	9.6 msec	"51", 1 is blinking rapidly
2	19.2 msec	"52", 2 is blinking more slowly
3	28.8 msec	"53", 3 is blinking more slowly
4	38.4 msec	"54", 4 is blinking more slowly
5	48.0 msec	"55", 5 is blinking more slowly
6	76.8 msec	"d1"; Watchdog Timer has caused Reset

When the interval for the Timer H1 interrupt increases past 52 milliseconds, the watchdog timer is not cleared in time, and the counter overflows, causing a watchdog timer reset. The program starts again and displays the value "d1" for RESF, which indicates that the watchdog timer reset flag (WTDRF) is set and shows that the watchdog timer caused the reset.

At this point, the program has been reinitialized, so the Timer TM0 interval is again at 9.6 milliseconds. The watchdog timer is being cleared before overflow.

3.4.2.5 Clock Monitor Demonstration

To demonstrate the clock monitor function, a double-pole/double-throw (DPDT) switch is wired in place of the jumpers for the crystal at JP4 (pin 1, 2, 3, 4), as shown in Figure 10.

To run the clock monitor demonstration, follow these steps:

- Place the DPDT switch to its NC (normally closed) position.
- Use SW2 to select test "=7".
- Press SW3.
- The LEDs display the count-up number when the CPU is running with the main clock.
- Toggle the DPDT switch to its NO (normally open) position.
- The LEDs display "CL" when the CPU is running with the internal ring oscillator.

Table 5. Clock Monitor Test

Test	LED	Test Function	Function Operation
7	"7 "	clock_monitor()	Clock Monitor reset

3.5 Hardware Block Diagram

Figure 10. Hardware Block Diagram

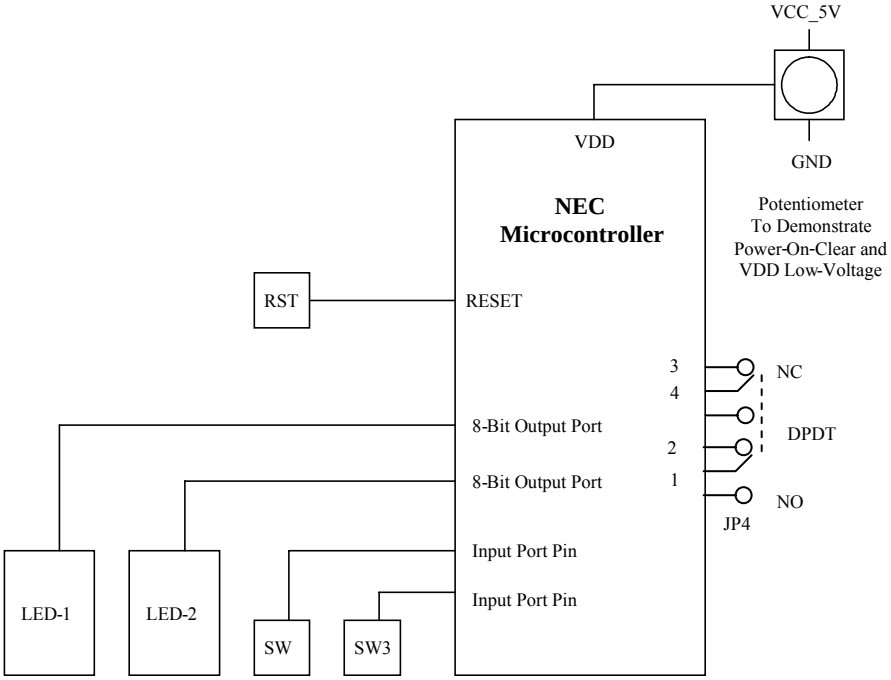
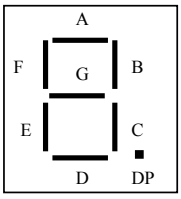


Figure 11. LED Connections



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

3.6 Software Modules

The software order is set up by the Applilet, which builds a make file when you generate the base code. The make file has the name of your project and the suffix ".mak". In the make file, the start function is given by the command line "STARTUP= crtE.o". This module is linked to load at address zero, which is where the reset vector is located. In crtE.s in section "RESET" is an instruction to jump to __start. This is where the program begins when the processor starts running.

In crtE, when __start is called, the routine must set up various registers that control the program operation. The routine initializes registers to be a text pointer, global variable pointer, and stack pointer. The routine

clears the sbss, bss and IRAM areas of memory. Variable __argc is loaded into r6, and pointer __argv is loaded into r7. Then the routine calls main(), the normal start of a C program. Although registers r6 and r7 have been loaded, it is unusual for an embedded program to take input parameters.

When main() is entered, it immediately calls SystemInit() to set up the predefined variables for the C program and initialize all of the peripheral subsystems. On return, main() starts the watch timer and Interval Timer 1, and displays the current test number. The routine then enters an infinite loop, checking for operator input from push buttons SW2 and SW3.

The watch timer blinks the left LED's decimal point, changing state every 50 interrupts. The interval timer T01 blinks the right LED's decimal point, changing state on every interrupt.

When you press SW2, the routine increments the test number. This incrementing continues until you reach the maximum number of tests, and then the test number restarts at 1.

When you press SW3, the function corresponding to the current test number (set by SW2) is called.

The flowcharts in Appendix A show the operation of each test function.

Reference Appendix C for the complete software listing.

4. Appendix A — Software Flowcharts

4.1 Main()

Figure 12. Safety_1.1.1 Main()

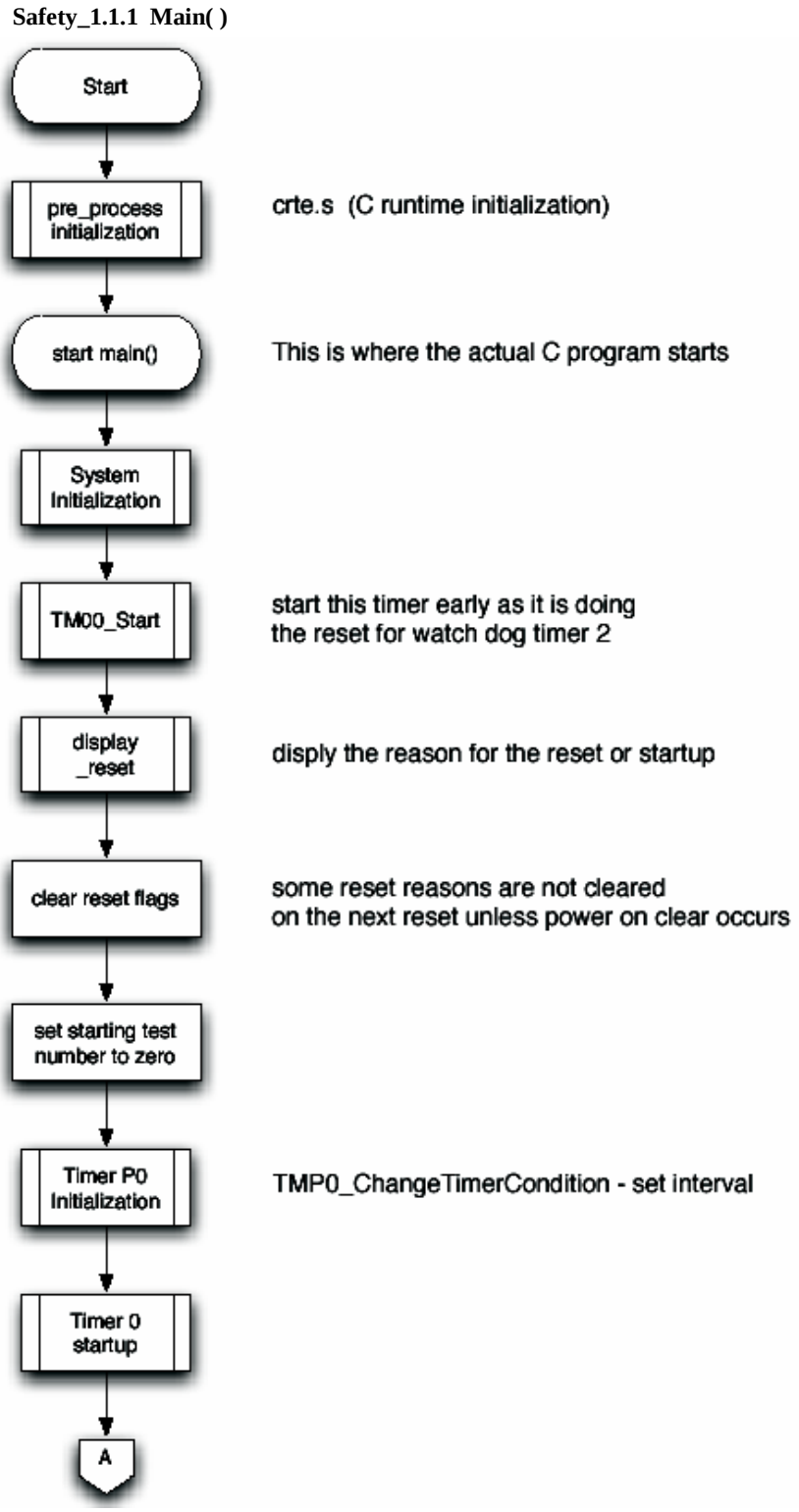


Figure 13. Safety_1.1.2 Main

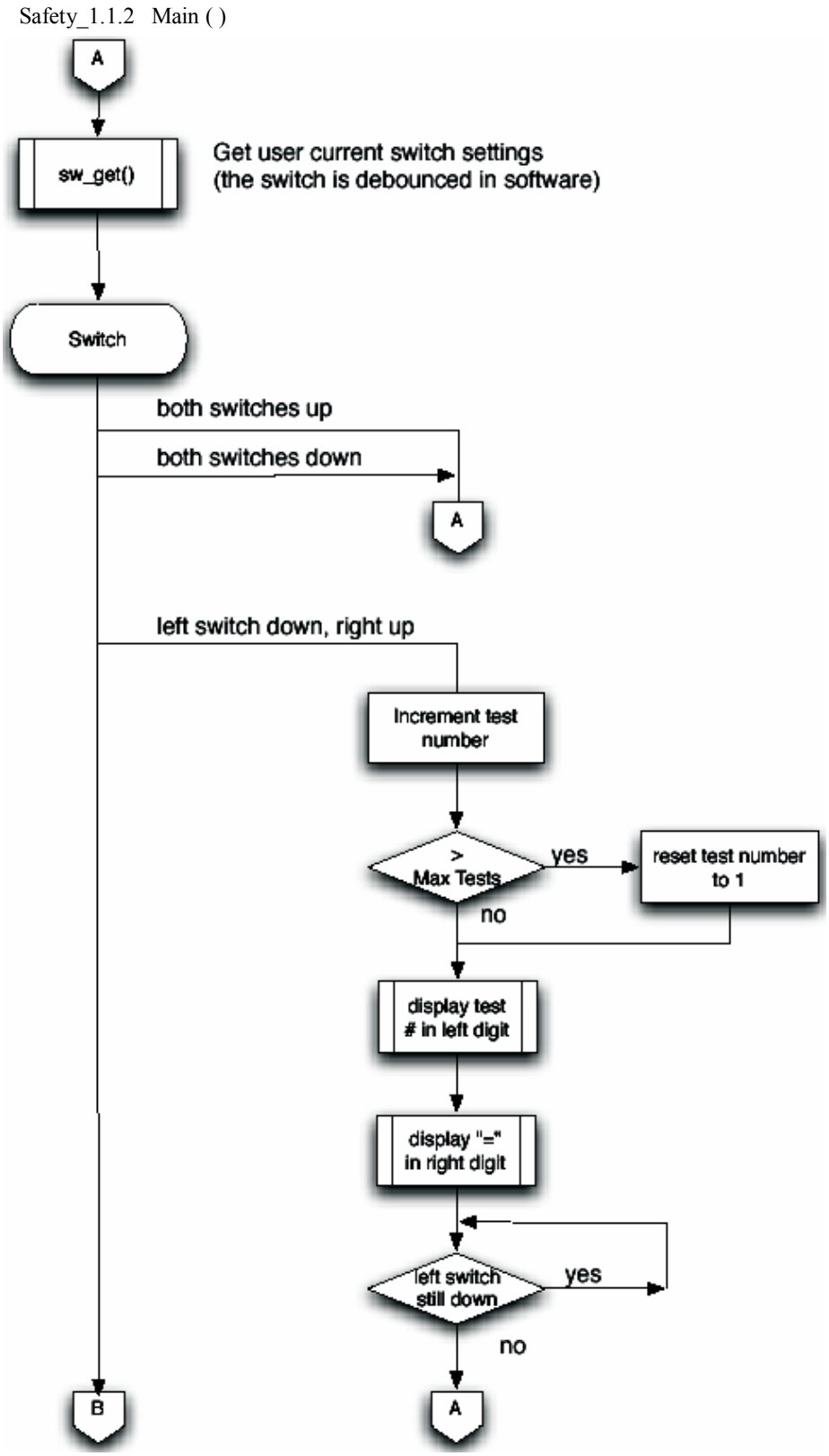
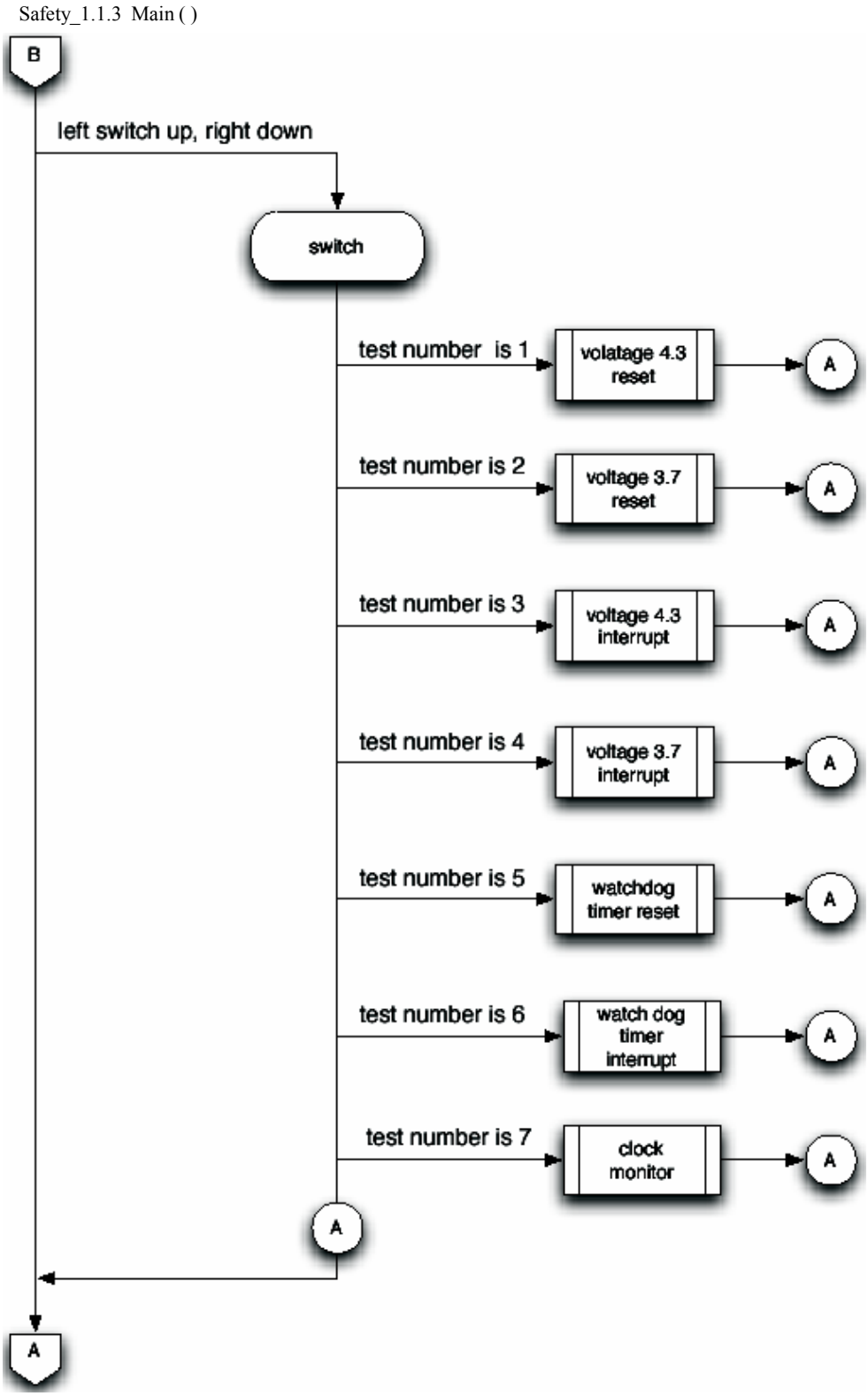
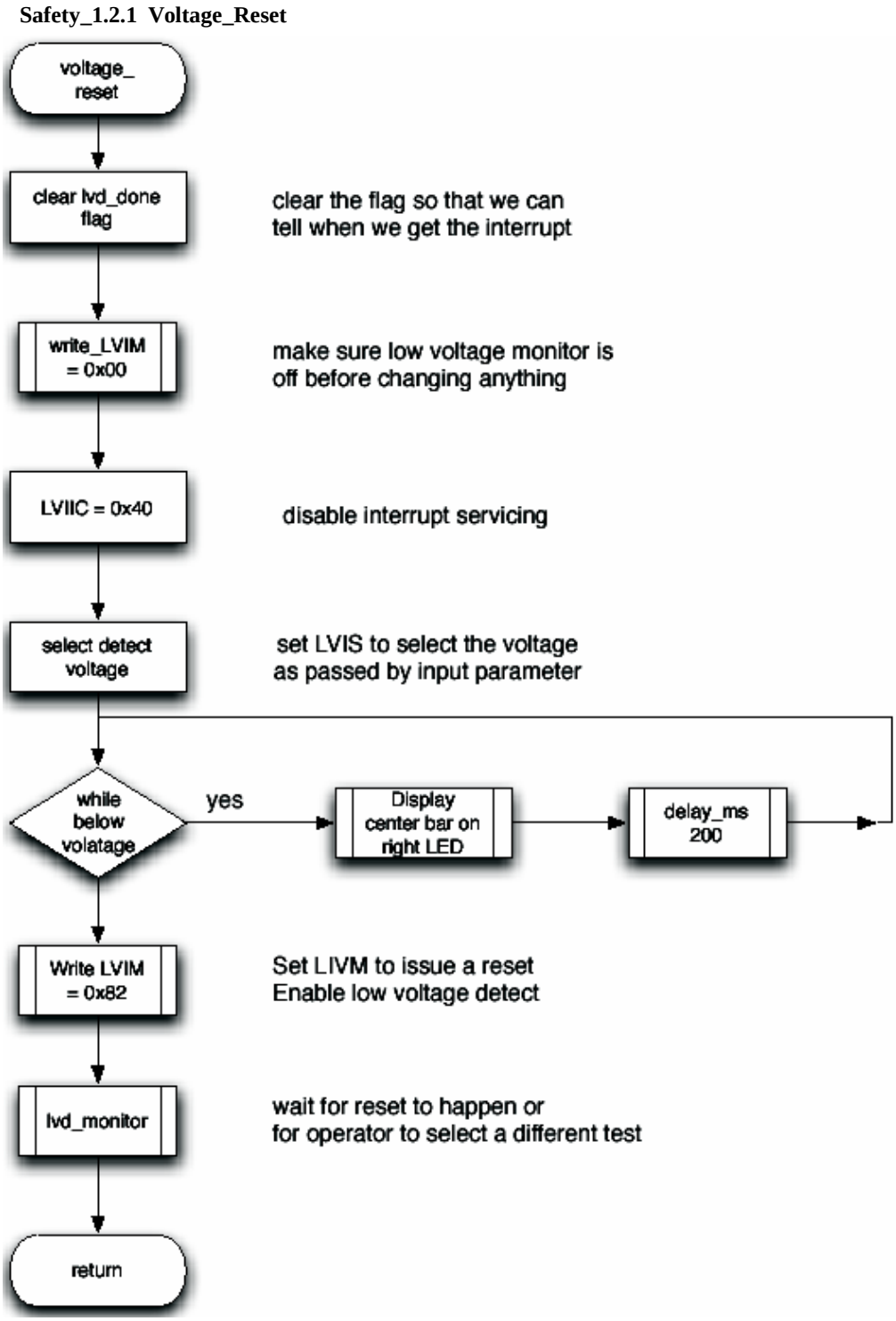


Figure 14. Safety_1.1.3 Main()



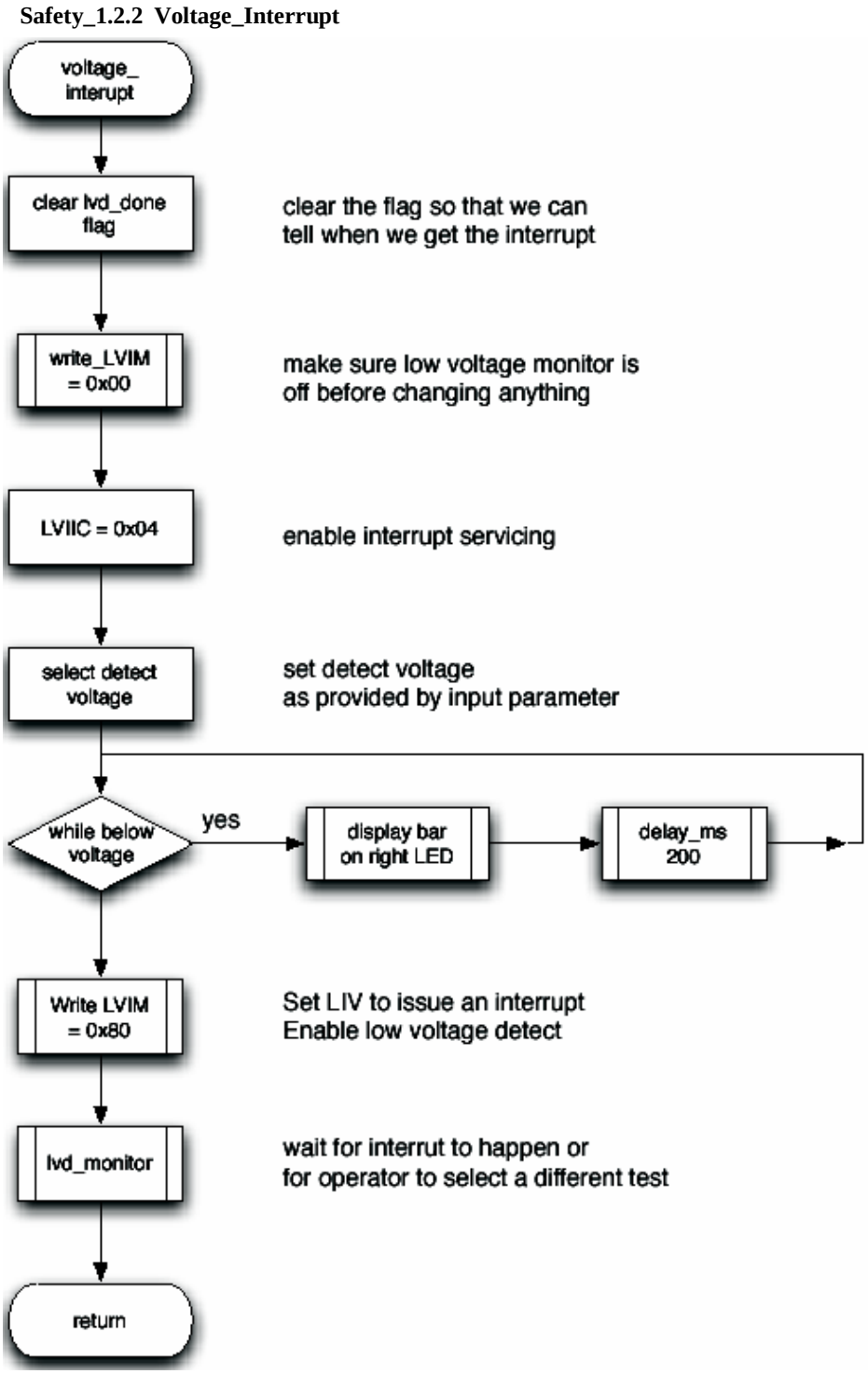
4.2 Voltage Reset

Figure 15. Safety_1.2.1 Voltage_Reset



4.3 Voltage Interrupt

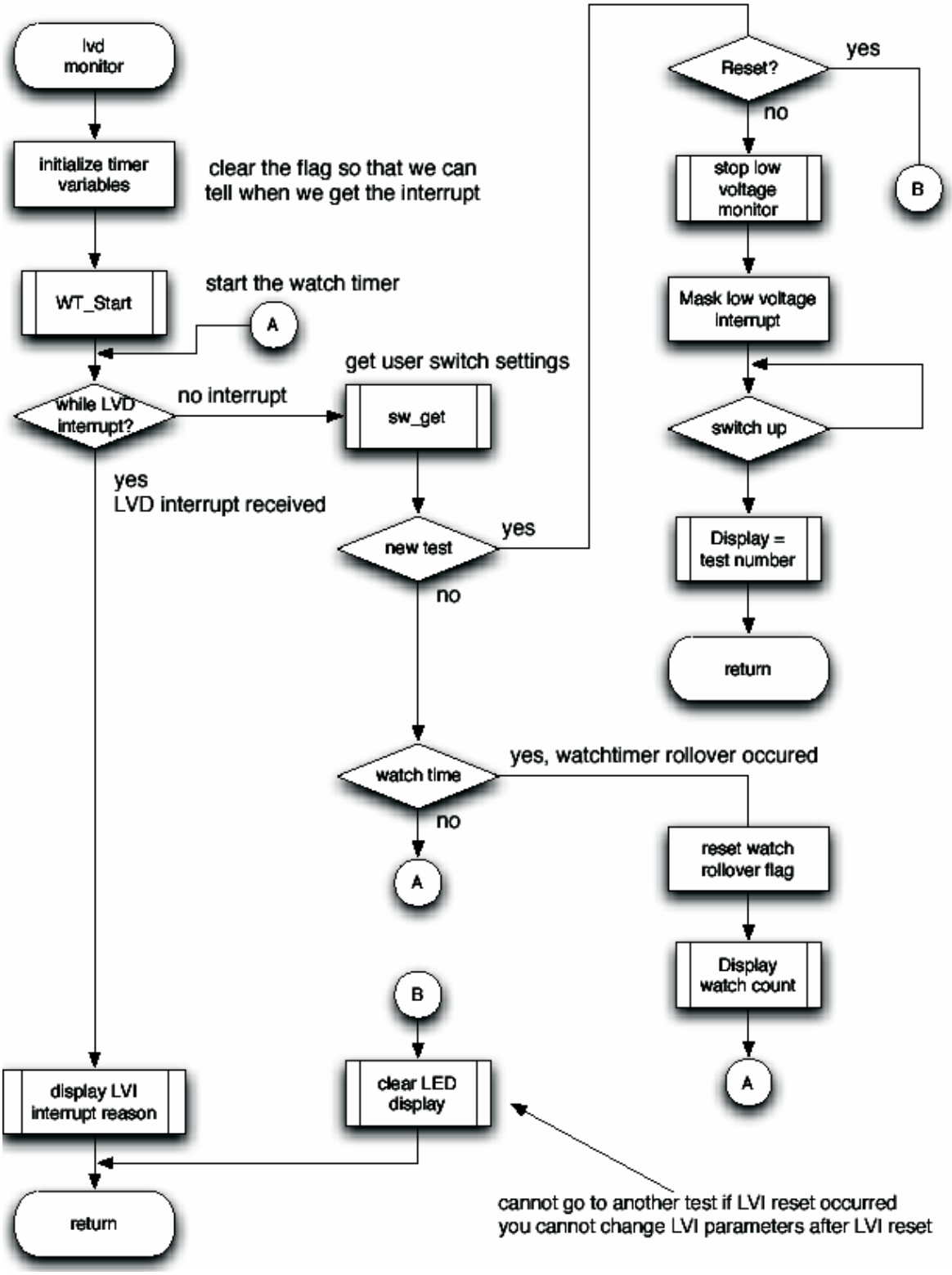
Figure 16. Safety_1.2.2 Voltage_Interrupt



4.4 Low Voltage Detect Monitor

Figure 17. Safety_1.2.3 lvd_mon

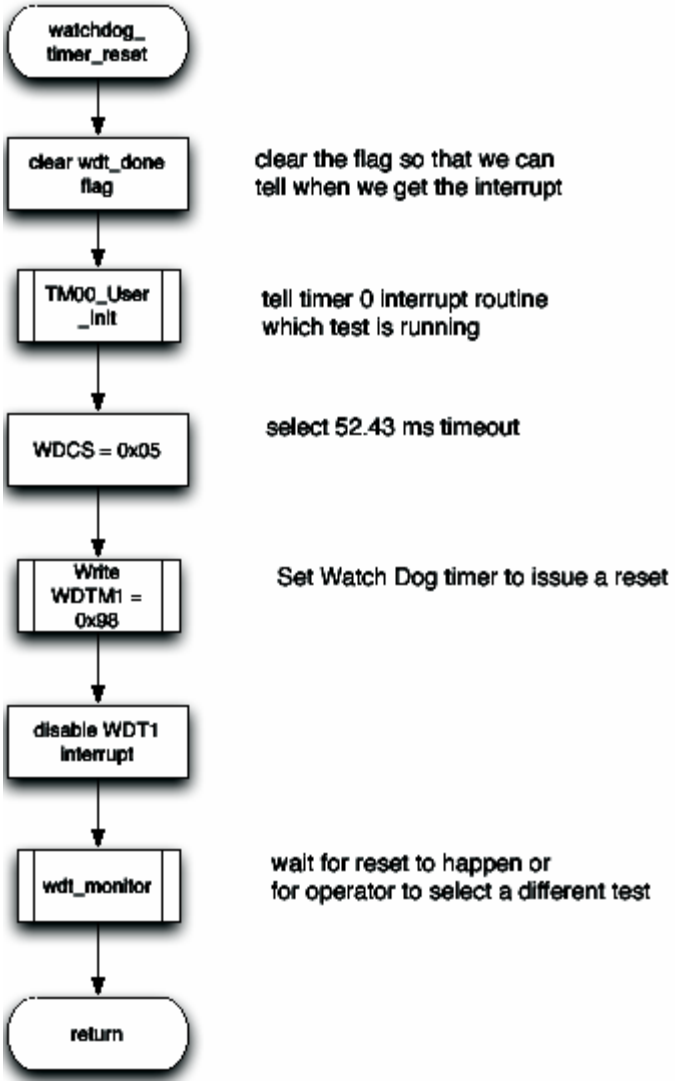
Safety_1.2.3_lvd_mon



4.5 Watchdog Timer Reset

Figure 18. Safety_1.3.1 Watchdog_Timer-Reset

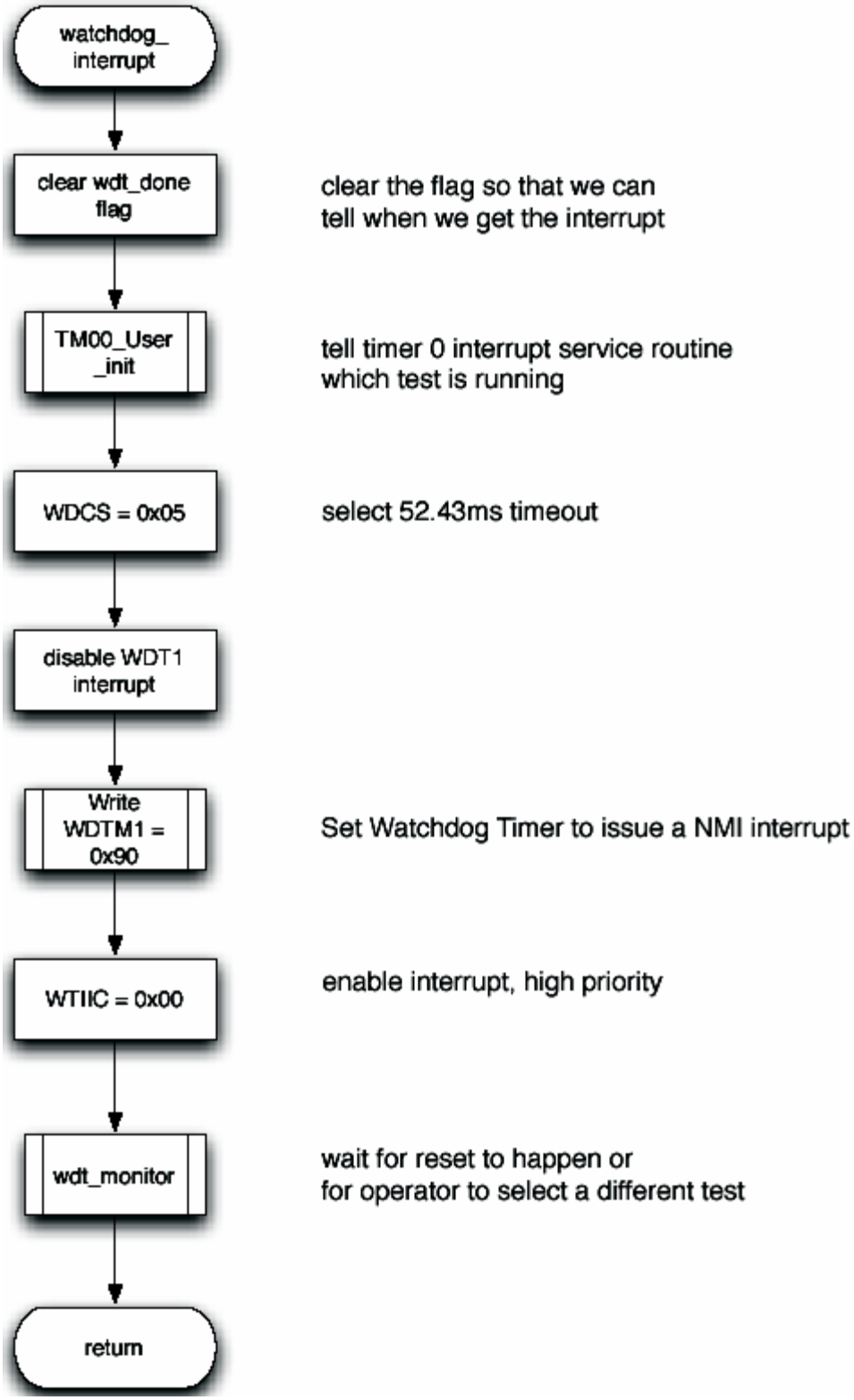
Safety_1.3.1 Watchdog_Timer-Reset



4.6 Watchdog Timer Interrupt

Figure 19. Safety_1.3.2 Watchdog_Interrupt

Safety_1.3.2 Watchdog_Interrupt



4.7 Watchdog Timer Monitor

Figure 20. Safety_1.3.3.1 Watchdog_Timer_Monitor

Safety_1.3.3.1 Watchdog_Timer_Monitor

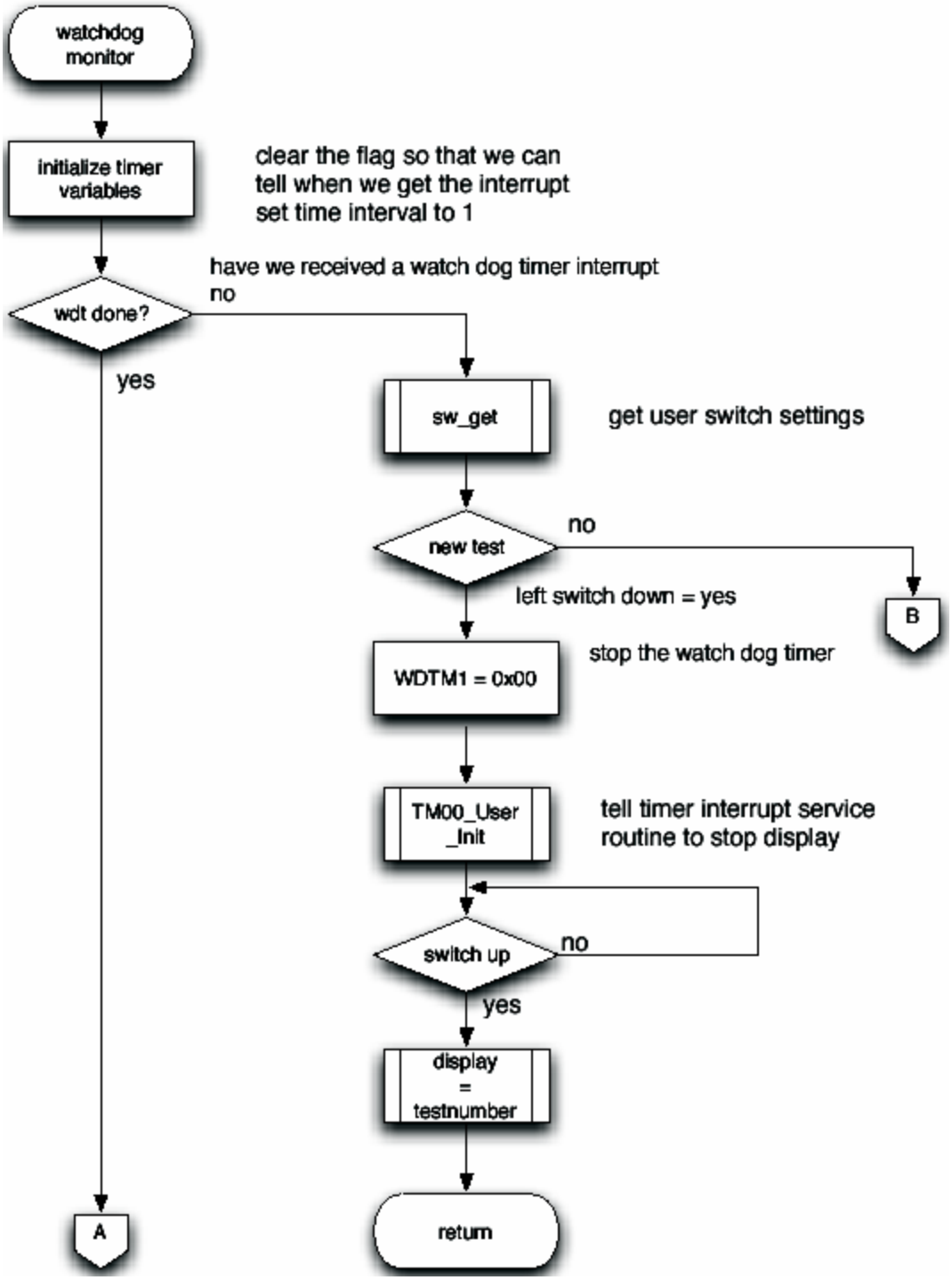
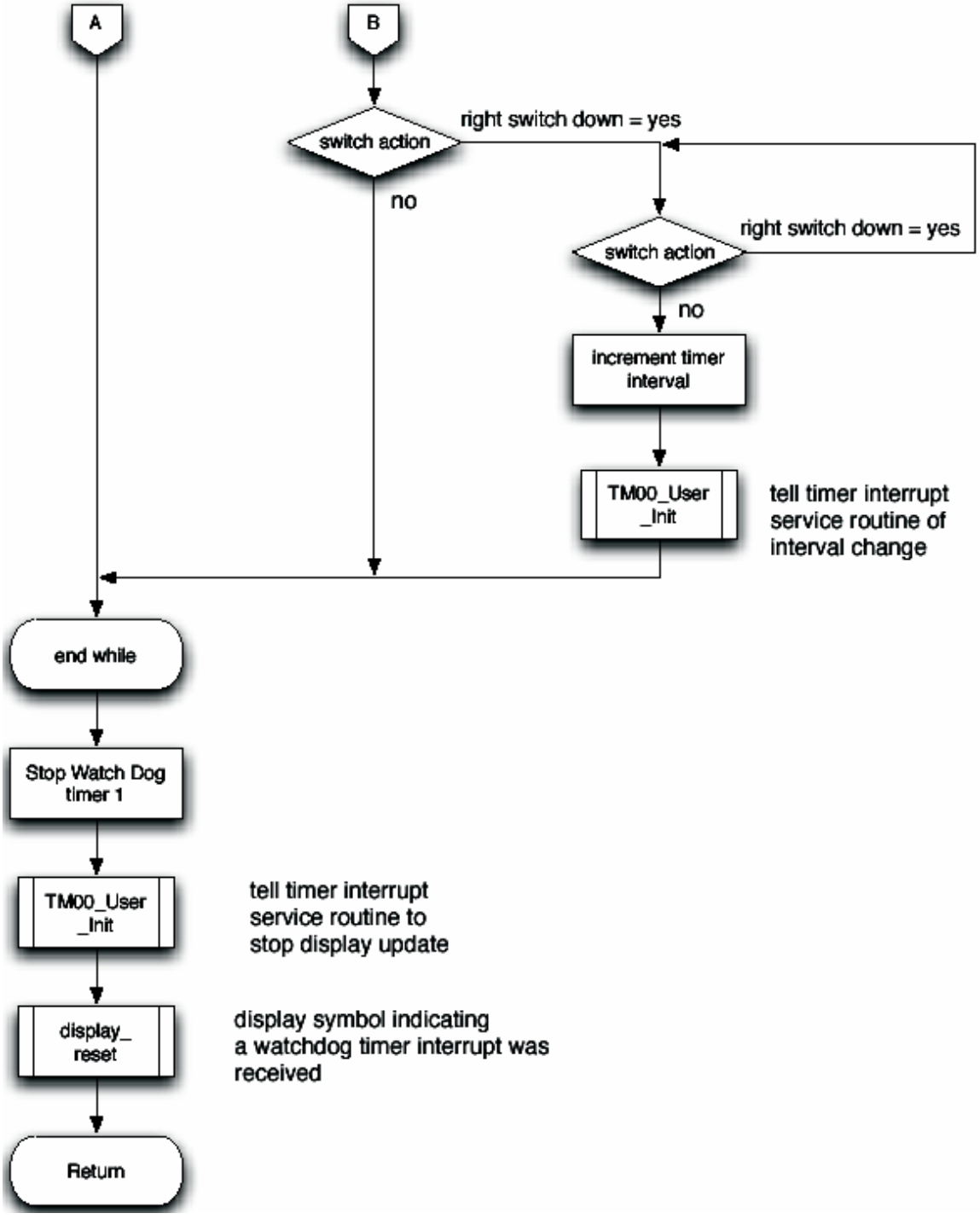


Figure 21. Safety_1.3.3.2 Watchdog_Timer_Monitor

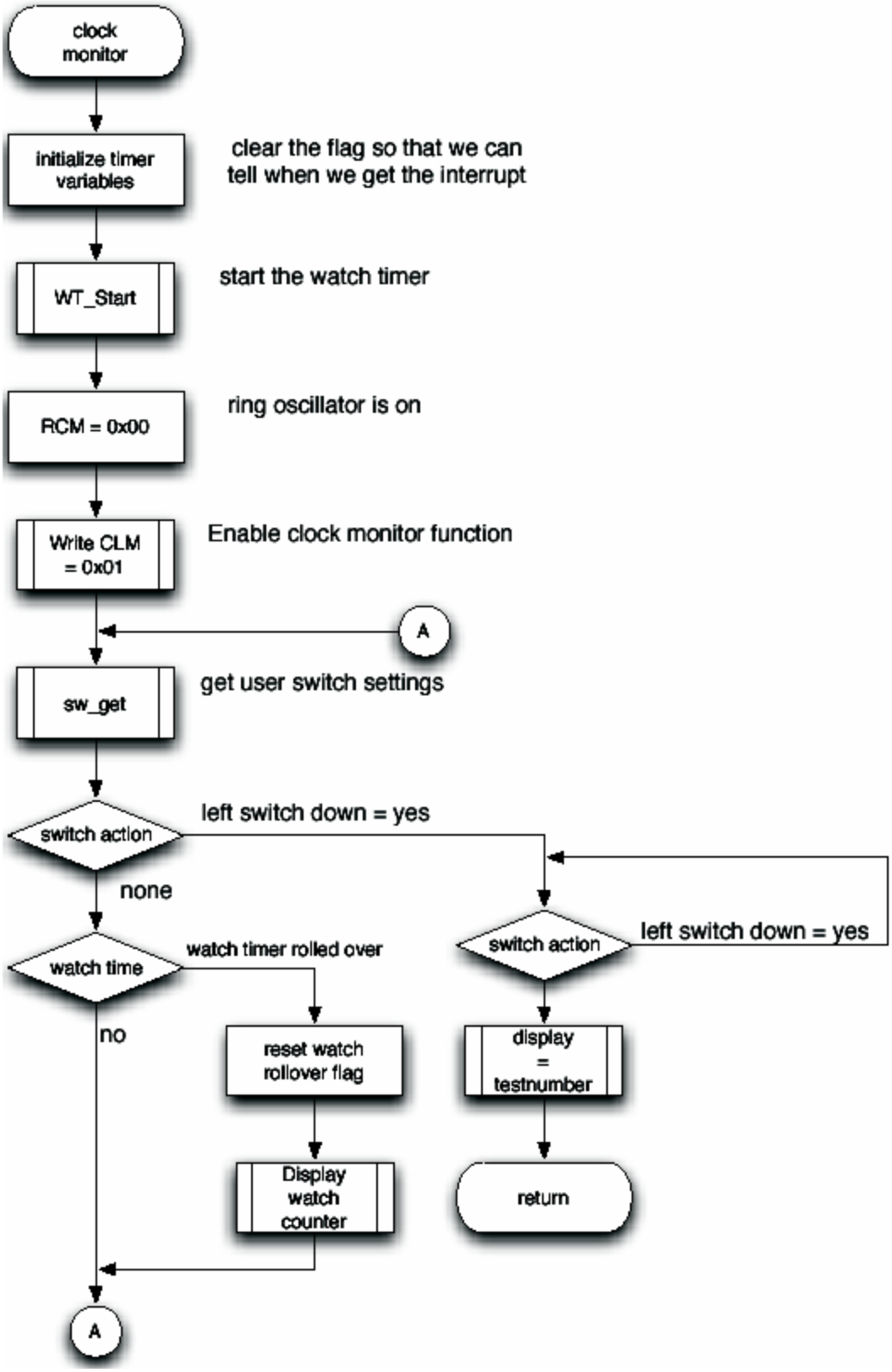
Safety_1.3.3.2 Watchdog_Timer_Monitor



4.8 Clock Monitor

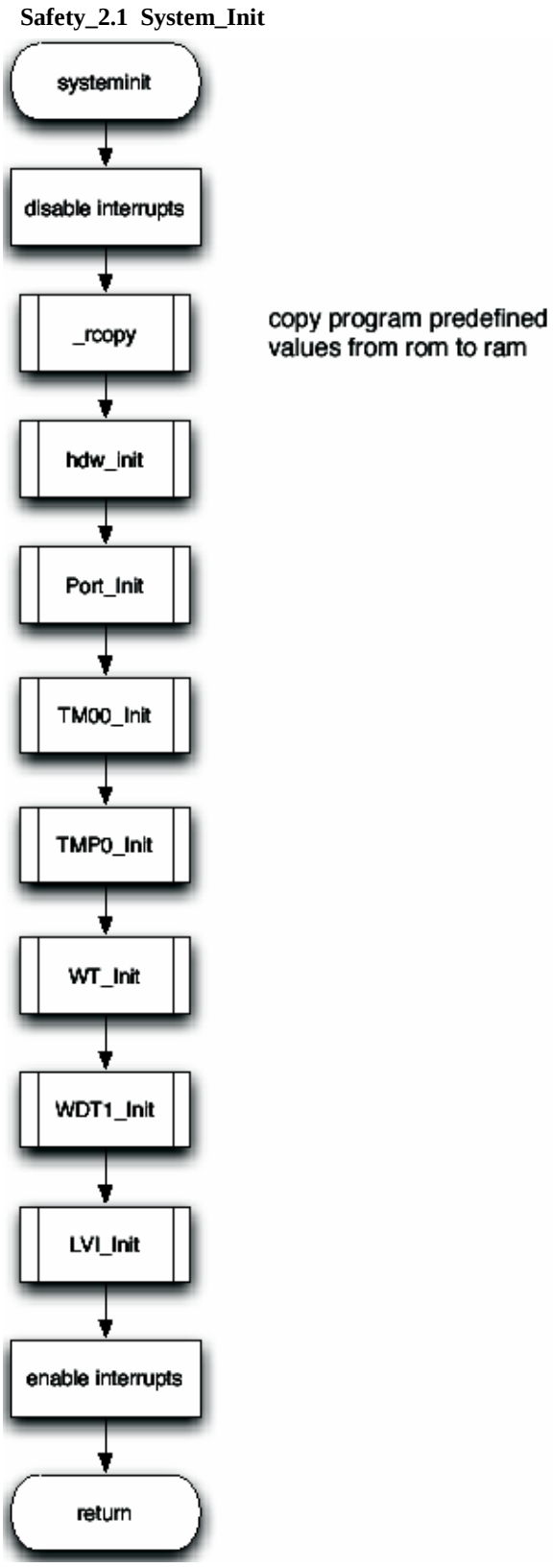
Figure 22. Safety_1.4.1 Clock_Monitor

Safety_1.4.1 Clock_Monitor



4.9 System Initialize

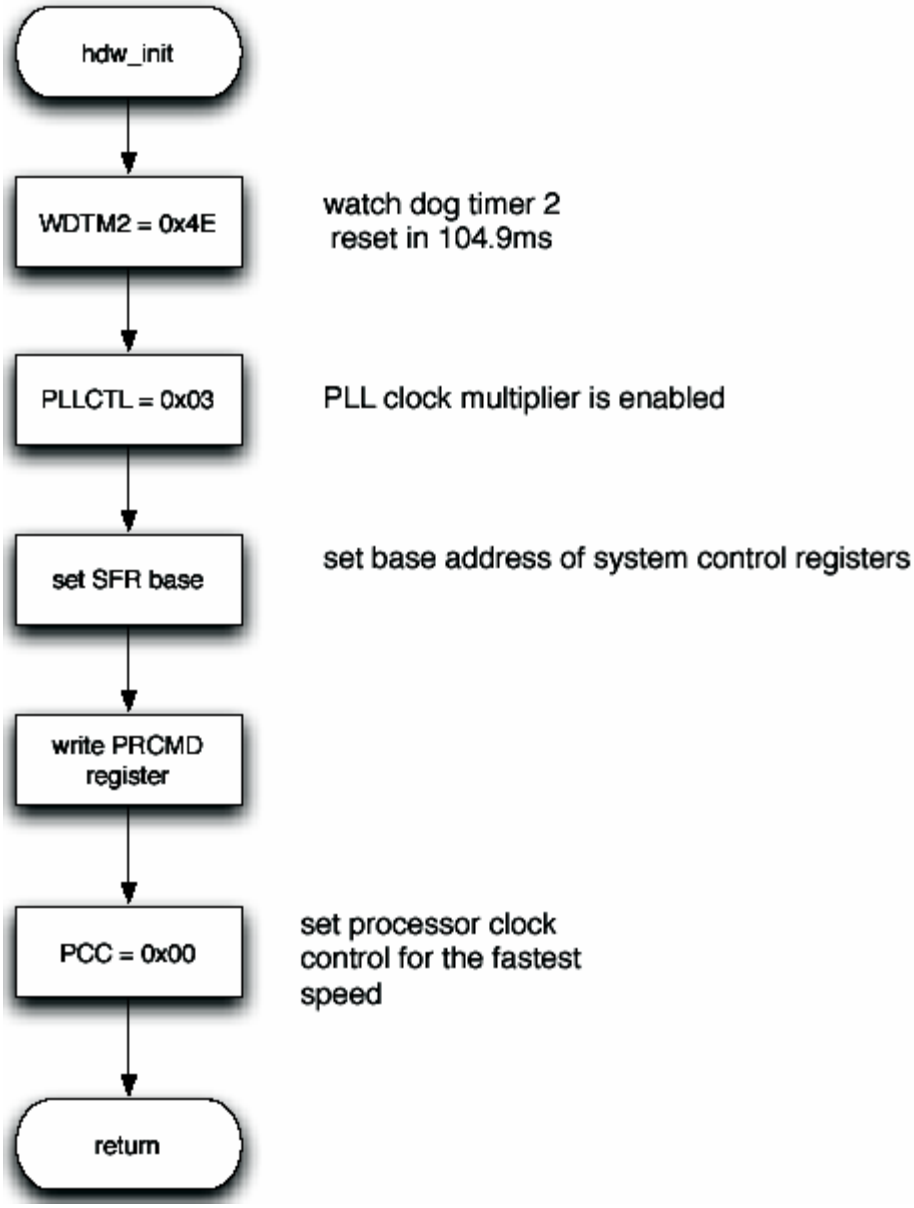
Figure 23. Safety_2.1 System_Init



4.10 System Initialize Hardware

Figure 24. Safety_2.2.1 Systeminit_hdw_init

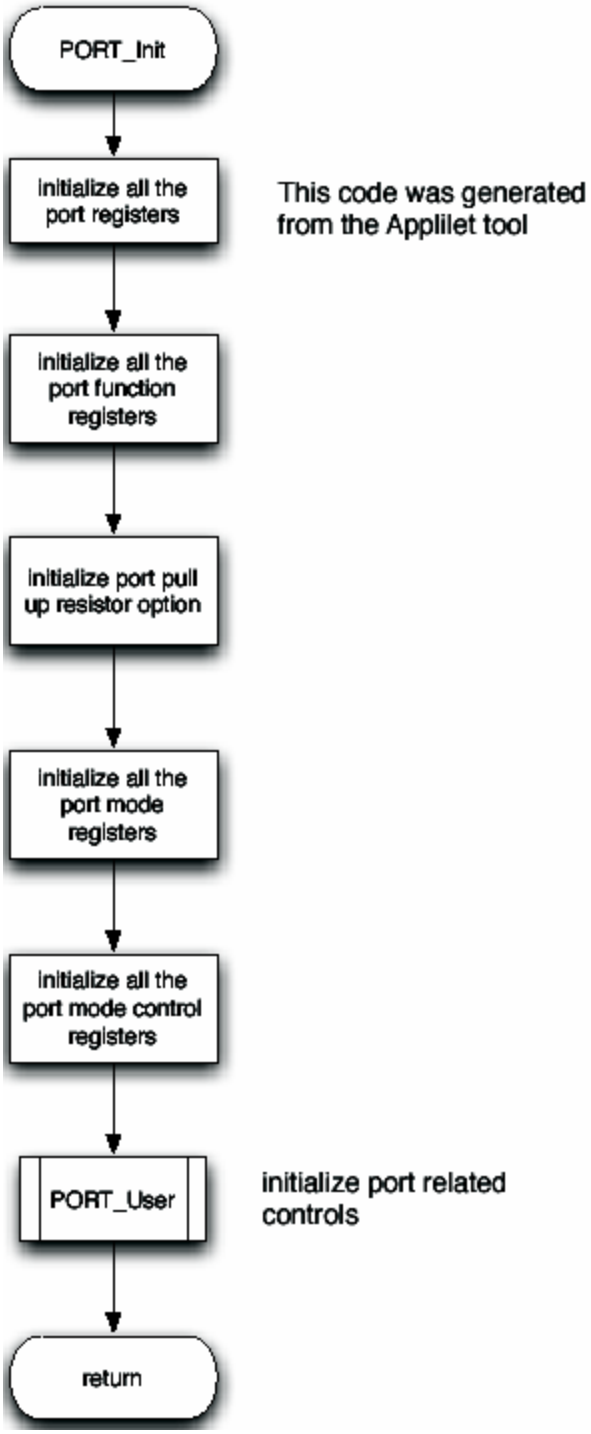
Safety_2.2.1 Systeminit_hdw_init



4.11 Port Initialize

Figure 25. Safety_2.3.1 Port_Init

Safety_2.3.1 Port_Init



4.12 Port User

Figure 26. Safety_2.3.2.1 Port_User_sw

Safety_2.3.2.1 Port_User_sw

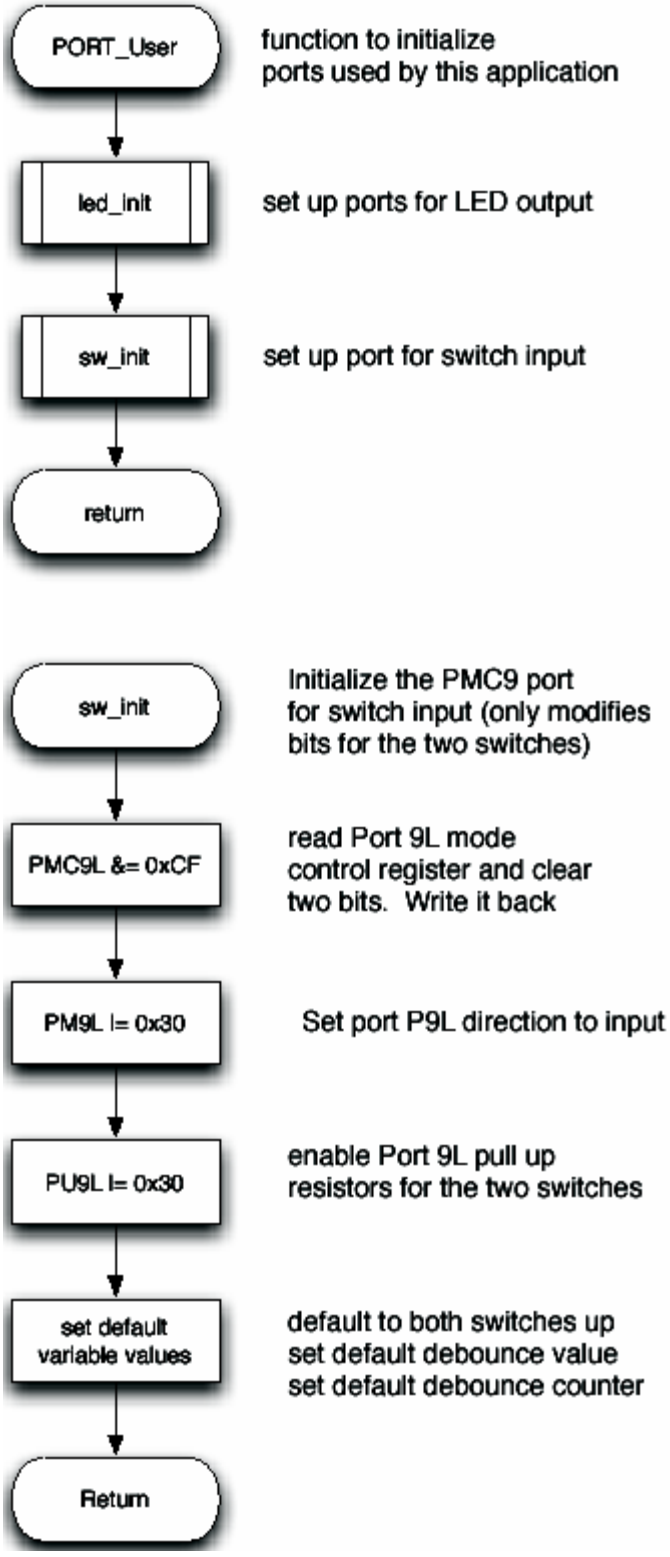


Figure 27. Safety_2.3.2.2 Port_User_sw

Safety_2.3.2.2 Port_User_sw

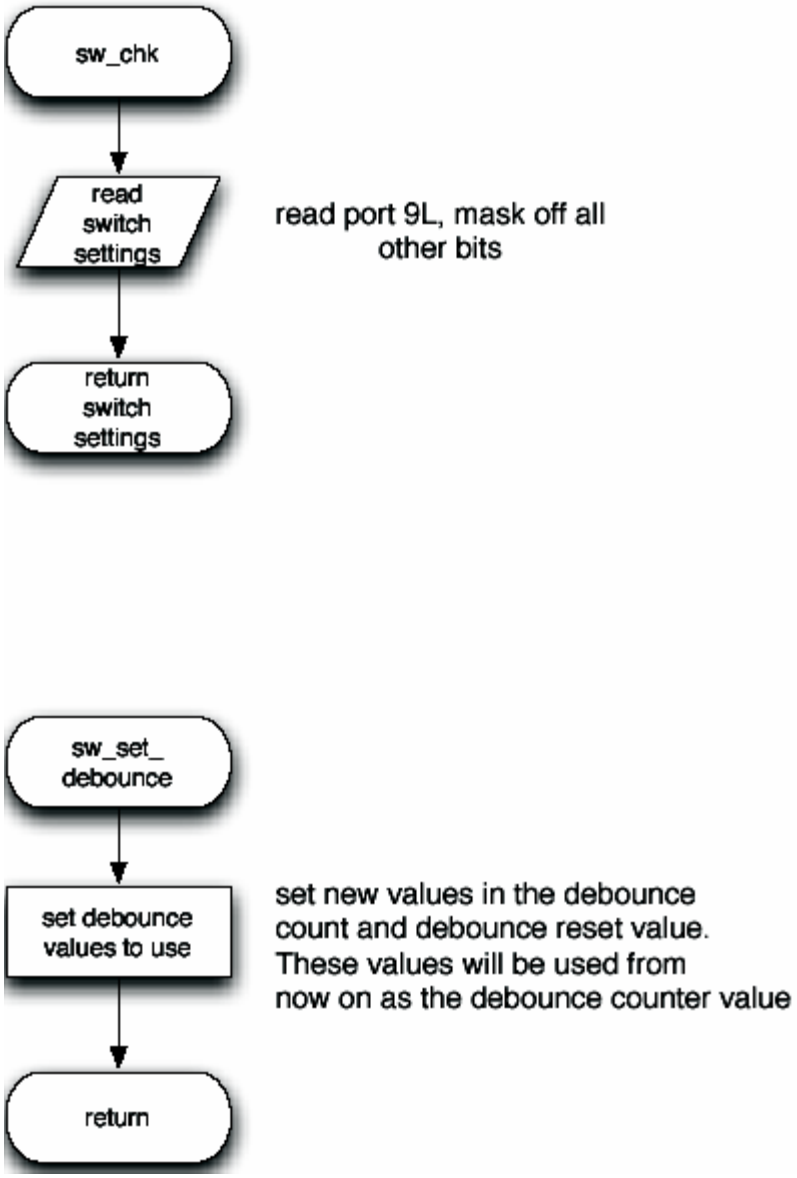
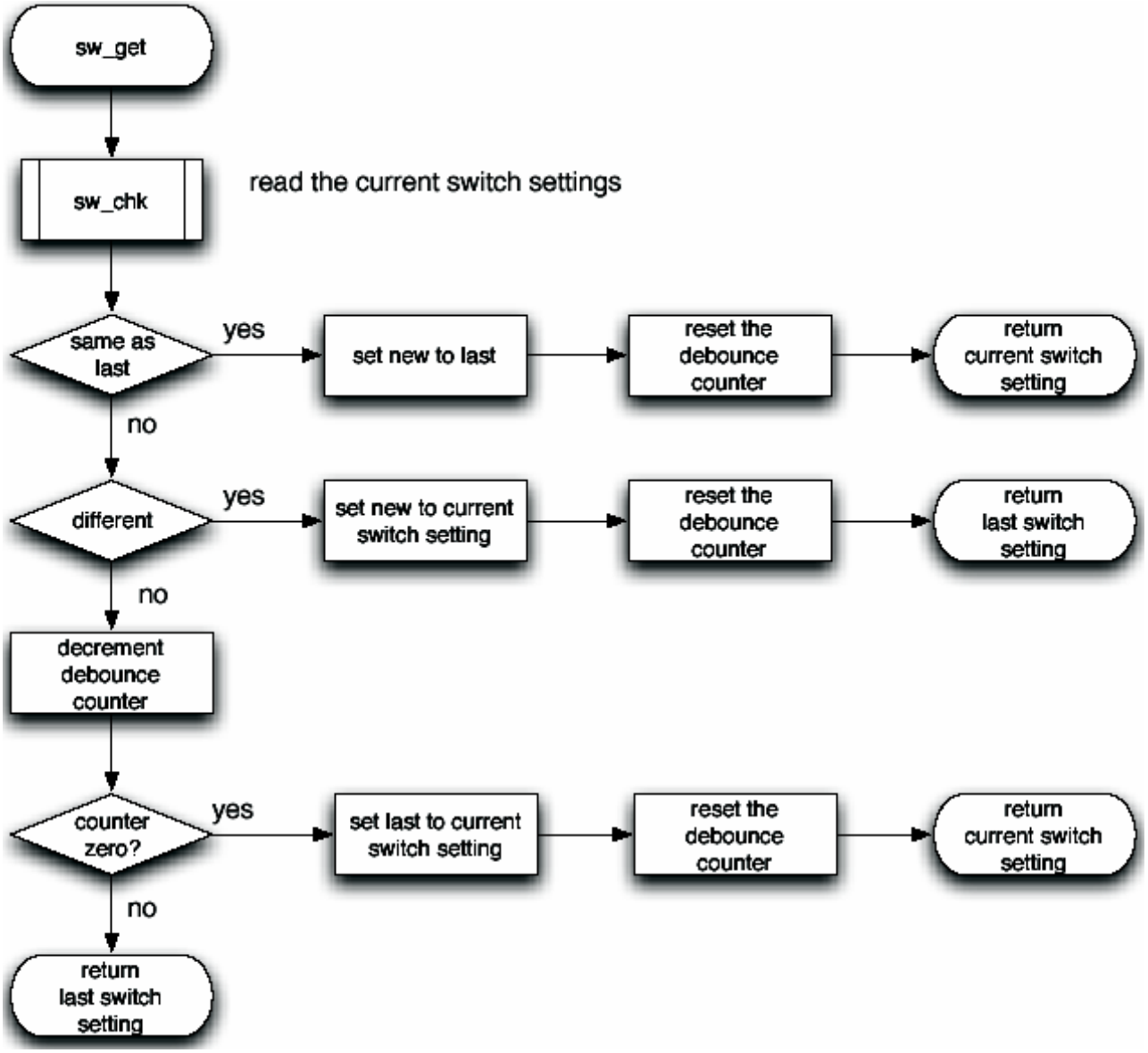


Figure 28. Safety_2.3.2.3 Port_User_sw

Safety_2.3.2.3 Port_User_sw



4.13 Port User LED

Figure 29. Safety_2.3.3.1 Port_User_LED

Safety_2.3.3.1 Port_User_led

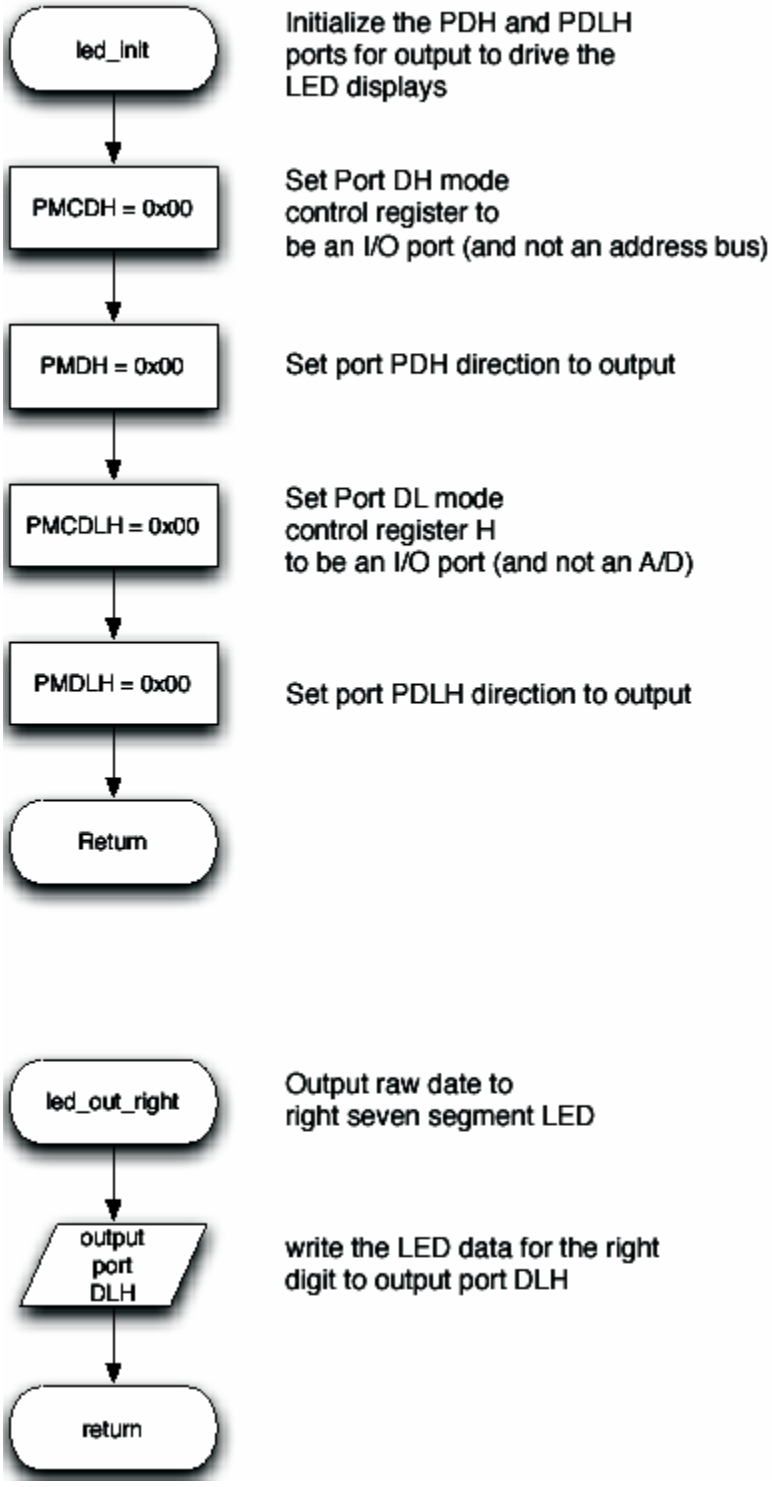


Figure 30. Safety_2.3.3.2 Port_User_LED

Safety_2.3.3.2 Port_User_led

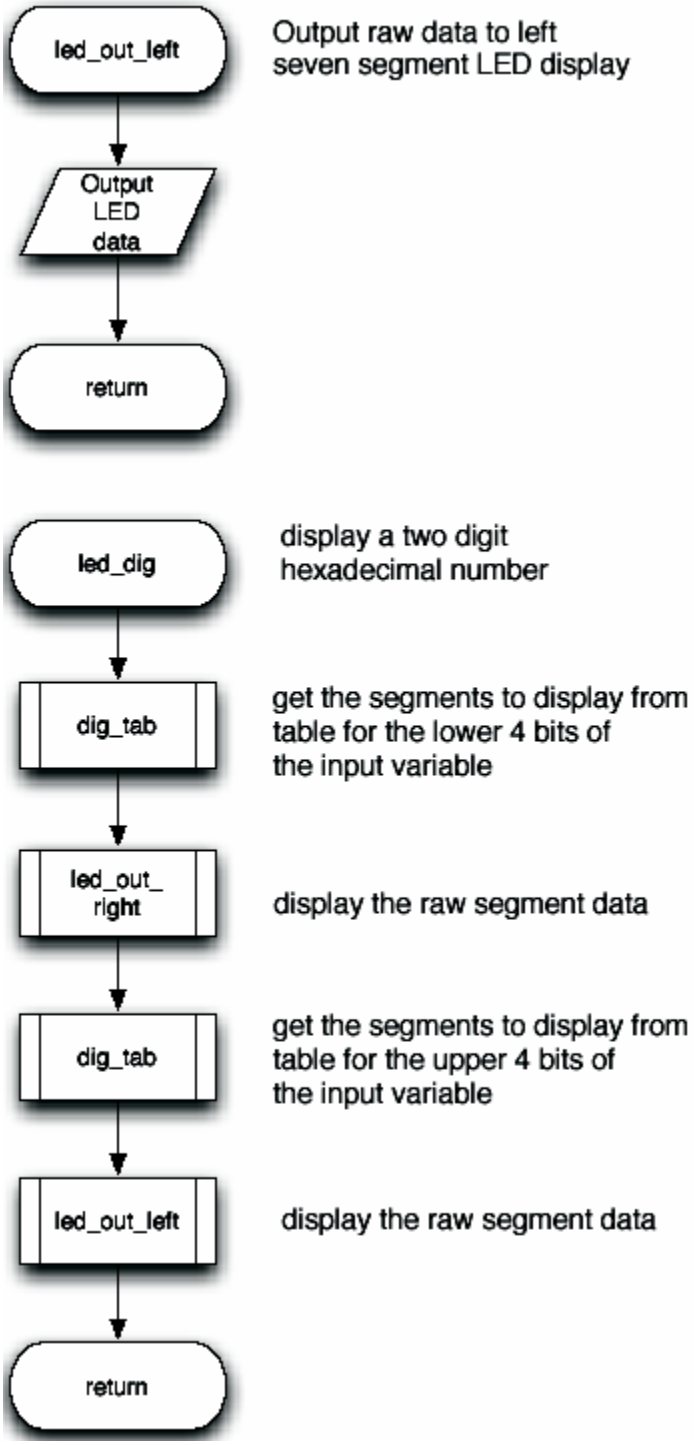


Figure 31. Safety_2.3.3.3 Port_User_LED

Safety_2.3.3.3 Port_User_led

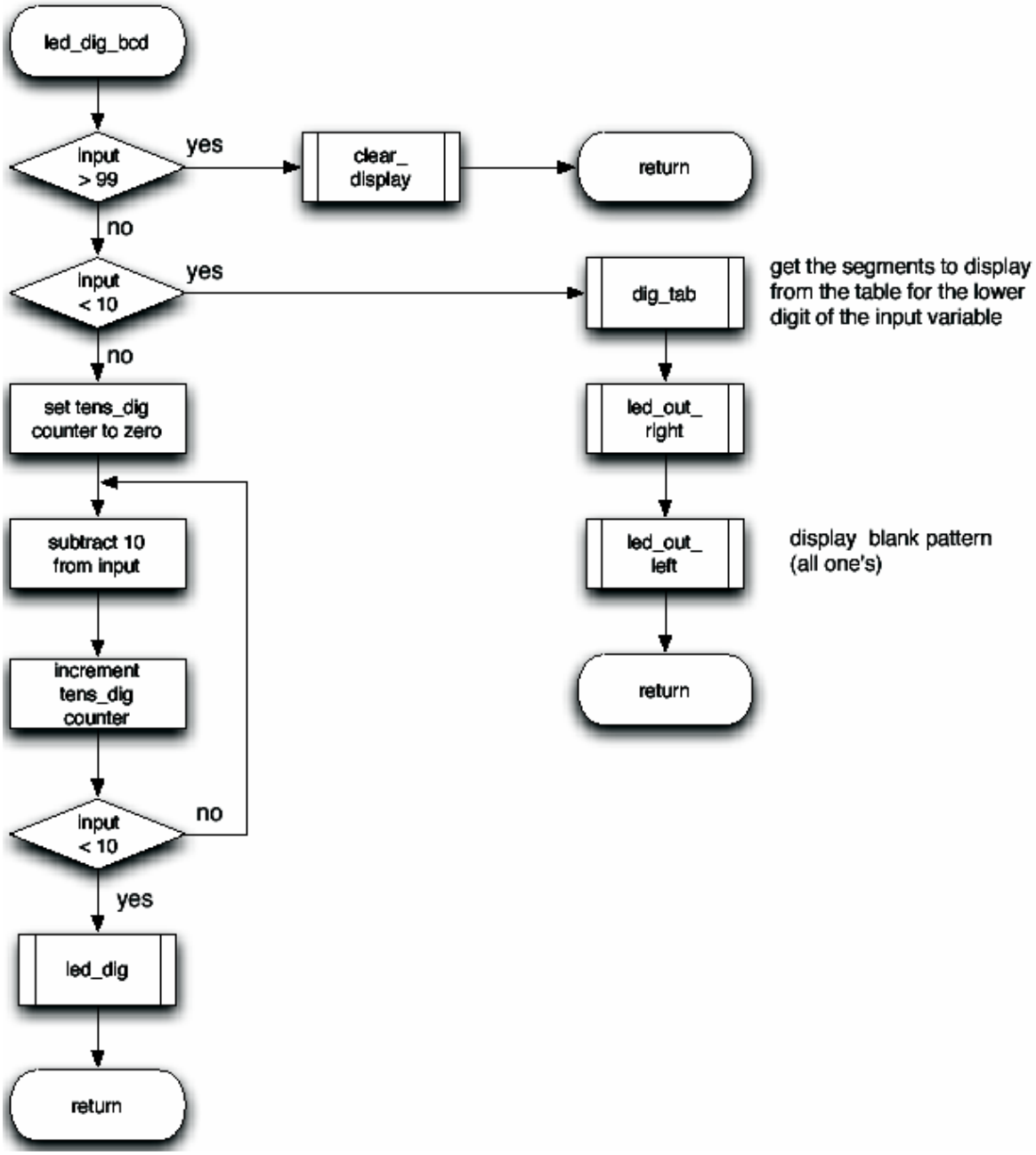


Figure 32. Safety_2.3.3.4 Port_User_LED

Safety_2.3.3.4 Port_User_led

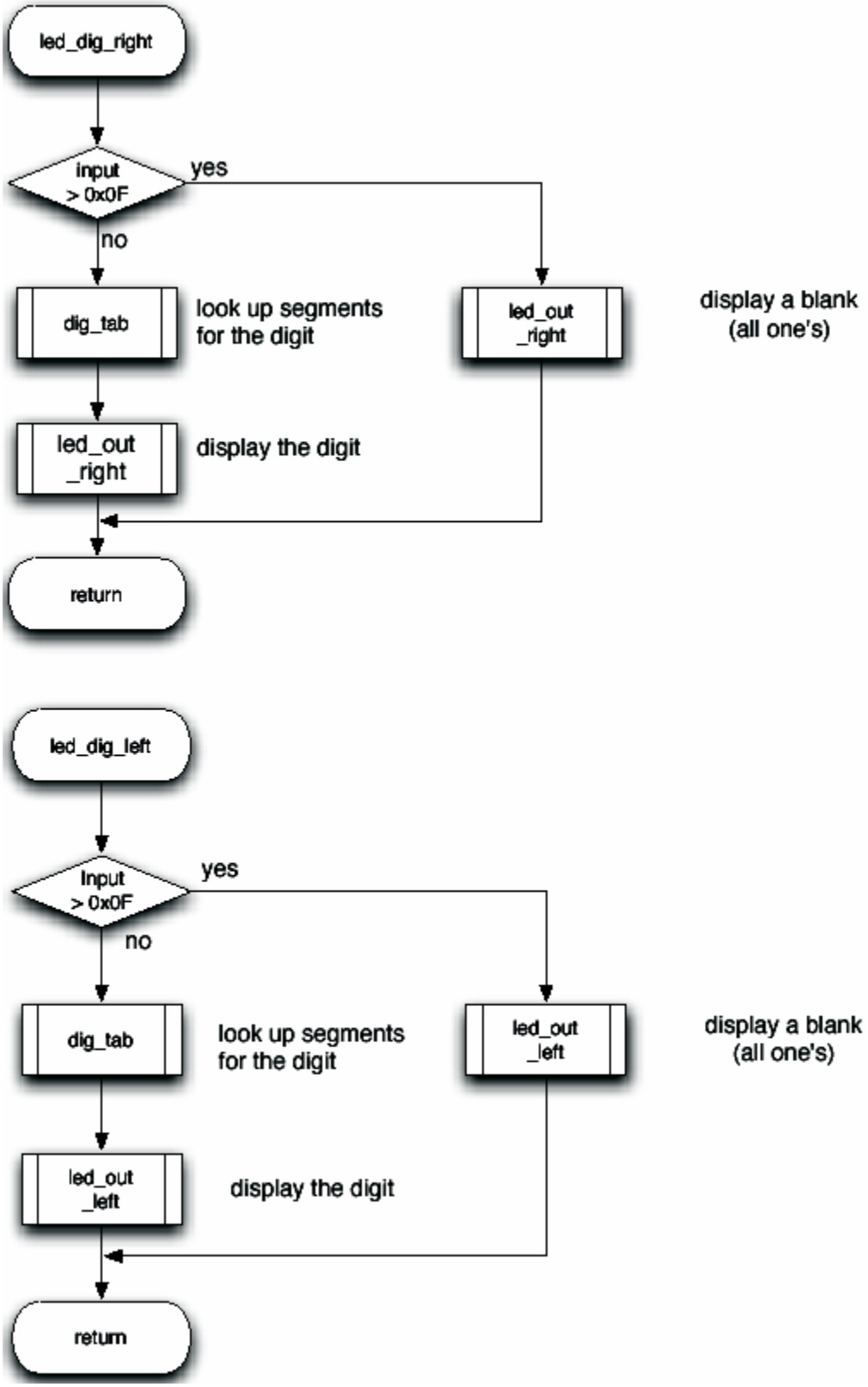


Figure 33. Safety_2.3.3.5 Port_User_LED

Safety_2.3.3.5 Port_User_led

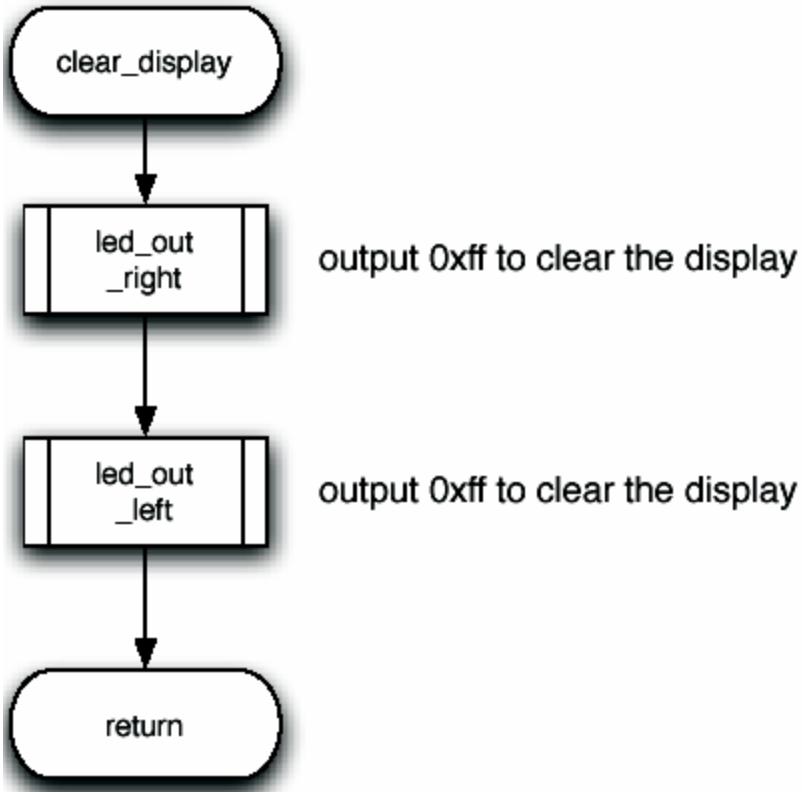
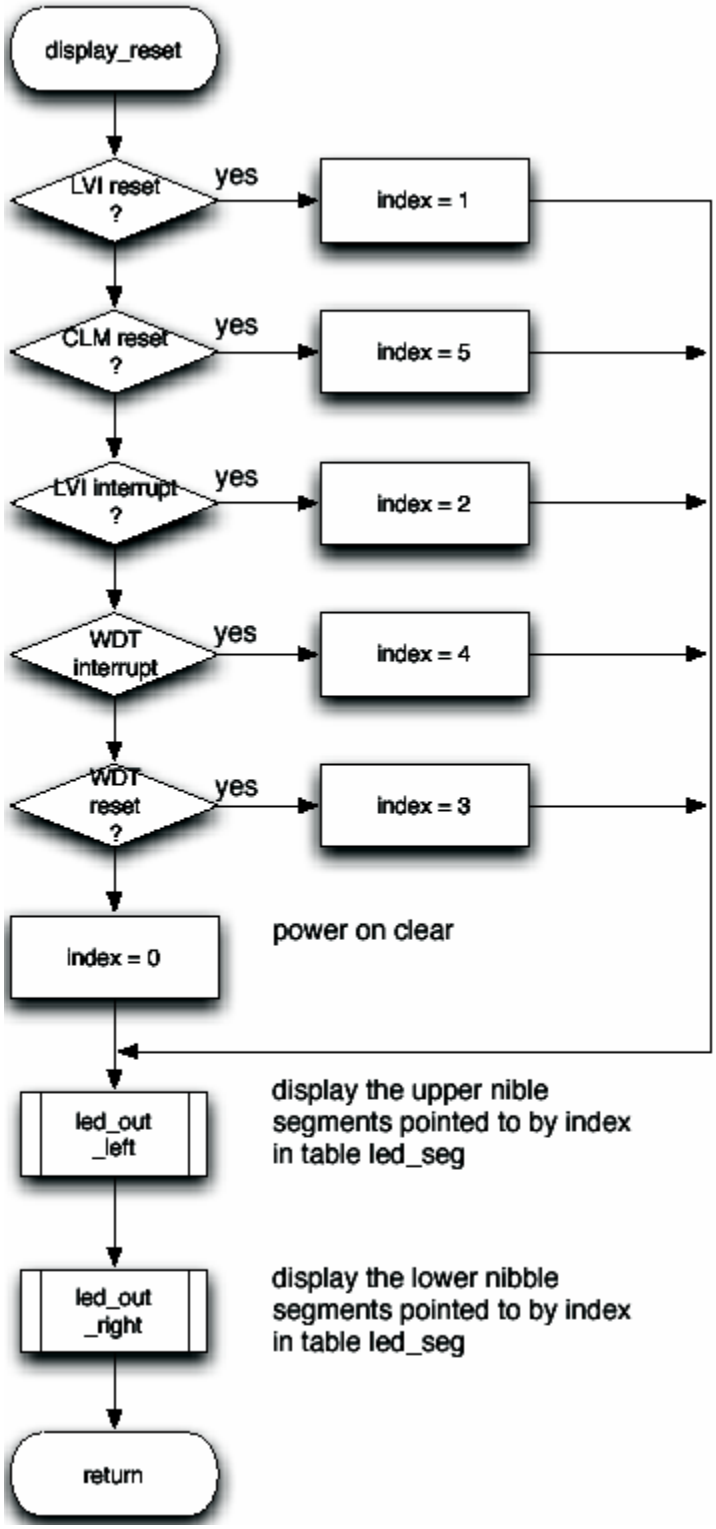


Figure 34. Safety_2.3.3.6 Port_User_LED

Safety_2.3.3.6 Port_User_led



4.14 Watchtimer Initialize

Figure 35. Safety_2.4.1.1 Watchtimer_Init

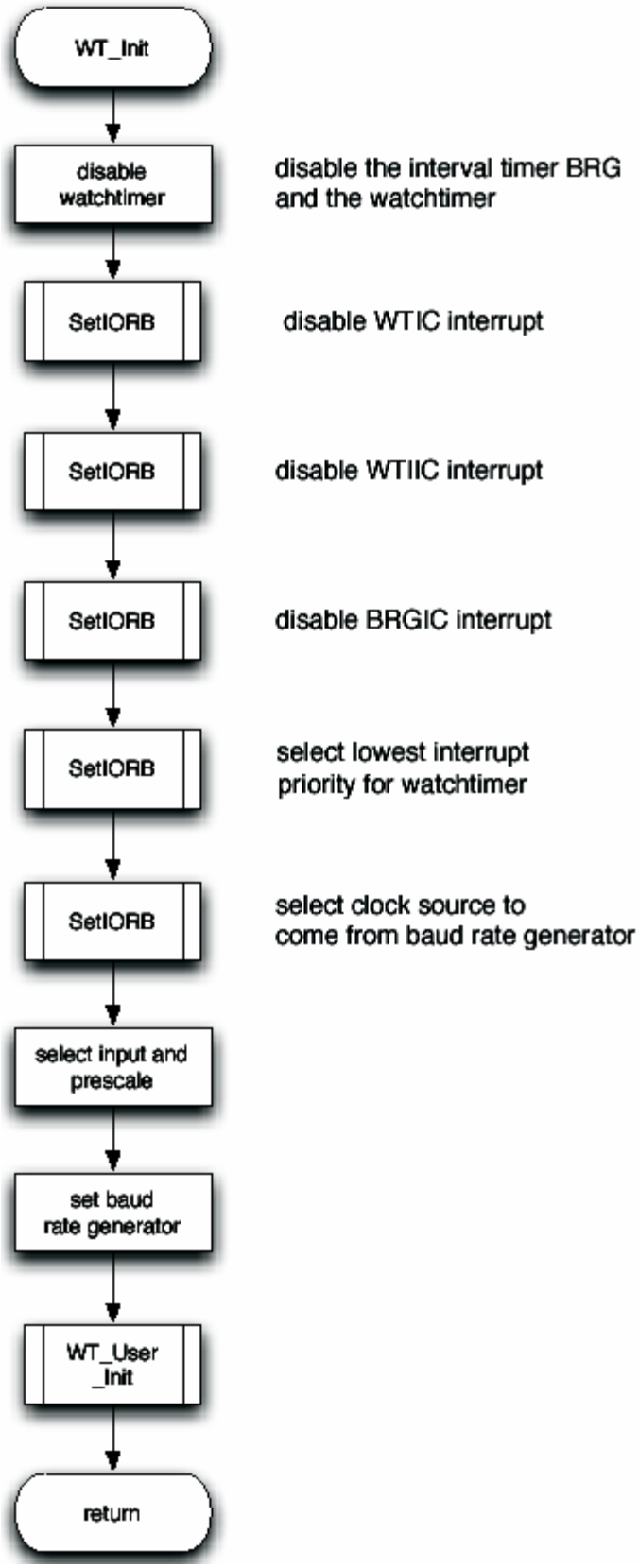
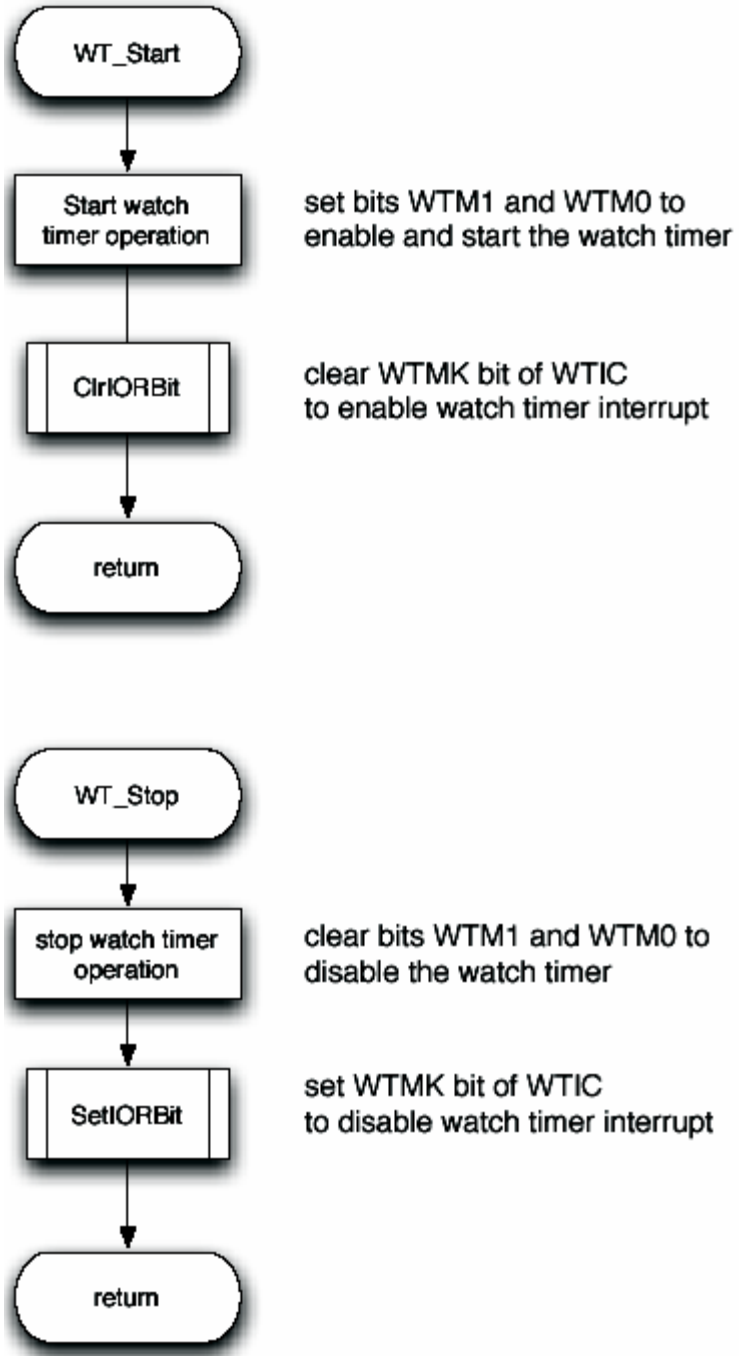


Figure 36. Safety_2.4.1.2 Watchtimer_Init

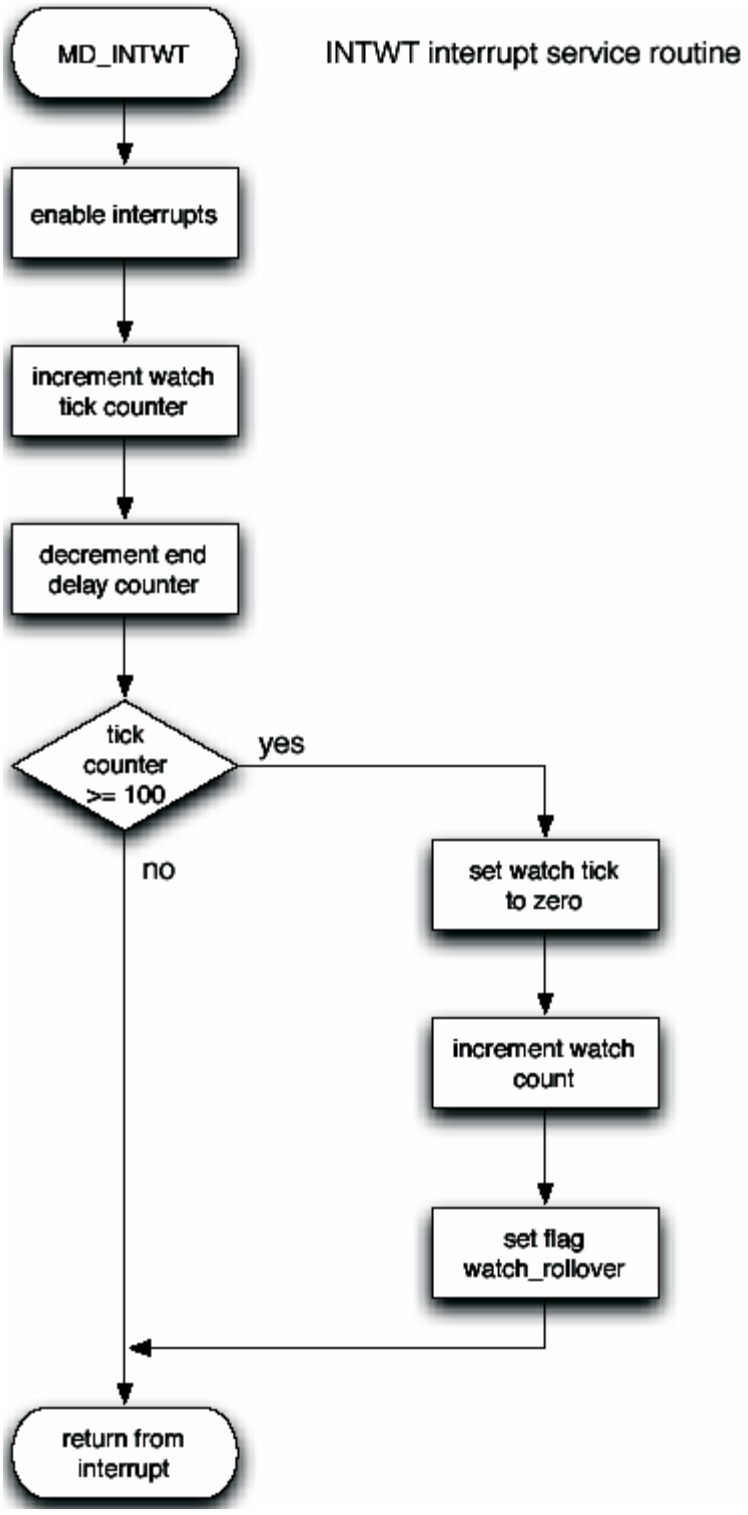
Safety_2.4.1.2 Watchtimer_Init



4.15 Watchtimer User

Figure 37. Safety_2.4.2.2 Watchtimer_User

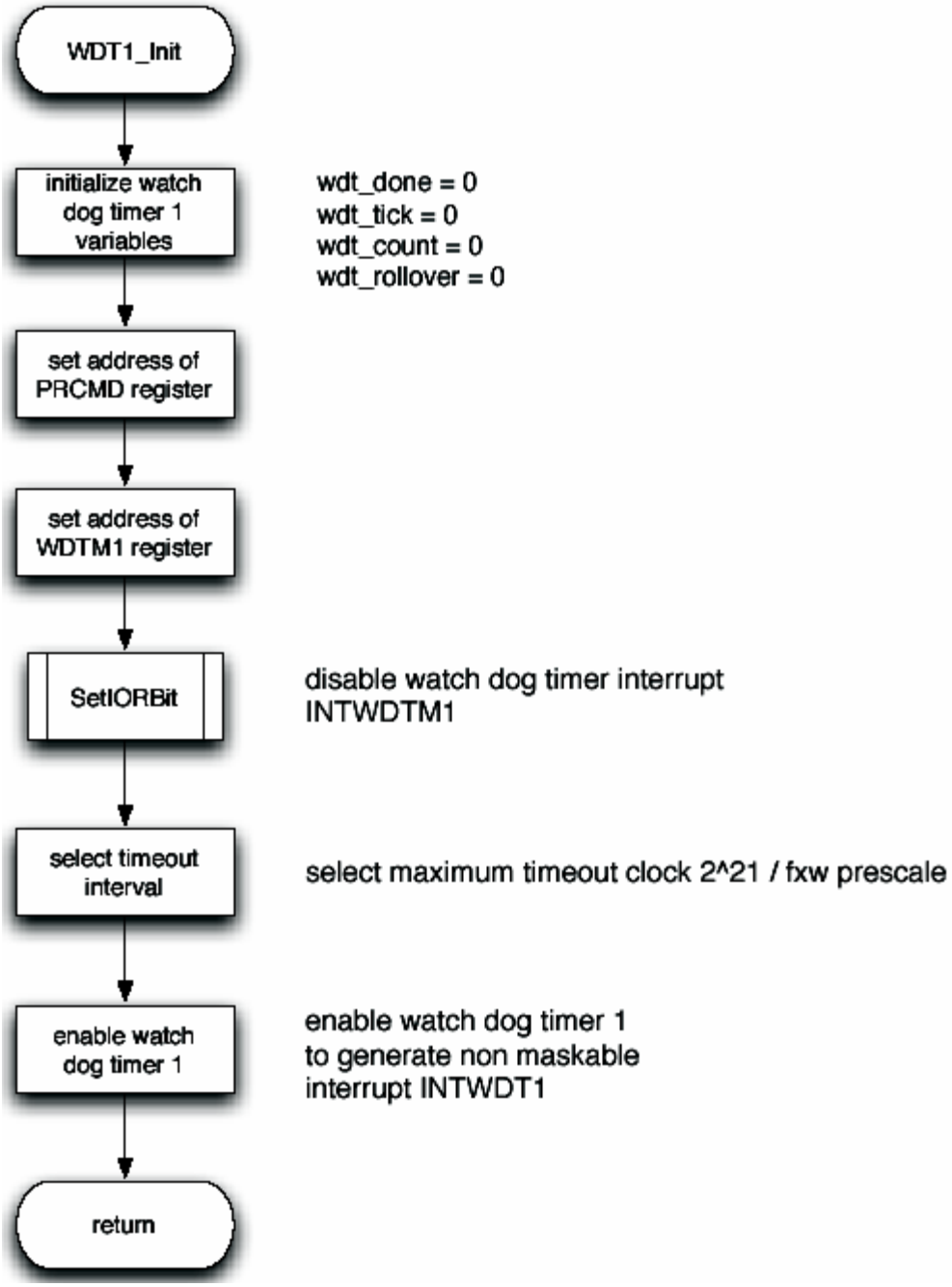
Safety_2.4.2.2. Watchtimer_User



4.16 Watchdog 1 Initialize

Figure 38. Safety_2.5.1 Watchdog1_Init

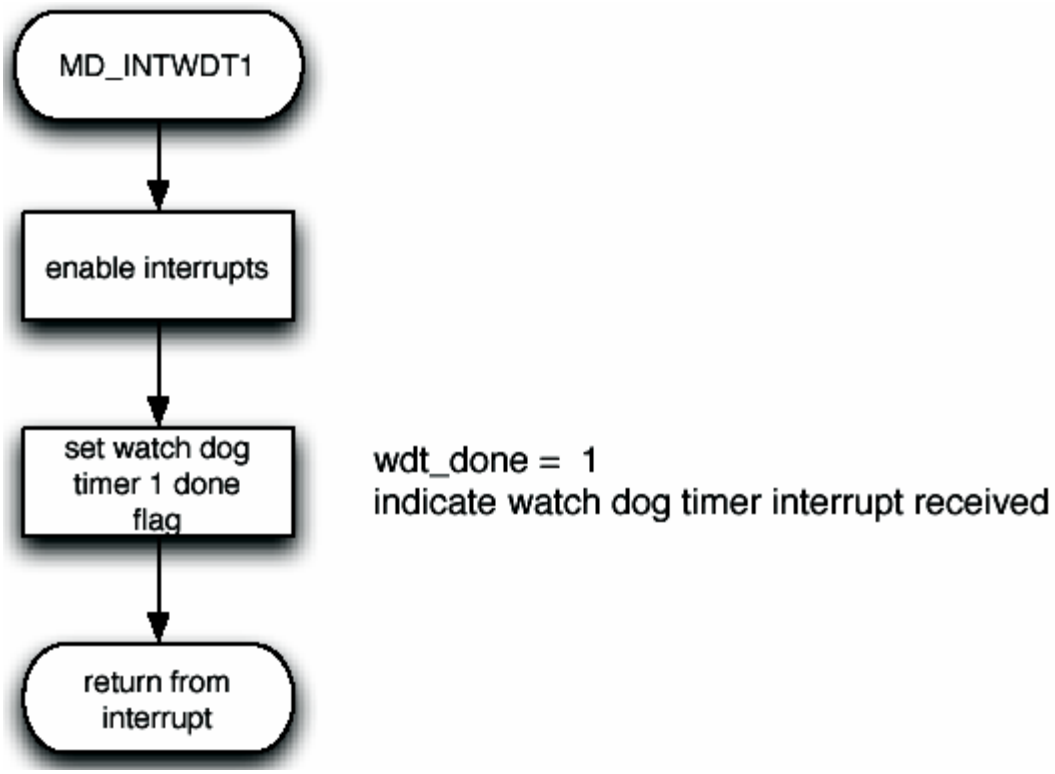
Safety_2.5.1 Watchdog1_Init



4.17 Watchdog 1 User

Figure 39. Safety_2.5.2 Watchdog1_User

Safety_2.5.2 Watchdog1_User



4.18 LVI Initialize

Figure 40. Safety_2.6.1.1 LVI_Init

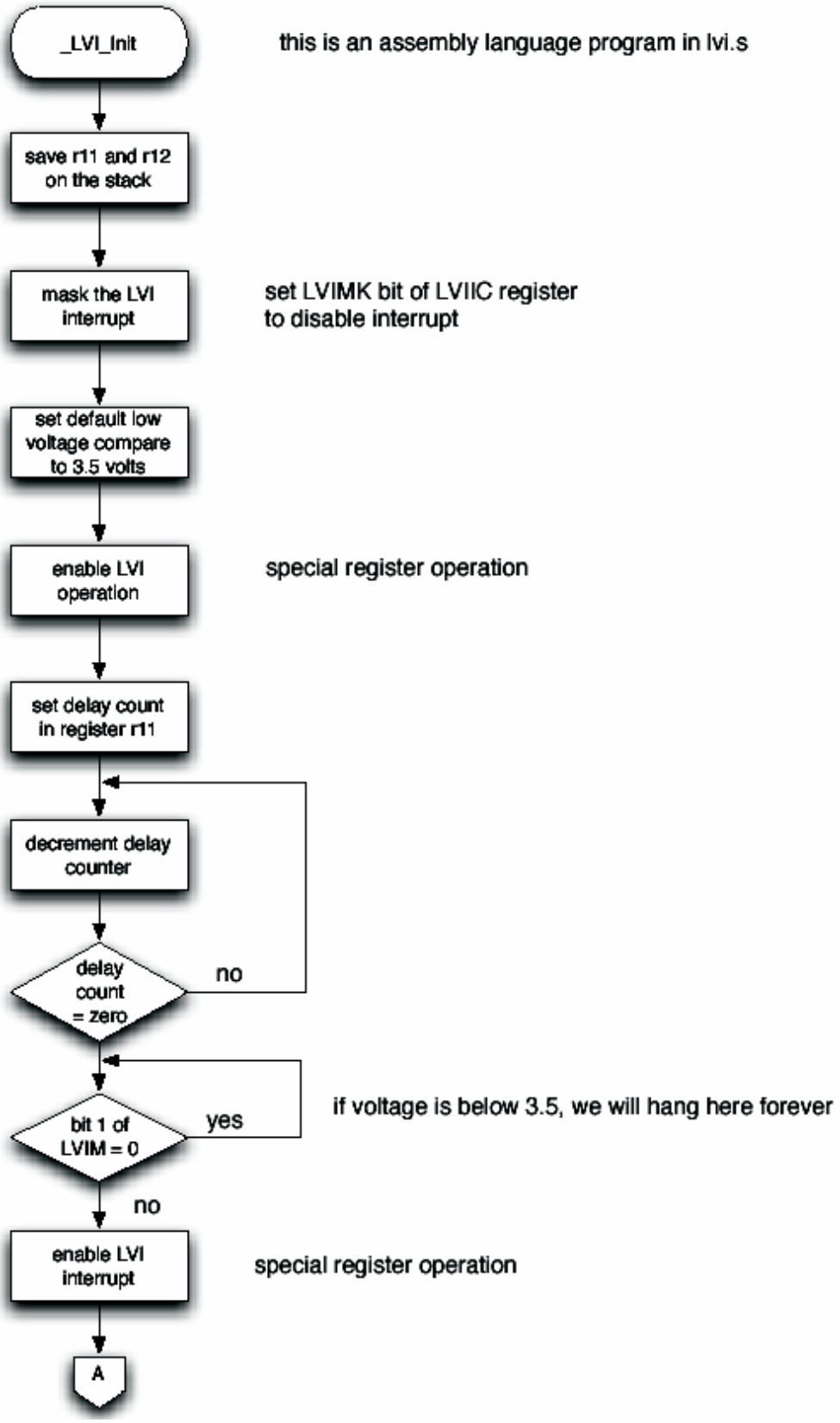
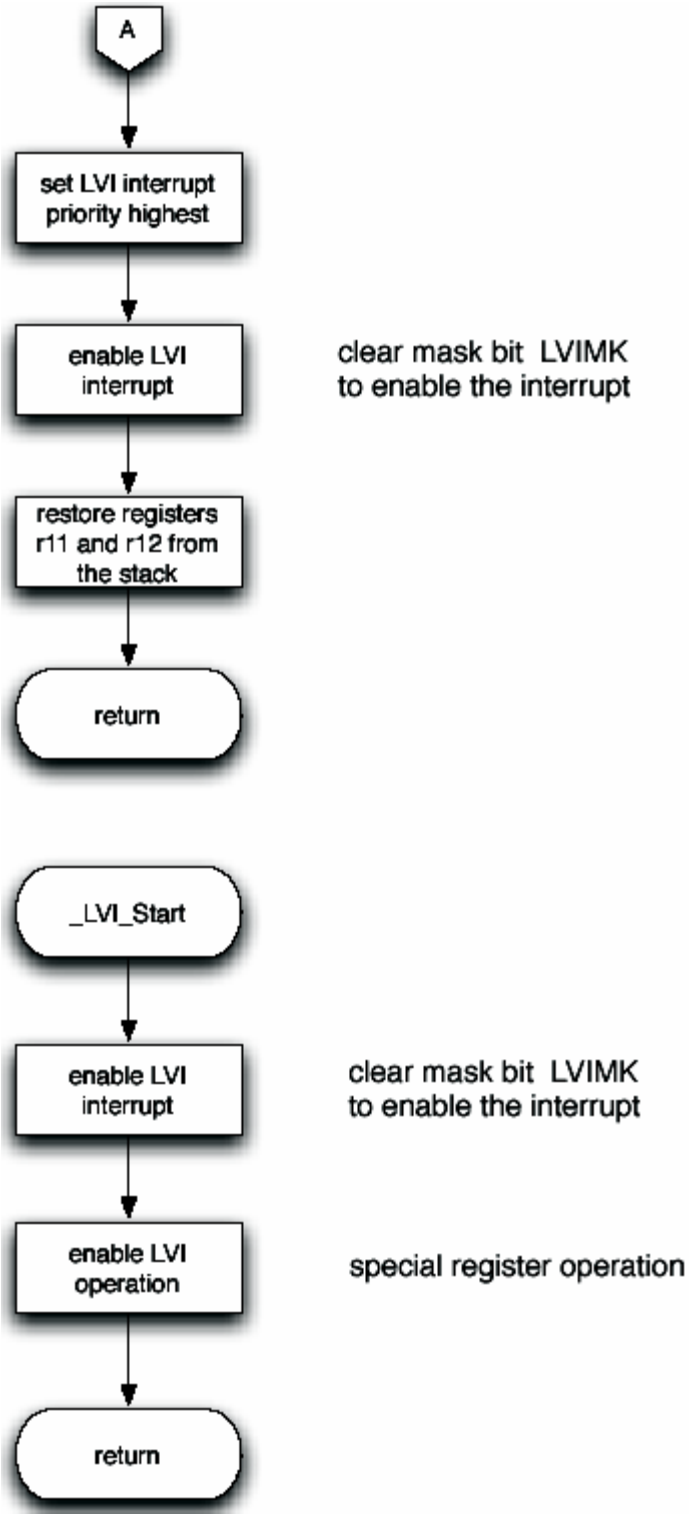


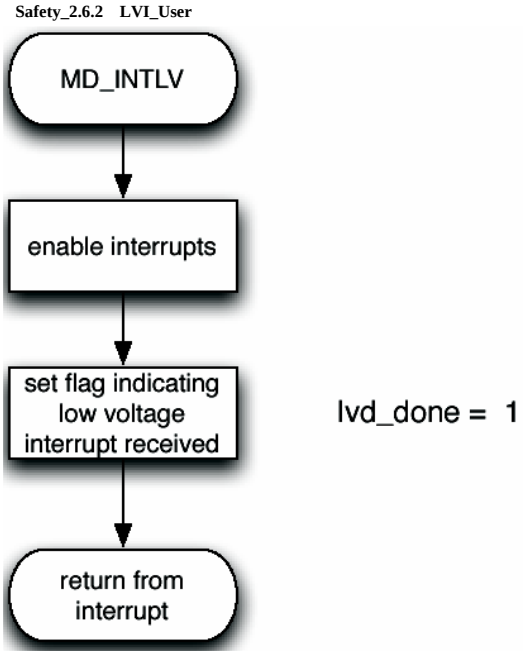
Figure 41. Safety_2.6.1.2 LVI_Init

Safety_2.6.1.2 LVI_Init



4.19 LVI User

Figure 42. Safety_2.6.2 LVI_User



4.20 Write Special

Figure 43. Safety_2.7.1.1 Write_Special

Safety_2.7.1.1. Write_Special

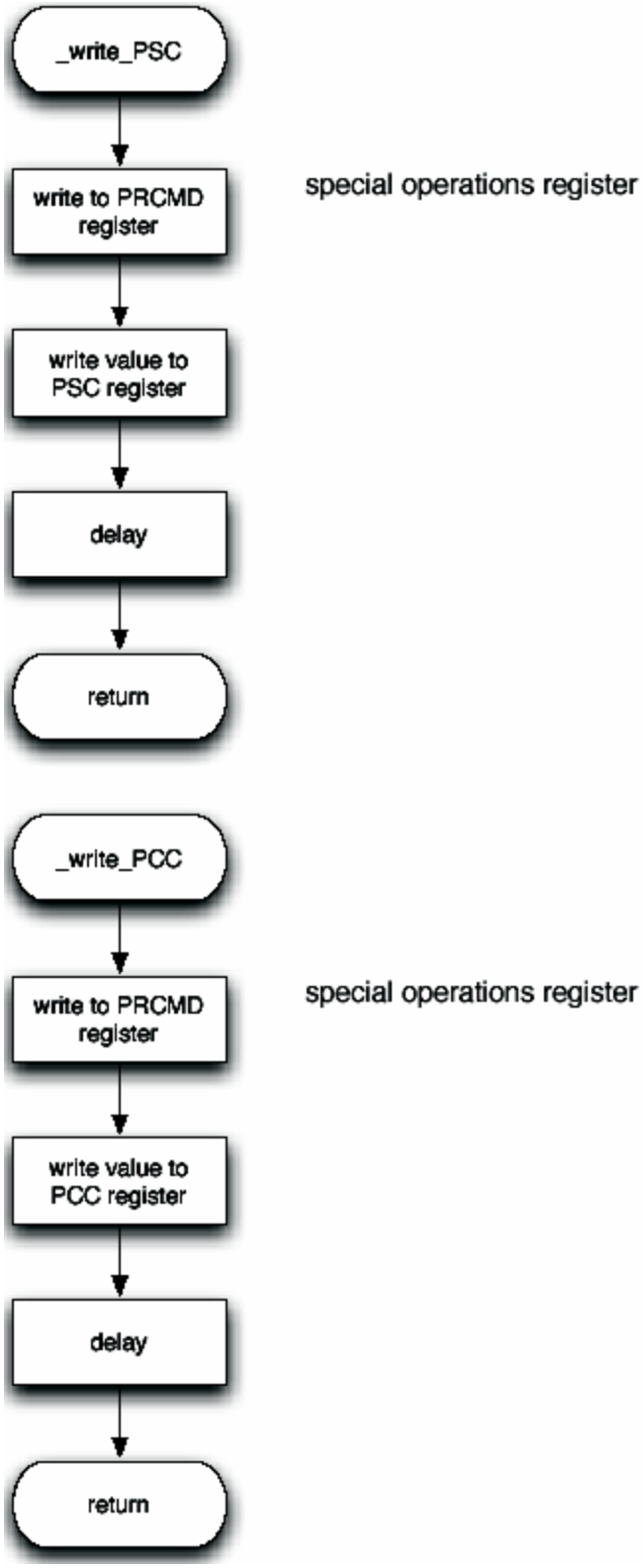


Figure 44. Safety_2.7.1.2 Write_Special

Safety_2.7.1.2 Write_Special

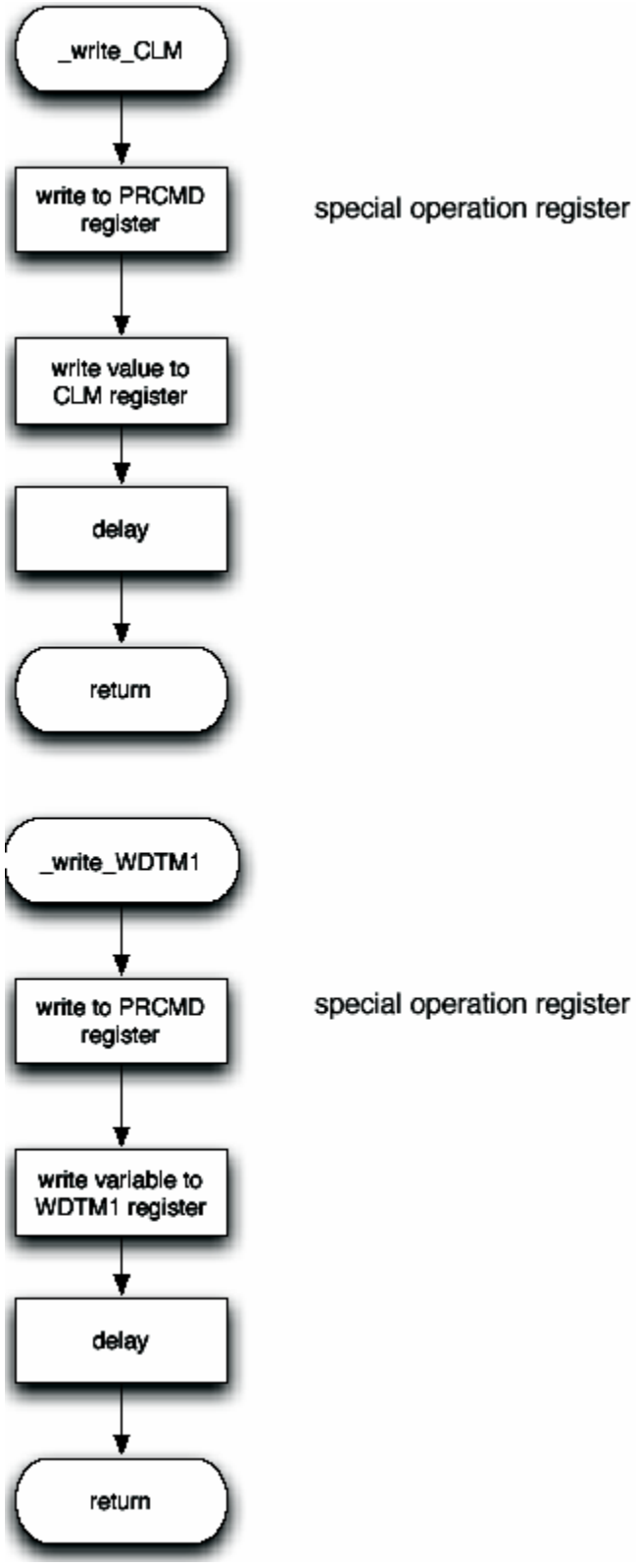
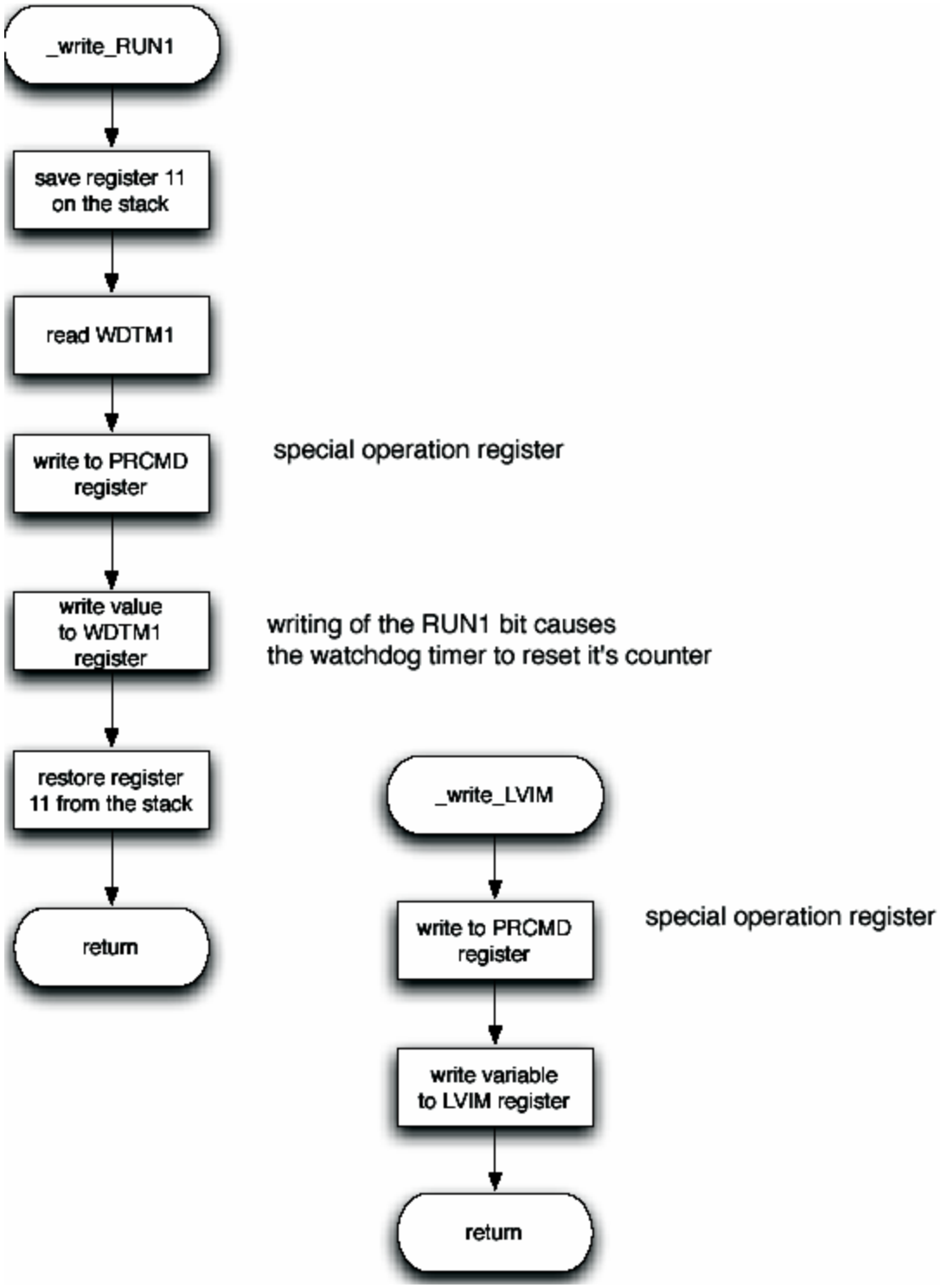


Figure 45. Safety_2.7.1.3 Write_Special

Safety_2.7.1.3 Write_Special



4.21 Pre-Processing

Figure 46. Safety_3.1.1 Pre-Processing

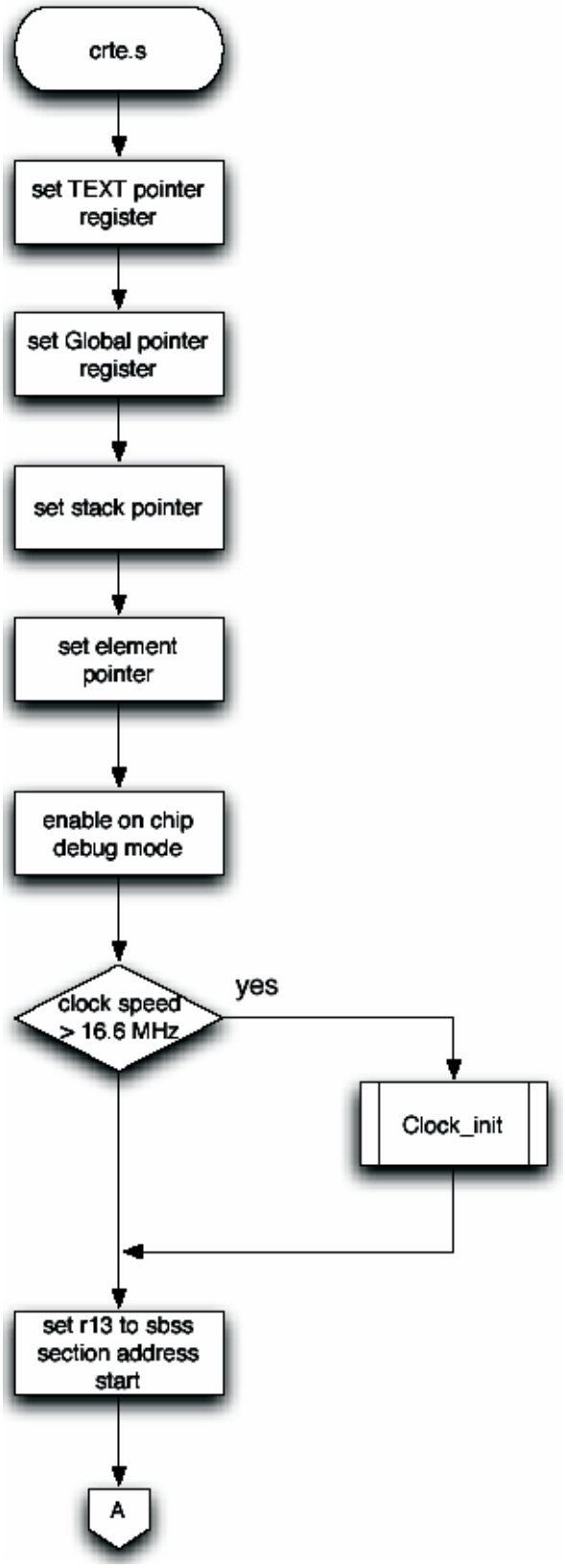


Figure 47. Safety_3.1.2 Pre-Processing

Safety_3.1.2 Pre-Processing

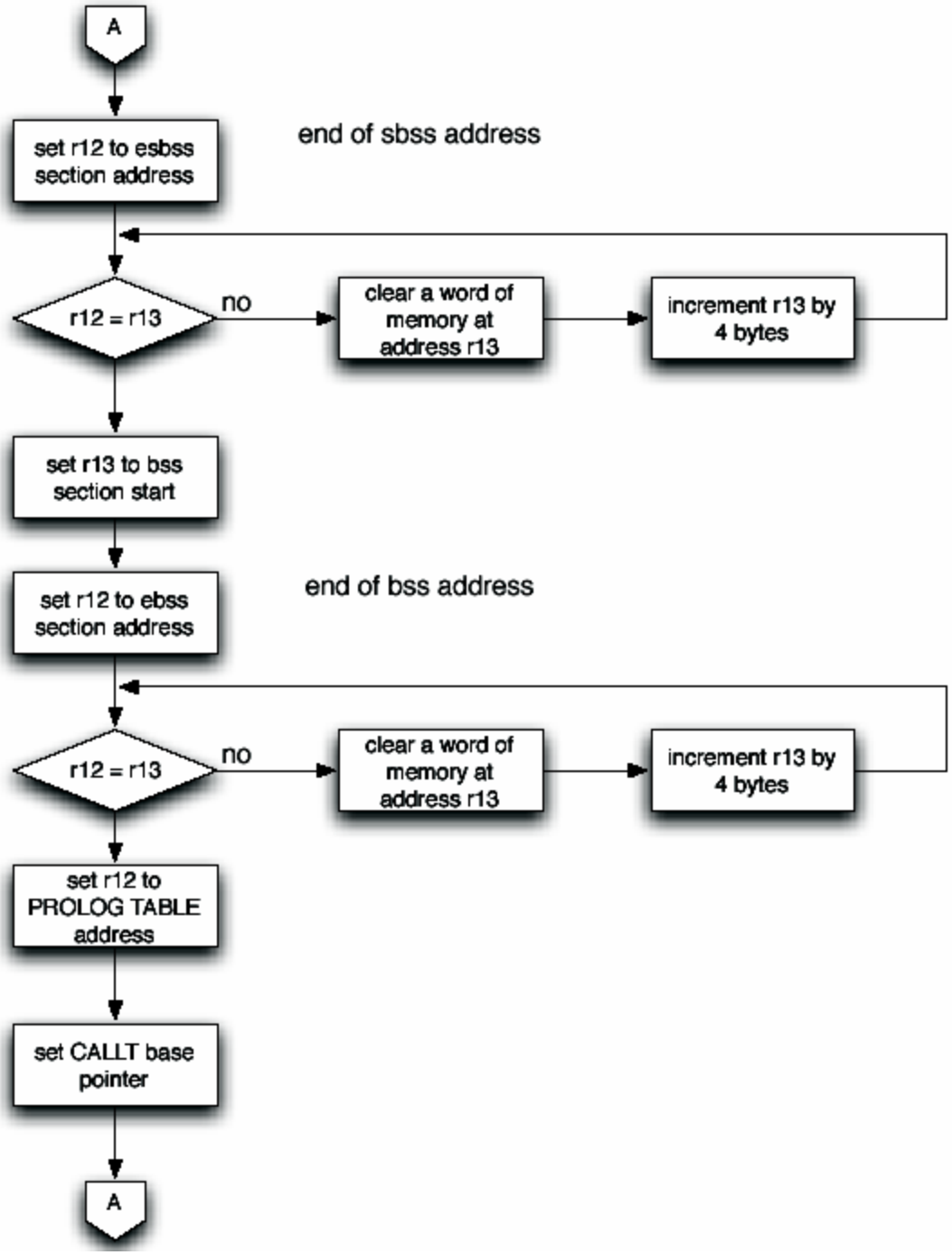
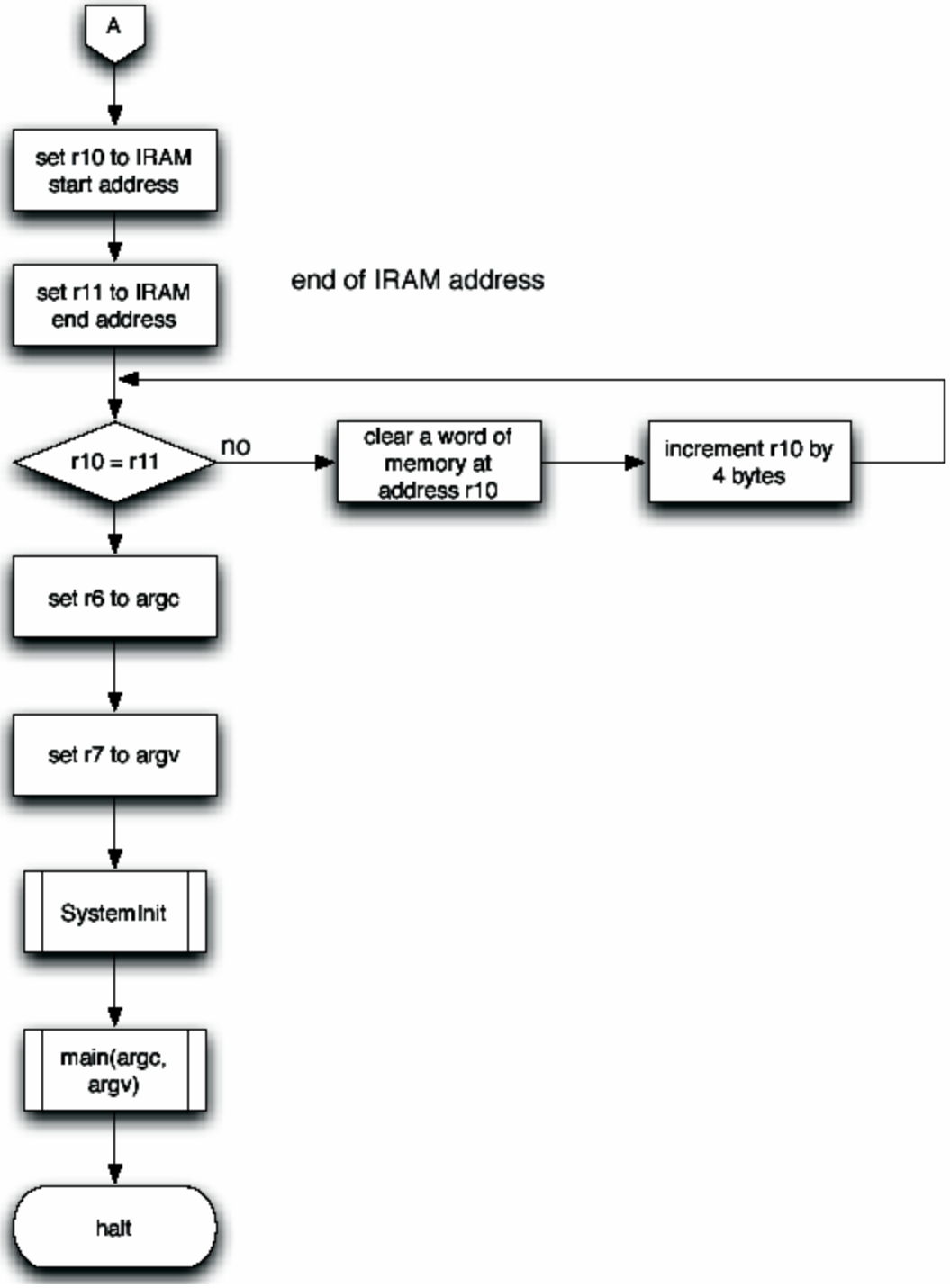


Figure 48. Safety_3.1.3 Pre-Processing

Safety_3.1.3 Pre-Processing



5. Appendix B – Applilet Tool

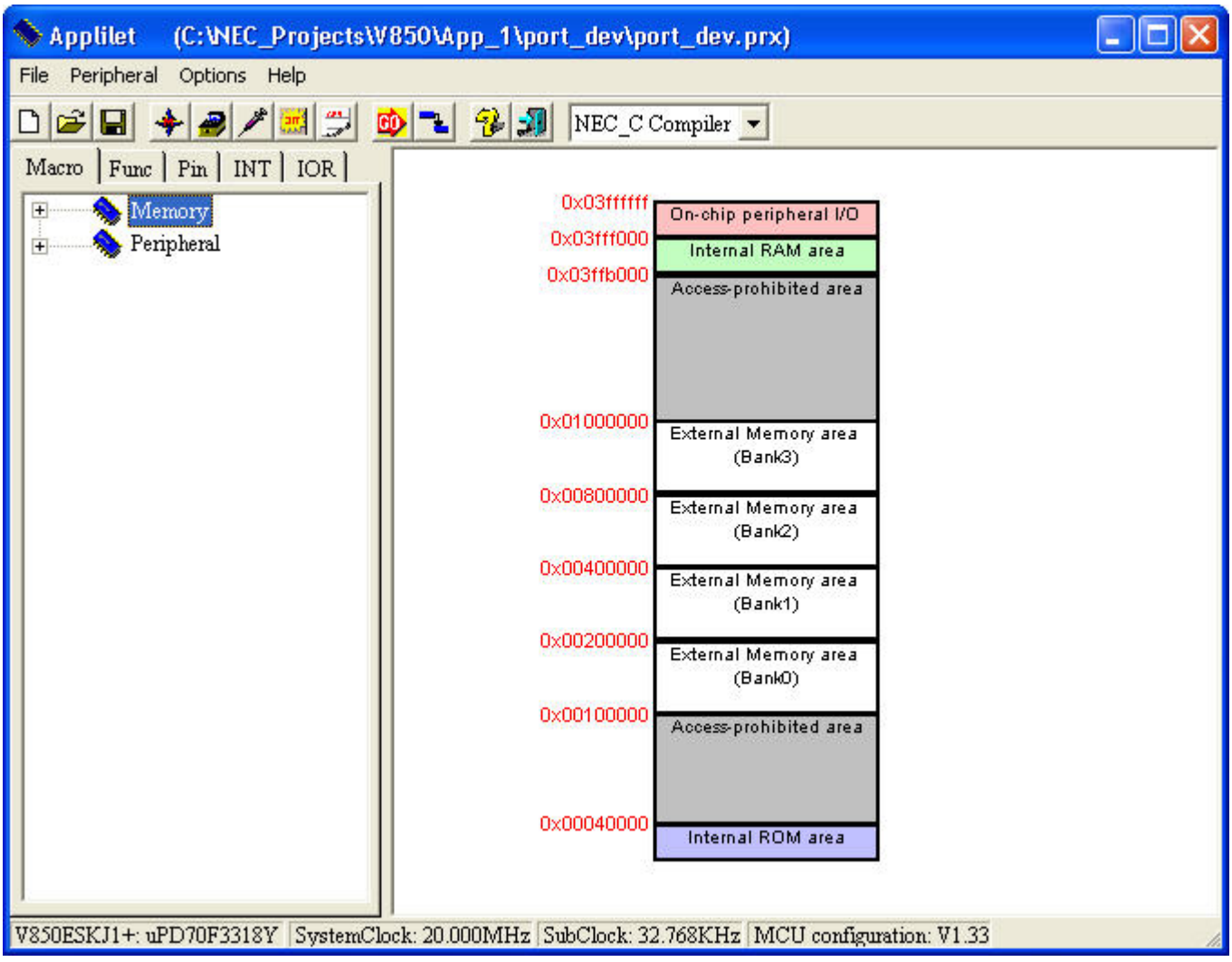
The Applilet tool provides a GUI-based environment for generating a base framework of code. As shown in the following series of screen captures, you can select the features needed for your application and get a quick start in creating your code. The screens shown below are only for the features applicable to this demonstration.

5.1 System Memory

To enable the Applilet to include the options available for your MCU, you need to import the MCU information. As shown at the bottom of the screen below, this demonstration uses the V850ESKJ1+ uPD70F3318Y MCU. This screen is just informational and shows how the internal memory is laid out in the processor. Other Option menu items are:

Options | Compilers | NEC Compiler
Options | Source Code Language | C Language
Options | Import MCU

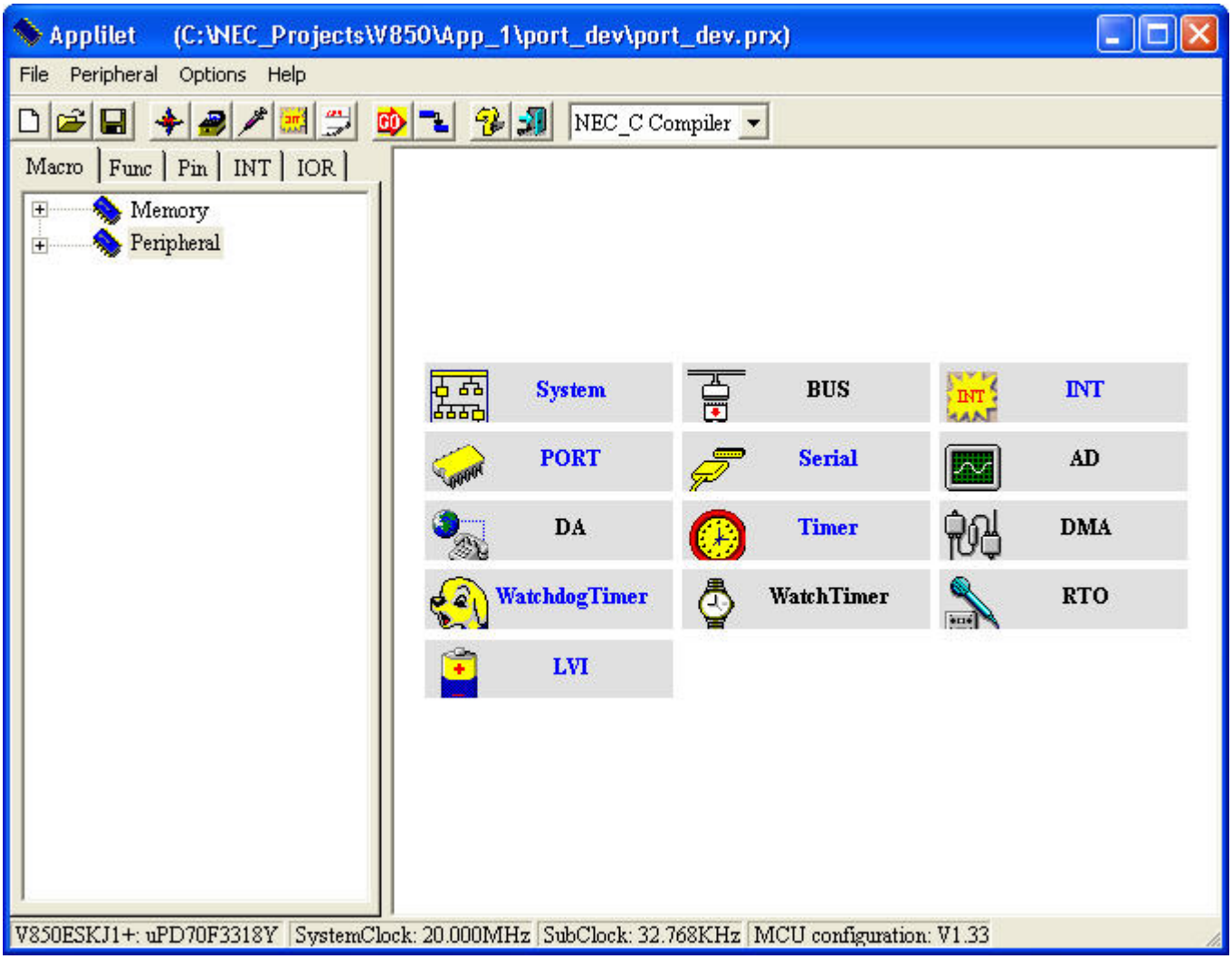
Figure 49. Applilet Screen Showing Memory Layout



5.2 System Peripherals

The Applilet screen below is your starting point. From this screen you select each of the peripheral subsystems. It is best to start with the system first since the clock selections made here are used by the other peripherals.

Figure 50. Selecting Peripherals

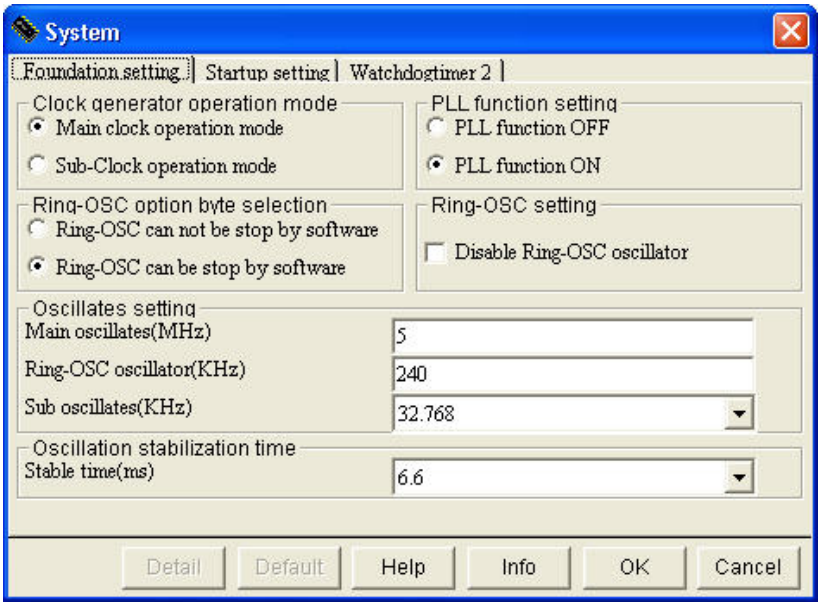


5.3 System

5.3.1 System Foundation Settings

After you click **System**, the Applilet's **Foundation setting** tab allows you to select various clock operations. When you start the demonstration, you want to be in high-speed mode, so select the main clock option and the PLL multiplier. You also want to be flexible, so allow the ring oscillator to be stopped by software. The main oscillator selection should be based on the crystal used to drive the main clock of the microcontroller. The same is true for the sub-clock oscillator crystal or whatever method you use to generate the low frequency for that clock source.

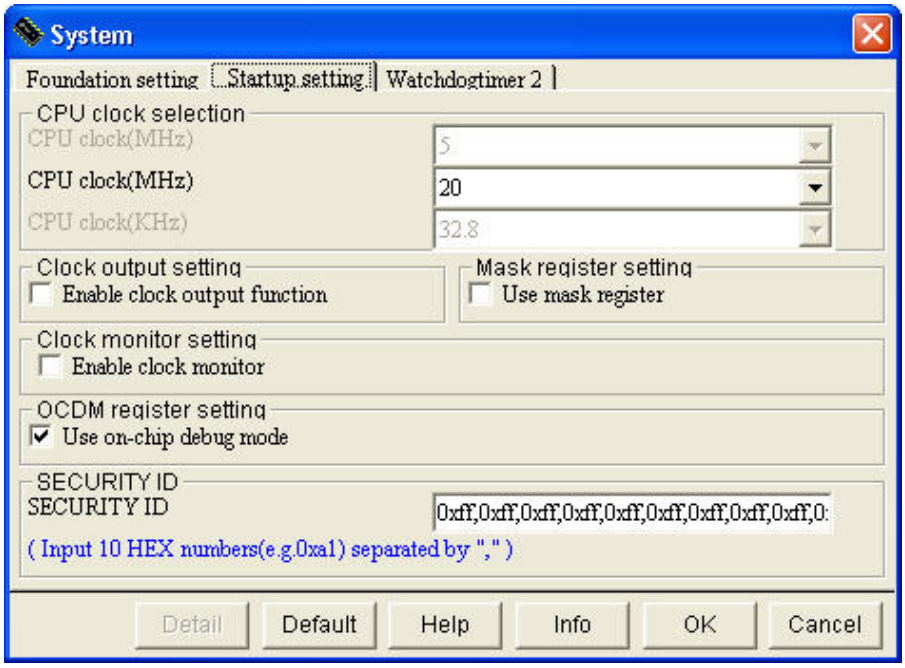
Figure 51. System Foundation Settings



5.3.2 System Startup Settings

When you click on the **Startup setting** tab, the Applilet shows **CPU clock(MHz)** at "20" because you previously selected a 5MHz main clock with PLL on. Under **OCDM register setting**, allow on-chip debug mode since you are working with development code. To be safe, leave the Security ID as all ONEs; you do not want to pick a strange value and then forget what it was.

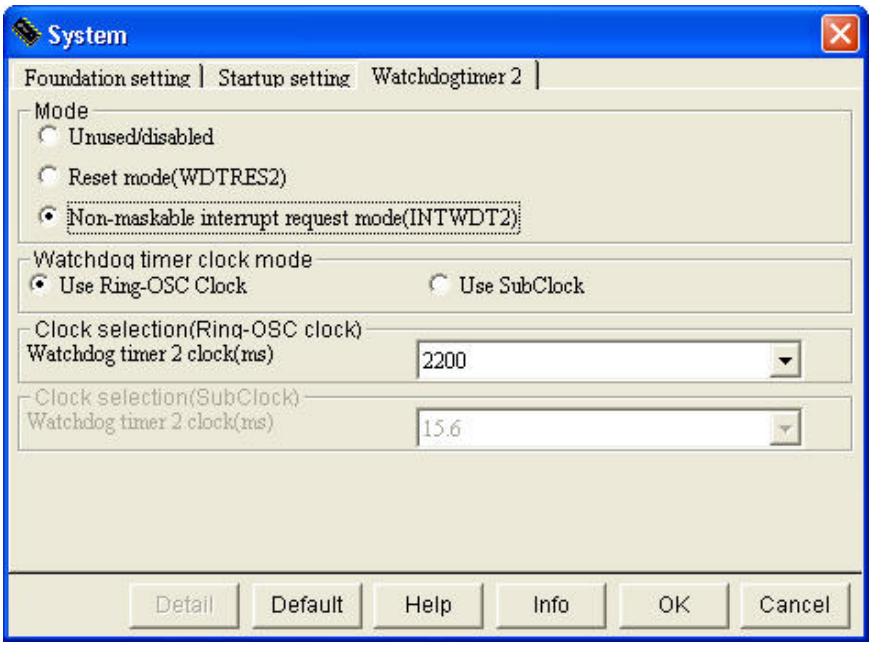
Figure 52. System Startup Settings



5.3.3 System Watchdog Settings

After clicking the **Watchdogtimer 2** tab, select the non-maskable interrupt. Select the ring oscillator as the clock source for Watchdog Timer 2. Select the longest delay possible from the clock selection pull down menu.

Figure 53. System Watchdog Settings



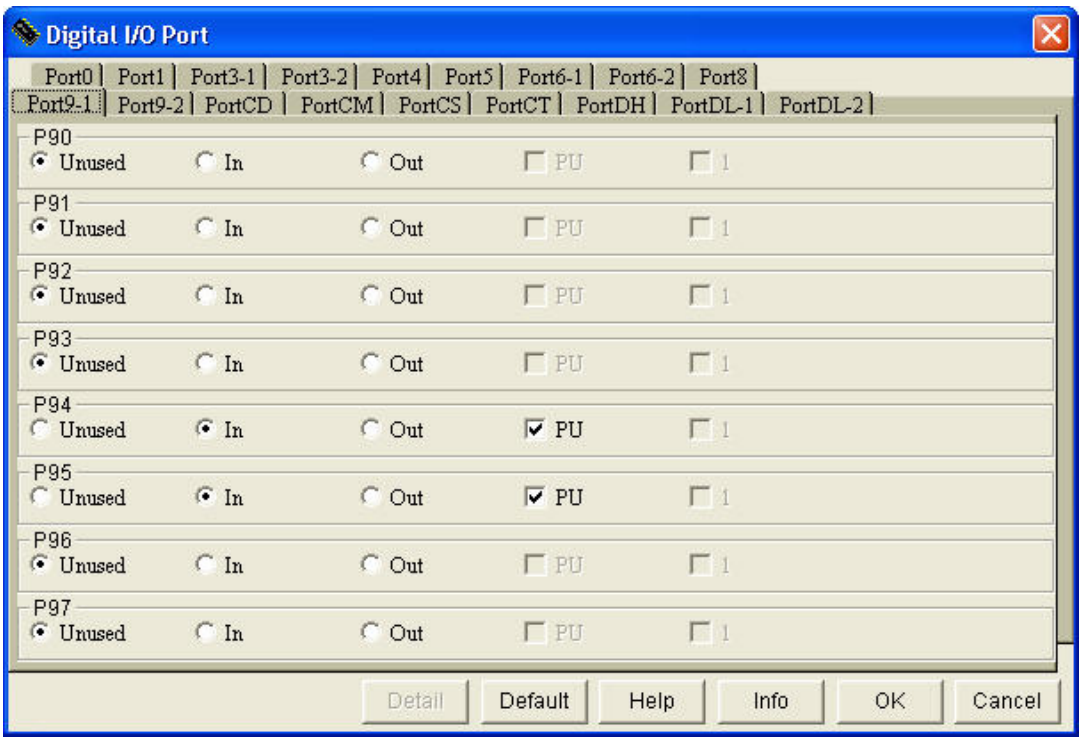
5.4 Port Subsystem

Clicking the **Port** button on the main Applilet screen gives you access to port settings.

5.4.1 Port 9-1 Selections

Port 9-1 connects to the pushbuttons (SW2 and SW3). Note in the figure below that the pull-ups are enabled. With this setting, you do not need extra pull-up resistors on the switches. Pressing the switch pulls the input to ground. Pin P94 connects to SW2 on the M-Station (the left switch) and P95 connects to SW3 (the right switch).

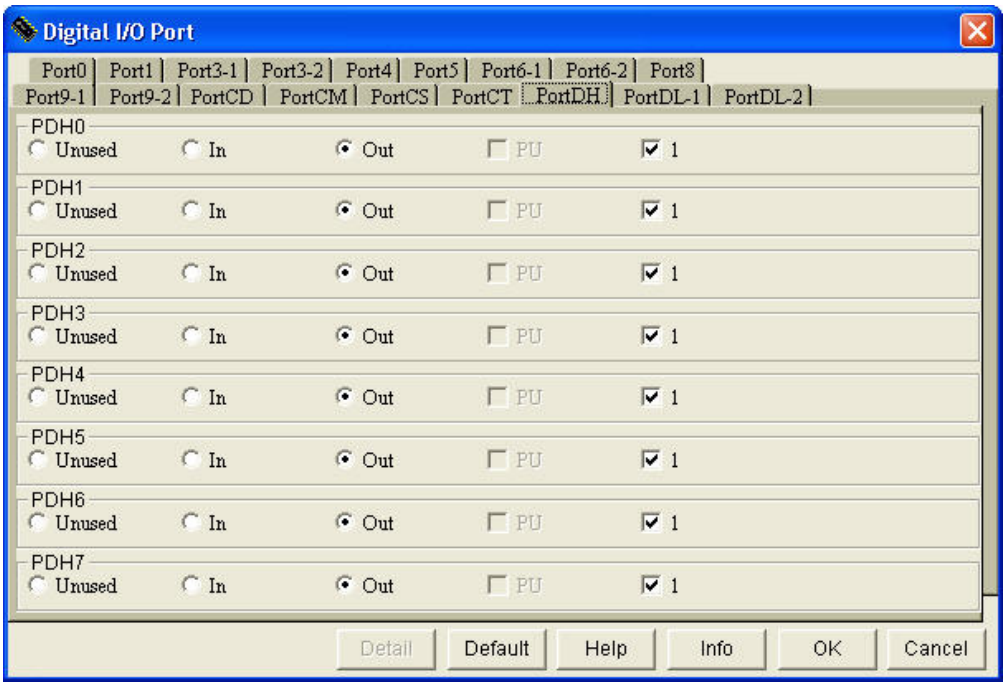
Figure 54. Port 9-1 Selections



5.4.2 Port DH Selections

Go to the tab for Port DH to set up this port as an output for the right LED. When the port pin outputs a 1, the LED segment turns off. Outputting a 0 causes the port pin to sink current, and the LED turns on. On each pin, select a 1 for the initial output so that the LEDs do not come on until they are supposed to.

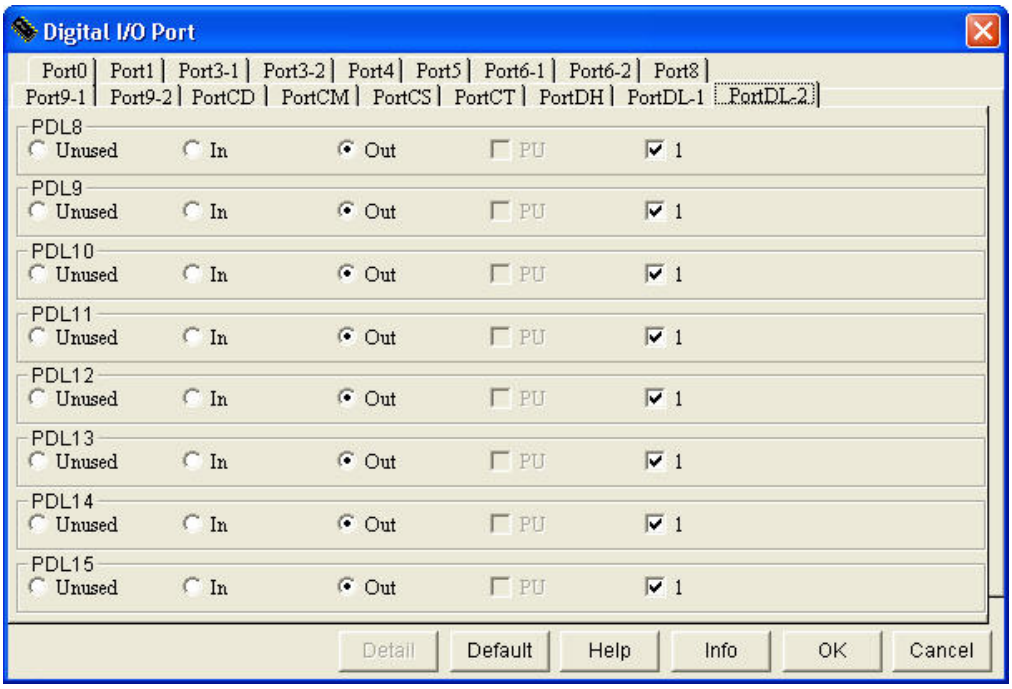
Figure 55. Port DH Selections



5.4.3 Port DL-2 Selections

Using the Port DL-2 tab, set up this port as an output for the left LED with the same settings as for Port DH above.

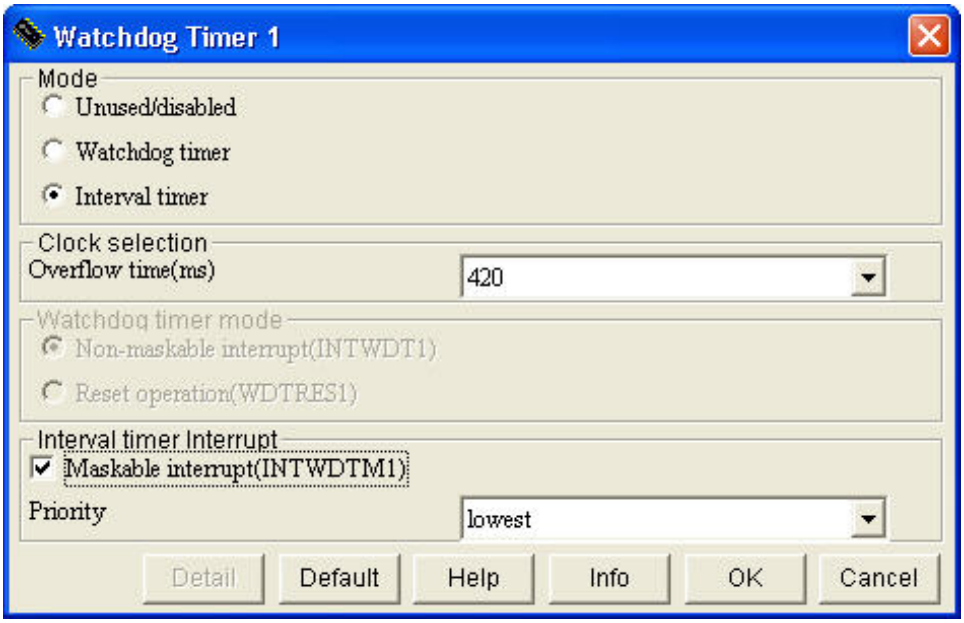
Figure 56. Port DL-2 Selections



5.5 Watchdog Timer 1 Selections

Select the watchdog timer to operate as an interval timer for now. You can change this selection later, if desired.

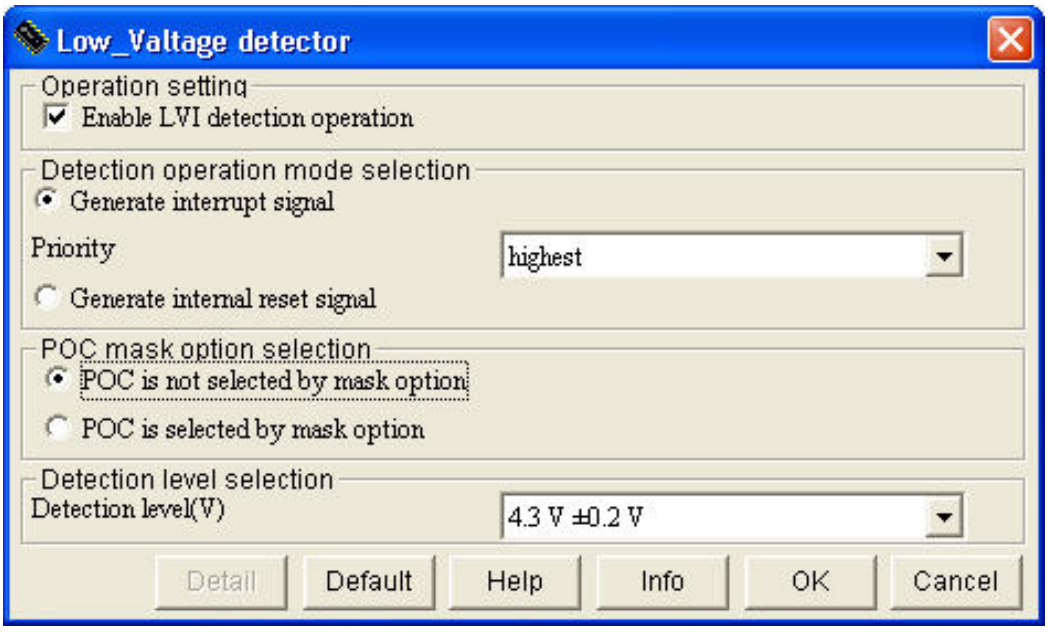
Figure 57. Watchdog Timer 1 Selections



5.6 Low-Voltage Detect Selections

When picking the initial conditions for the LVI tests, you do not want power-on clear to occur when low voltage is detected. The first LVI test is at a voltage threshold of 4.3V.

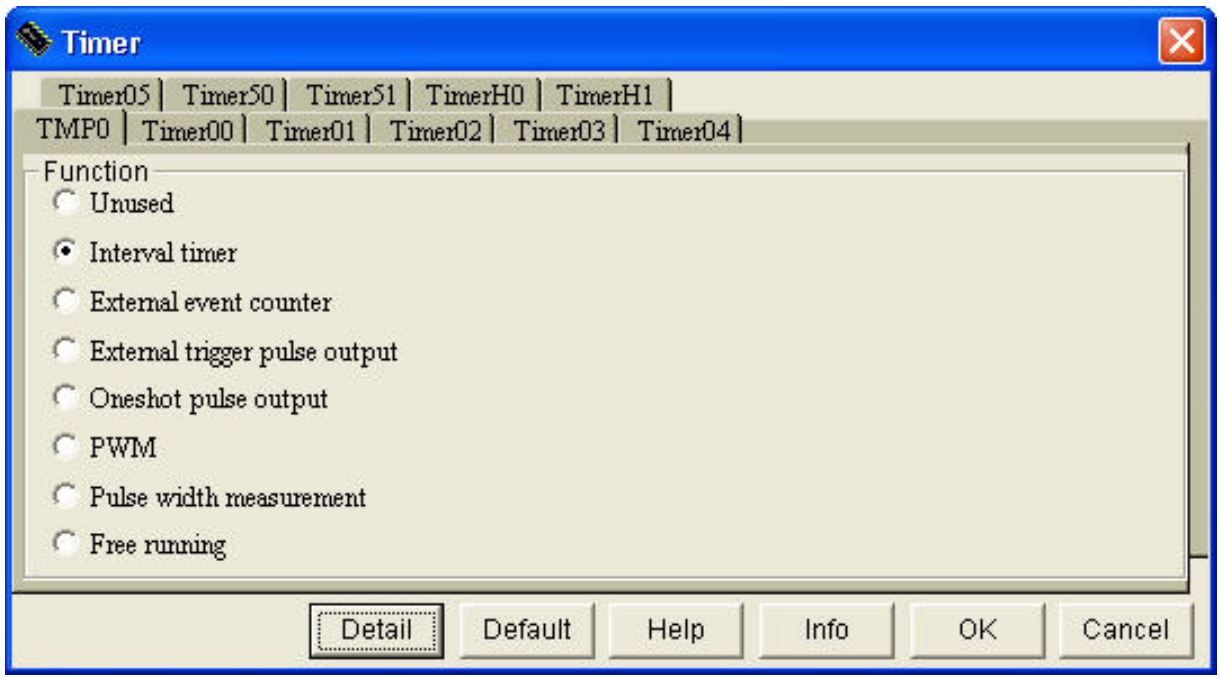
Figure 58. Low-Voltage Detect Selections



5.7 Timer Subsystem

This demonstration uses timer TMP0 as an interval timer from the timer subsystem.

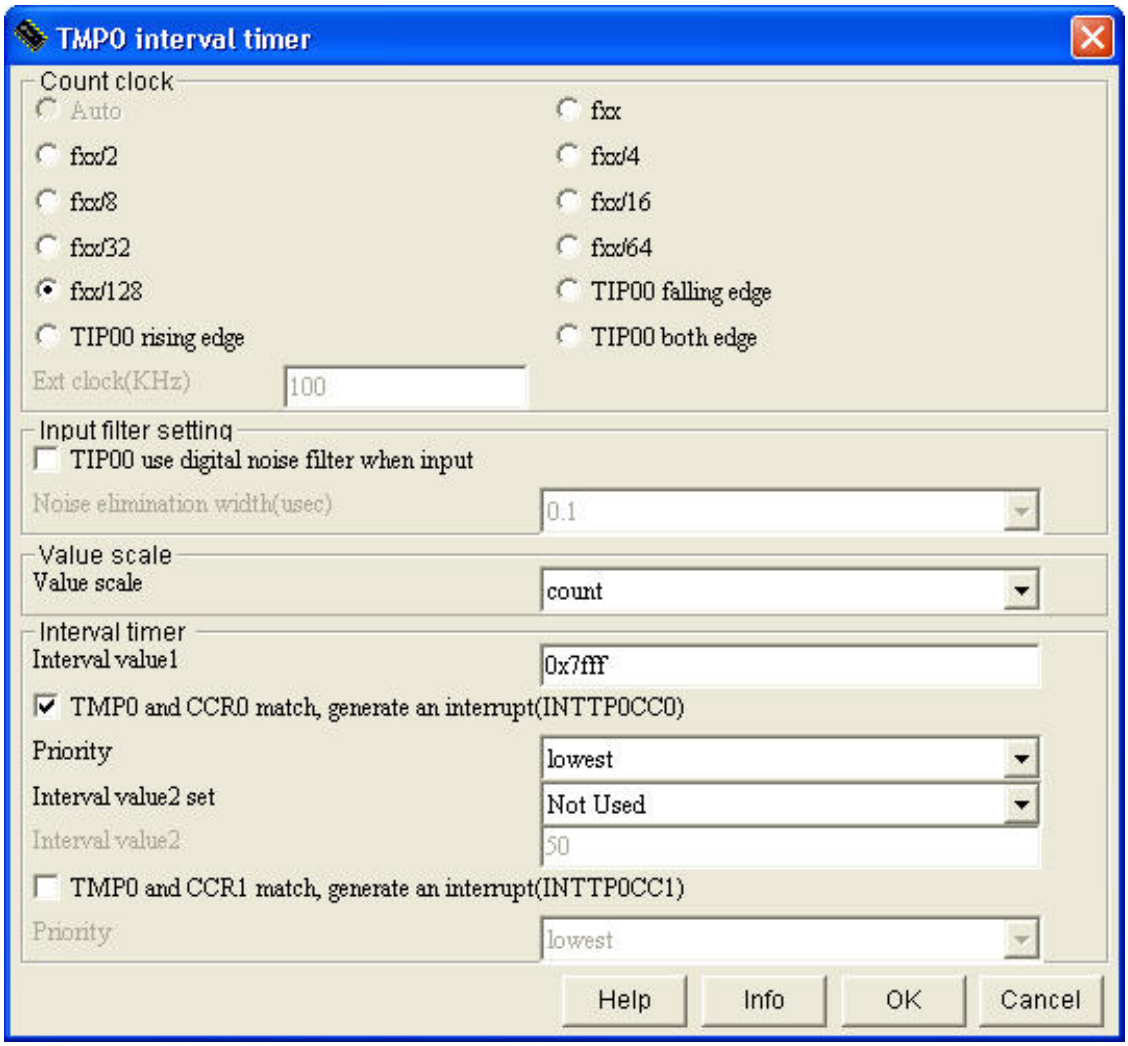
Figure 59. Timer Subsystem



5.7.1 Timer TMP0 Selections

For timer TMP0, you want to divide the system clock down as much as possible. Use the interval timer as a counter compare. Interrupt when the count and interval value 1 match.

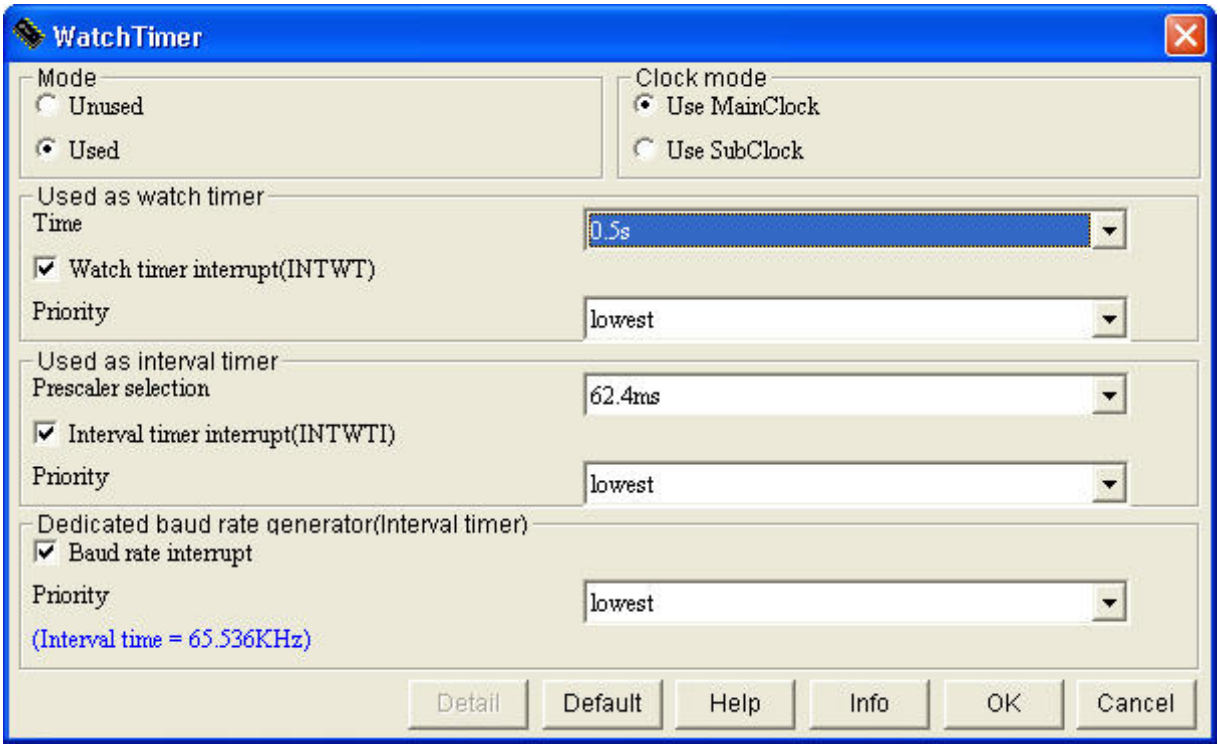
Figure 60. Timer TMP0 Selections



5.8 Watch Timer Selections

The initial selection for the watch timer is to run from the main system clock. You want a slow watch timer for long delays. The interval timer runs a little faster for blinking and shorter delays.

Figure 61. Watch Timer Selections

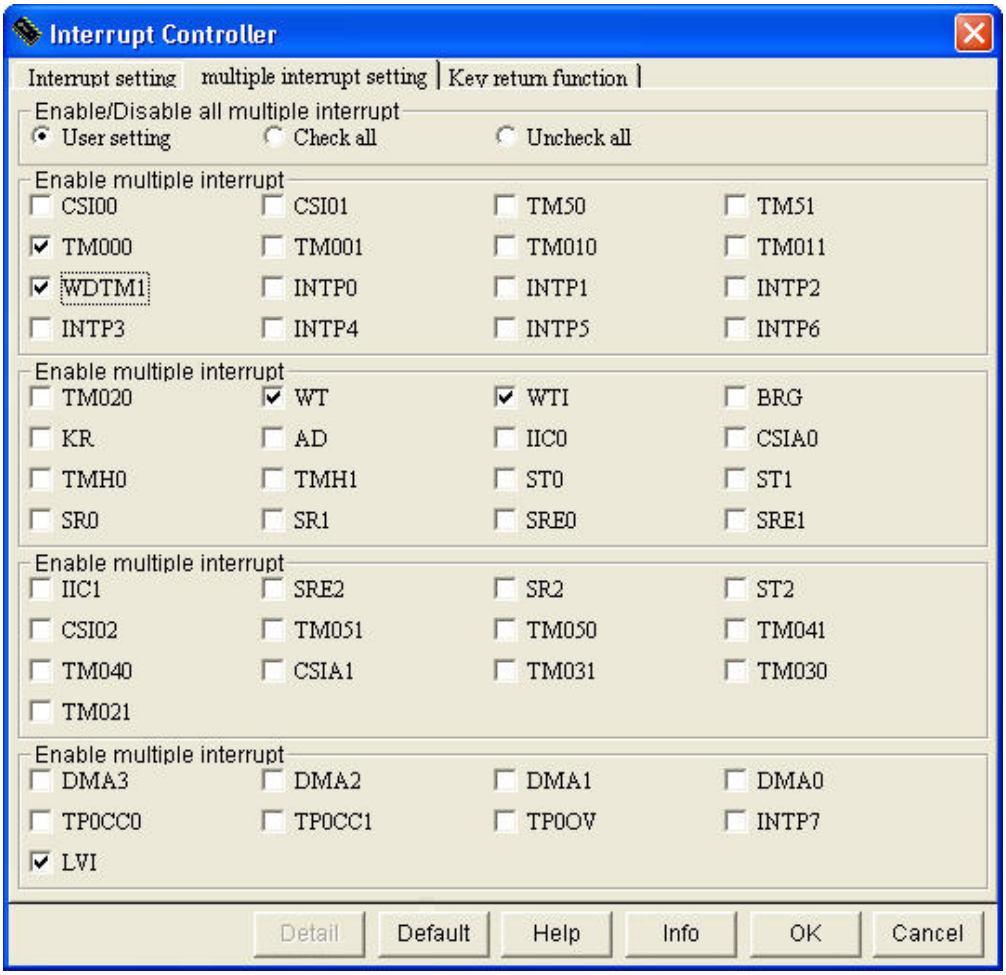


5.9 Interrupt Selections

Enable interrupts from the following sources:

- TM000: Timer 0
- WDTM1: Watch Dog Timer !
- WT: Watch Timer
- WTI: Watch Timer
- LVI: Low Voltage Detect

Figure 62. Interrupt Selections



5.10 Generation of Source files

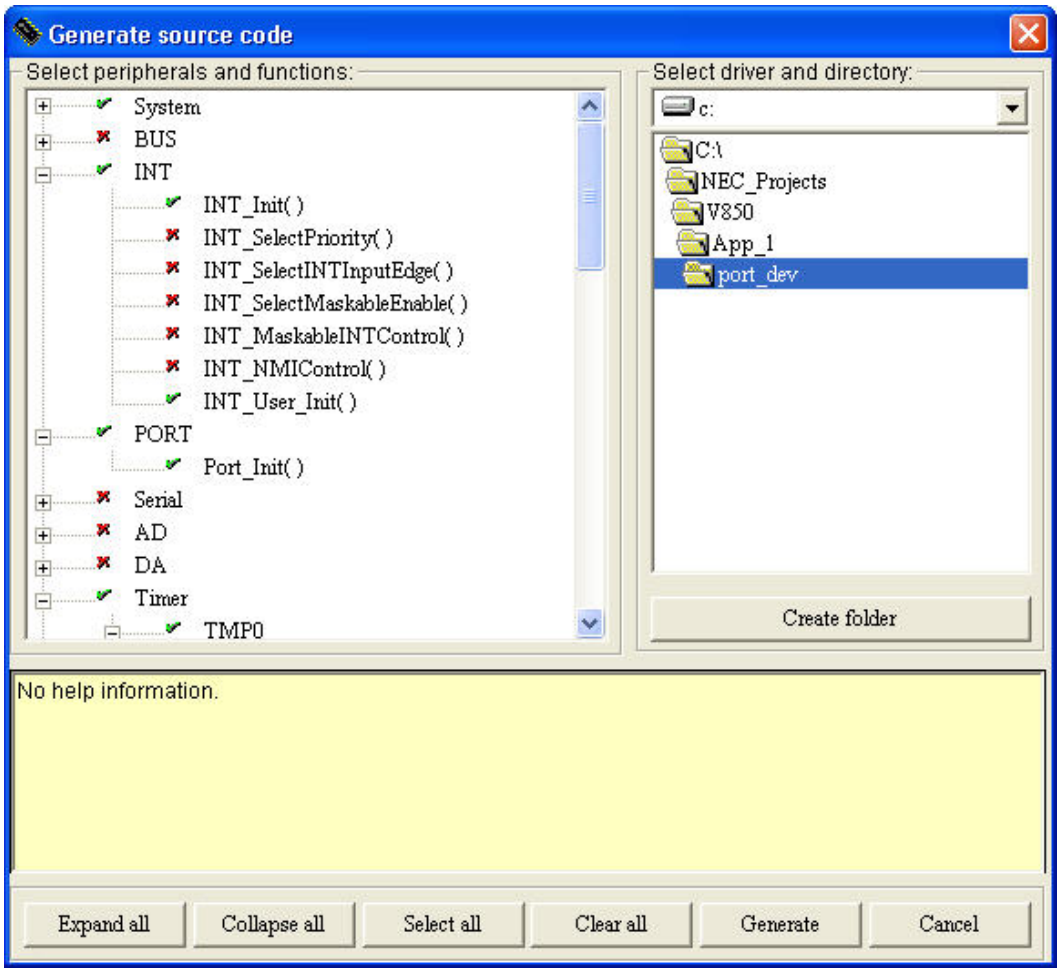
Once you have chosen all the subsystem selections, it is time to specify the code you want the Applilet to generate. If you do not make all the choices correctly, you can always come back, click a different selection, and regenerate the base code. Therefore, it is a good idea to save the files that are generated in a separate

directory and make a copy of them for use in development. Then you can easily generate changes and just do a “difference.”

5.10.1 Interrupt and Port Functions

Make the selections shown below.

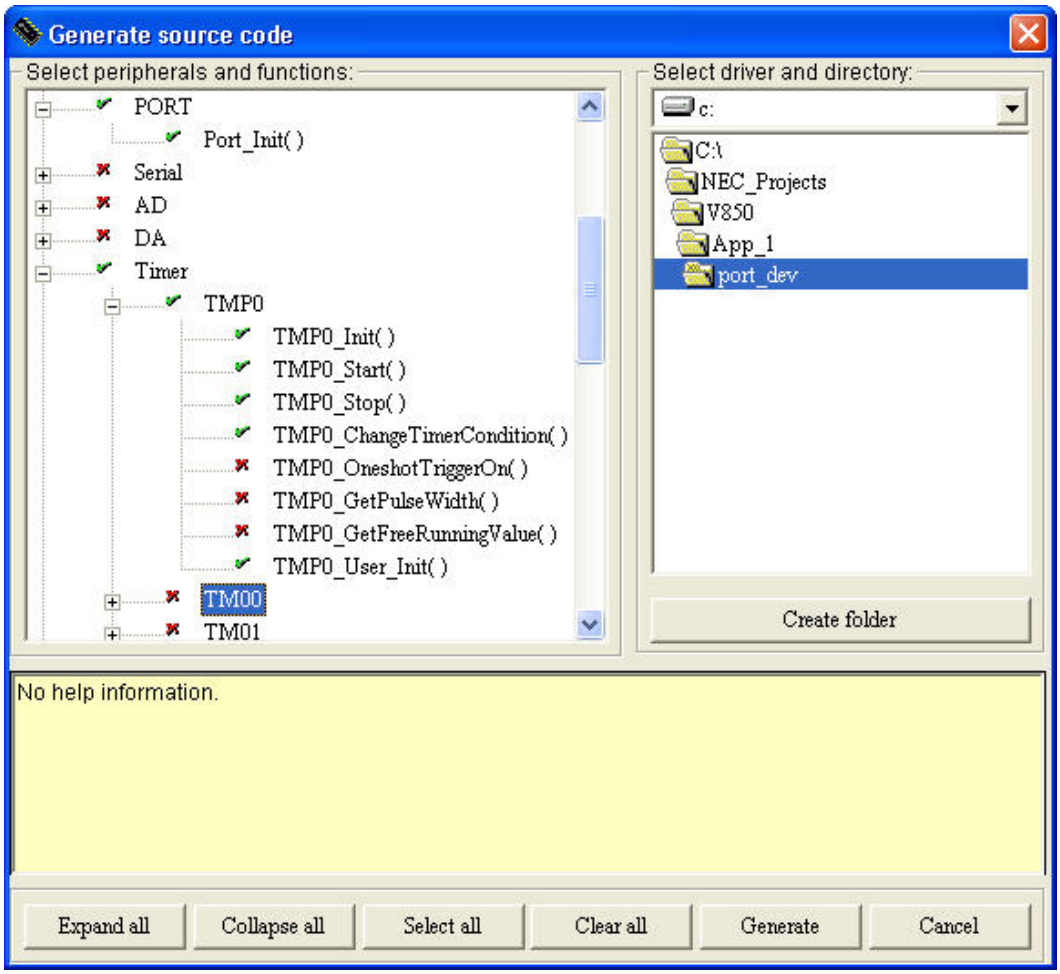
Figure 63. Interrupt and Port Functions



5.10.2 Timer Functions

Make the selections shown below.

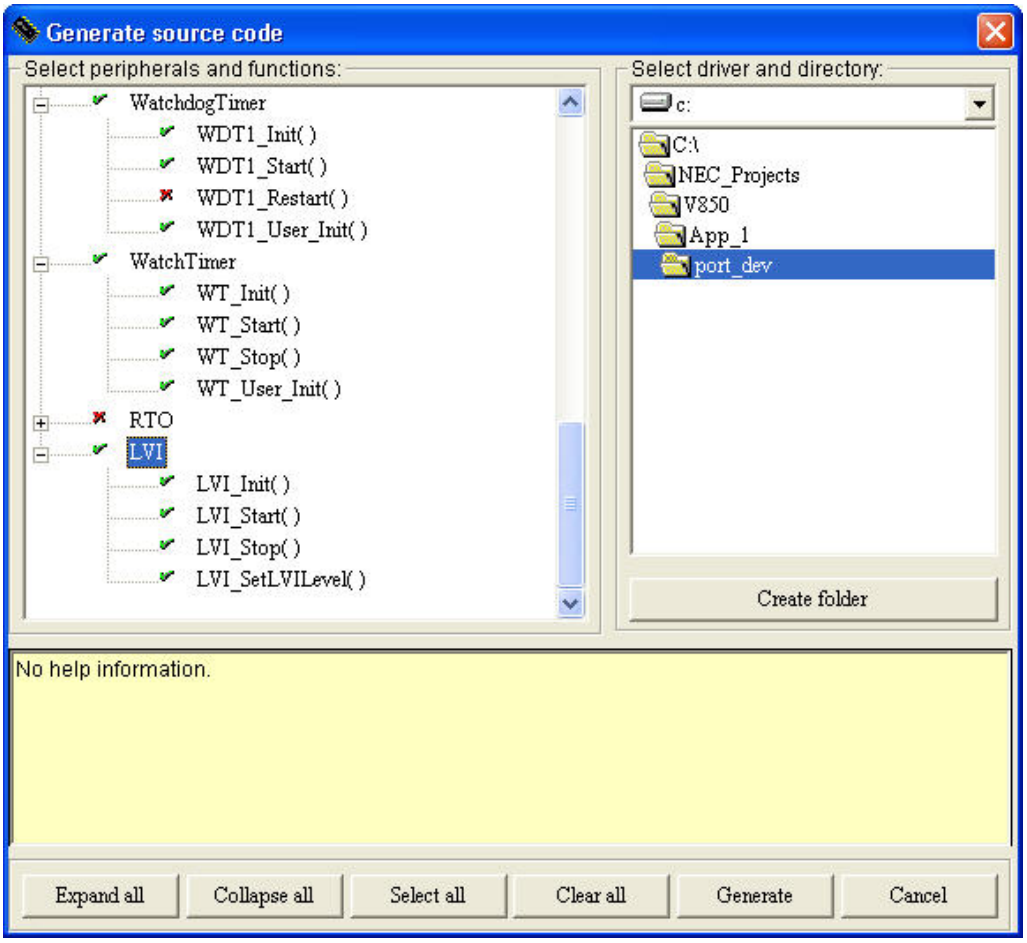
Figure 64. Timer Functions



5.10.3 Watchdog, Watch Timer and LVI Functions

Make the selections shown below.

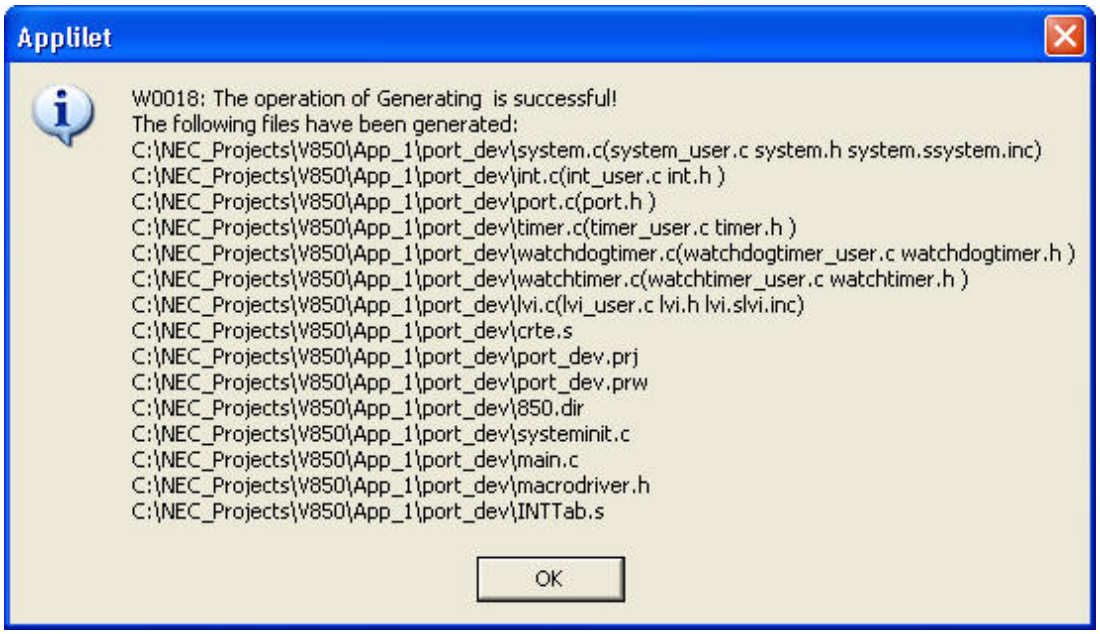
Figure 65. Watchdog, Watch Timer and LVI Functions



5.10.4 Source File List

The Applilet provides a list of files that it has generated, per your selections, as shown below.

Figure 66. Source File List



6. Appendix C – Source Code Listings

6.1 crtE.s

```
#  This device driver was created by Applilet for the V850ES/KX1+
#  32-Bit Single-Chip Microcontrollers
#
#  Copyright(C) NEC Electronics Corporation 2002-2004
#  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 : crte.s
#  Abstract : start file for CA850
#  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
#
#                 |        :        |
#                 |        :        |
#      tp -> -+-----------------+ __start     __tp_TEXT
#                 |  start up       |
#                 |---------------- |
#   text section  |                 |
#                 | user program    |
#                 |                 |
#                 |-----------------|
#                 | library         |
#                -+-----------------+
#                 |        :        |
#                 |        :        |
#                -+-----------------+ __argc
#                 |      0          |
#                 |---------------- | __argv
#   data section  |    #.L16        |
#                 |---------------- | .L16
#                 | 0x0,0x0,0x0,0x0 |
#                -+-----------------+
#                 |                 |
#   sdata section |                 |
#                 |                 |
#           gp-> -+-----------------+             __ssbss
#                 |                 |
#   sbss section  |                 |
#                 |                 |
#                 +-----------------+ __stack     __esbss     __sbss
#                 | stack area      |
#   bss section   |                 |
#                 |  0x200 bytes    |
#           sp-> -+-----------------+ __stack + STACKSIZE     __ebss
#
#===============================================================================

#-------------------------------------------------------------------------------
#   special symbols
```

```
#-------------------------------------------------------------------------------
    .extern __tp_TEXT, 4
    .extern __gp_DATA, 4
    .extern __ep_DATA, 4
    .extern __ssbss, 4
    .extern __esbss, 4
    .extern __sbss, 4
    .extern __ebss, 4


#-------------------------------------------------------------------------------
#    C program main function
#-------------------------------------------------------------------------------
    .extern _SystemInit
    .extern _main
    .extern _Clock_Init

#-------------------------------------------------------------------------------
#   for argv
#-------------------------------------------------------------------------------
    .data
    .size   __argc, 4
    .align  4
__argc:
    .word   0
    .size   __argv, 4
__argv:
    .word   #.L16
.L16:
    .byte   0
    .byte   0
    .byte   0
    .byte   0


#-------------------------------------------------------------------------------
#   dummy data declaration for creating sbss section
#-------------------------------------------------------------------------------
    .sbss
    .lcomm  __sbss_dummy, 0, 0


#-------------------------------------------------------------------------------
#   system stack
#-------------------------------------------------------------------------------
    .set    STACKSIZE, 0x200
    .bss
    .lcomm  __stack, STACKSIZE, 4


#-------------------------------------------------------------------------------
#   RESET handler
#-------------------------------------------------------------------------------

    .section   "RESET", text
    jr      __start
```

```
#------------------------------------------------------------------------------
#   start up
#       pointers:  tp - text pointer
#                  gp - global pointer
#                  sp - stack pointer
#                  ep - element pointer
#   exit status is set to r10
#------------------------------------------------------------------------------
    .text
    .align  4
    .globl  __start
    .globl  __exit
    .globl  __startend
    .extern  ___PROLOG_TABLE
__start:
    mov #__tp_TEXT, tp      -- set tp register
    mov #__gp_DATA, gp      -- set gp register offset
    add tp, gp          -- set gp register
    mov #__stack+STACKSIZE, sp  -- set sp register
    mov #__ep_DATA, ep      -- set ep register

    .option warning

    mov  1, r11     -- on-chip debug mode
    set1    5, PMCO[r0]
    set1    5, PO[r0]
    st.b     r11, PRCMD[r0]
    st.b     r11, OCDM[r0]

    nop
    nop
    nop
    nop
    nop
    mov 0x1, r11
    st.b r11, VSWC[r0]          --mainclock over 16.6MHz

    jarl    _Clock_Init, lp     -- call Clock_Init function

    mov #__ssbss, r13       -- clear sbss section
    mov #__esbss, r12
    cmp r12, r13
    jnl .L11
.L12:
    st.w    r0, [r13]
    add 4, r13
    cmp r12, r13
    jl  .L12
.L11:

    mov #__sbss, r13        -- clear bss section
    mov #__ebss, r12
    cmp r12, r13
    jnl .L14
.L15:
```

```
    st.w    r0, [r13]
    add 4, r13
    cmp r12, r13
    jl  .L15
.L14:

    mov     #___PROLOG_TABLE, r12   -- for prologue/epilogue runtime
    ldsr    r12, 20         -- set CTBP (CALLT base pointer)

    -- IRAM clean up --
    mov 0x3ffd800, r10  -- IRAM start address
    mov 0x3fff000, r11  -- IRAM end address
_clear_loop:            -- IRAM clean up
    st.w r0, 0x0[r10]
    add 4, r10
    cmp r11,r10
    jnz _clear_loop

    ld.w    $__argc, r6     -- set argc
    movea   $__argv, gp, r7     -- set argv
    jarl    _SystemInit, lp     -- call SystemInit function
    jarl    _main,lp        -- call main function
__exit:
    halt                -- end of program
```

6.2 systeminit.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+, V850ES/KG1+,

**  V850ES/KF1+, V850ES/KE1+ 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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 initiate
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
/*
** ************************************************************************
** Include files
** ************************************************************************
*/
#include "macrodriver.h"
#include "int.h"
#include "port.h"
#include "timer.h"
#include "watchtimer.h"
#include "watchdogtimer.h"
/*
** ************************************************************************
** MacroDefine
** ************************************************************************
*/
extern unsigned long _S_romp;
// RAM

/************************************************************************/
/* Function:    hdwinit()                                               */
/* Description: set up initial hardware                                 */
/*      note - this is called by startup code before main() is called*/
/* Input:       none                                                    */
/* Return:      none                                                    */
/************************************************************************/
void hdw_init( void )
{
```

```
unsigned char uc;

    /* not necessary to stop Watchdog timer 1, not running by default*/
    WDTM2 = 0x4e;   /* change timer settings, you cannot write to    */
                    /* this register again                           */

    PLLCTL = 0x03;                      // Enable PLL
    __asm("mov 0x3fff000,r10");         // Set SFR base in R10
    __asm("st.b r0,0x1fc[r10]");        // Write to PRCMD
    __asm("st.b r0,0x828[r10]");        // Set PCC for Fxx
    __asm("nop");
    __asm("nop");
    __asm("nop");
    __asm("nop");
    __asm("nop");
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  Init every Macro
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
void  SystemInit( void )
{
    __asm("di");                /* disable interrupt */
    _rcopy(&_S_romp, -1);       /* copy predefined values from rom to ram */
    hdw_init();                 /* some hardware initialization */
    PORT_Init( );               /* Port initiate    */
    TM0O_Init( );               /* TM0O initiate    */
    TMP0_Init( );               /* TMP0 initiate    */
    WT_Init( );                 /* WT initiate      */
    WDT1_Init( );               /* WDT1 initiate    */
    LVI_Init( );                /* lvi initiate     */
    __asm("ei");                /* enable interrupt */
}
```

6.3 main.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+, V850ES/KG1+,
**  V850ES/KF1+, V850ES/KE1+ 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
#pragma ioreg
/*
** ***********************************************************************
** Include files
** ***********************************************************************
*/
#include "macrodriver.h"
#include "int.h"
#include "port.h"
#include "timer.h"
#include "watchtimer.h"
#include "watchdogtimer.h"

/*
** ***********************************************************************
** MacroDefine
** ***********************************************************************
*/
#define TRUE          1
#define FALSE         0
#define MAX_TEST      7          /* maximum test number defined */

/* global variables */
int lvd_done = 0;                /* flag for low voltage detect interrupt */
int watch_tick, watch_rollover;  /* watch timer variables for interrupt handler */
unsigned char watch_count;       /* counter for display */
int test_number;
```

```
/* external prototypes */
extern void SystemInit( void );
extern void write_CLM(int mode);
extern void write_WRTM1(int mode);
extern void write_LVIM(int mode);

/* local prototypes */
void voltage_reset(unsigned char select);
void voltage_interrupt(unsigned char select);
void watchdog_timer_reset(void);
void watchdog_timer_interrupt(void);
void clock_monitor(void);
void lvd_monitor(void);
void wdt_monitor(void);

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**      main function
**              Saftey feature demonstration program
**  Parameters:
**      None
**
**  Returns:
**      None
**
**-------------------------------------------------------------------------
*/
void  main( void )
{
    int forever = 1;
    unsigned char swval;
    unsigned short interval[2];

    /* initialize watch timer */
    /* initialize interrupts  */
    /* initialize watchdog    */
    /* set up port for switch input */
    /* set up port for seven segment led output */
    /* clear LVIM, LVIS */
    SystemInit();
    TM00_Start();  /* watchdog 2 is running, get this timer going */

    /* read reason for reset from register RESF - chapter 24.3 */
    /* display the reason for starting up or resetting */
    display_reset(RESF);
    RESF = 0;           /* clear the reset reason register */
                        /* some reset reasons are retained */
                        /* unless cleared or Power-on-Clear occurs */
    test_number = 0;

    interval[0] = 0x7fff;
    interval[1] = 0xffff;
    TMP0_ChangeTimerCondition(interval,1); /* set interval compare value */
    TMP0_Start();
```

```
interval[0] = TM00_INTERVALVALUE;
// value has already been set, only call if change is needed
TM00_ChangeTimerCondition(interval,1);


/* loop forever waiting for operator input */
while(forever)
{
    swval = sw_get();   /* get a debounced switch value */
    switch (swval) {
    case SW_LU_RU:
        break;      /* do nothing if both switches are up */

    case SW_LD_RU:  /* left switch (SW2) is down */
        test_number++;
        if(test_number > MAX_TEST)
            test_number = 1;
        led_dig_left(test_number);
        led_out_right(LED_PAT_EQUAL);
        while(sw_get() == SW_LD_RU) {};  /* wait for switch to be released */
        break;

    case SW_LU_RD:  /* right switch (SW3) is down */
        while (sw_get() != SW_LU_RU)    /* wait for switches to both be up */
            ;
        switch(test_number)
        {
            case 1:
                voltage_reset(0x00);      // select 4.3 volts
                break;
            case 2:
                voltage_reset(0x03);      // select 3.7 volts
                break;
            case 3:
                voltage_interrupt(0x00);  // select 4.3 volts
                break;
            case 4:
                voltage_interrupt(0x03);  // select 3.7 volts
                break;
            case 5:
                watchdog_timer_reset();
                break;
            case 6:
                watchdog_timer_interrupt();
                break;
            case 7:
                clock_monitor();
                break;
            default:
                break;
        }  /* end switch */

        break;

    case SW_LD_RD:  /* both switches down   */
```

```
            break;      /* ignore at this level */
        } /* end switch (swval) */
    } /* end while(1) */
    return;  /* will never get here */
}

/*********************************************************************/
/* demonstration test #1 and #2                                      */
/* set up test for reset when low voltage detect of 4.3 or 3.7 volts  */
/* the low voltage detect is described in chapter 26                 */
/*                                                                   */
/* you must do a reset after this test because the threshold voltange */
/* cannot be changed after a low voltage reset has occurred          */
/*********************************************************************/
void voltage_reset(unsigned char select)
{
    lvd_done = 0;
    write_LVIM(0x00);  /* make sure it is off         */
    LVIIC= 0x40;       /* disable interrupt servicing LVIMK = 1 */

    LVIS = select;     /* select LIV for 4.3 volts detect */
    /*  enable low voltage detect,  set LIV to issue reset       */
    /* when supply voltage (VDD) is less than detect voltage 4.3 */
    write_LVIM(0x80);
    while(lvd_done == 0)
    {
        led_out_right(0xbf); /* display center bar, show we are waiting */
        /* wait 200 milliseconds */
        delay_ms(200);
        /* make sure LVIF bit is clear */
        if((LVIM & 1) == 0)
            lvd_done = 1;
    }

    lvd_done = 0;      /* so we stay in monitor routine until reset */
    write_LVIM(0x82);
    lvd_monitor();     /* wait for reset or operator change of test */
}

/*********************************************************************/
/* demonstration test #3 and #4                                      */
/* set int and test for interrupt when low voltage detect of 4.3 volts*/
/* run M-Station at 4.0 volts for this test                          */
/*********************************************************************/
void voltage_interrupt(unsigned char select)
{
    lvd_done = 0;
    write_LVIM(0x00);  /* make sure it is off */
    LVIIC = 0x40;      /* disable interrupt servicing LVIMK = 1 */

    LVIS = select;     /* set LIV for 4.3 volts detect */
    /* set LIV to issue interrupt, enable low voltage detect */
    /* when supply voltage (VDD) is less than detect voltage 4.3 */
    write_LVIM(0x80);

    while(lvd_done == 0)
```

```
    {
        led_out_right(0xbf); /* display center bar, show we are waiting */
        /* wait 200 milliseconds */
        delay_ms(200);
        /* make sure LVIF bit is clear */
        if((LVIM & 1) == 0)
            lvd_done = 1;
    }
    LVIIC = 0x40;   /* clear out previous interrupts */
    lvd_done = 0 ;  /* so we stay in monitor routine until interrupt */
    /* enable low voltage detect, set LIV to issue interrupt     */
    /* when supply voltage (VDD) is less than detect voltage 4.3 */
    LVIIC= 0x02;    /* enable, level 2 interrupt                 */
    lvd_monitor();  /* wait for interrupt or operator change of test */
}

/*********************************************************************/
/* demonstration test #5                                             */
/* Description: demonstrate watchdog timer 1 reset                   */
/*********************************************************************/
void watchdog_timer_reset(void)
{
    wdt_done = 0;

    TM00_User_Init(test_number, 1 );

    WDCS  = 0x05;        /* select 52.43 ms timeout        */
    WTIIC = 0x47;        /* disable interrupt              */

    write_WDTM1(0x98);   /* select reset, start it running */

    wdt_monitor();       /* wait for timer to expire or change of test */
}

/*********************************************************************/
/* Description: demonstrate watchdog timer 1 interrupt               */
/*********************************************************************/
void watchdog_timer_interrupt(void)
{
    wdt_done = 0;

    TM00_User_Init(test_number, 1 );

    WDCS  = 0x05;        /* select 52.43 ms timeout       */
    WTIIC = 0x47;        /* disable interrupt              */

    write_WDTM1(0x90);   /* select NMI, start it running  */
    /* page 675 interrupt control register  */
    WTIIC = 0x00;   /* enable interrupt, highest priority */
    wdt_monitor();  /* wait for timer to expire or change of test */
}

/*********************************************************************/
/* Description:  demonstrate clock monitor function                  */
/*      reset clock monitor is described in 24.4.6 of hardware user manual */
/*      enable of clock monitor is described in chapter 25           */
```

```
/*      test is performed by first selecting test 7                  */
/*      the LED's will start counting showing that we are waiting for */
/*      the clock monitor reset to happen.                           */
/*      remove a jumper from JP4 on the M-V850ES-KJ1 module          */
/*********************************************************************/
void clock_monitor(void)
{
    unsigned char swval;

    /* enable watch timer */
    watch_tick = 0;
    watch_count = 0;
    watch_rollover = 1;

    WT_Start();

    /* be sure ring oscillator is on, it is on by default */
    RCM = 0x00;

    /* enable clock monitor function operation */
    write_CLM(0x01);

    while(1)
    {
        swval = sw_get();
        if(swval == SW_LD_RU)
        {
            /* disable clock monitor function */

            while(sw_get() == SW_LD_RU){};  /* wait for release */
            led_dig_left(test_number);
            led_out_right(LED_PAT_EQUAL);
            return;
        }

        if(watch_rollover)
        {
            watch_rollover = 0;
            led_dig(watch_count); /* display the count */
        }
    }
}

/*********************************************************************/
/* Description: monitor for low voltage interrupt or reset          */
/*              not really monitoring for reset, it will just happen */
/*              when votage changes.                                 */
/*              also see if switch two has been pressed to select a  */
/*              different test                                       */
/*********************************************************************/
void lvd_monitor(void)
{
    unsigned char swval;

    watch_count = 0;       /* start counting fresh */
    watch_tick = 0;
```

```
    watch_rollover = 1;

    WT_Start();

    while(lvd_done == 0)
    {
        swval = sw_get();
        if(swval == SW_LD_RU)
        {
            /* turn off lvim, user selected a different test  */
            /* this cannot be done if the user selected reset */
            if((LVIM & 3) != 3)
            {
                write_LVIM(0x00);
                LVIIC = 00;
                while(sw_get() == SW_LD_RU){};  /* wait for release */
                led_dig_left(test_number);
                led_out_right(LED_PAT_EQUAL);
                return;
            }
            else
                led_clear();  /* may want to display err */
        }
        if(watch_rollover)
        {
            watch_rollover = 0;
            led_dig(watch_count); /* display the count */
        }

    }
    display_reset(LVI_INTERRUPT); /* display the LVI interrupt reason */
}

/*********************************************************************/
/* Description:  monitor for watch dog timer interrupt               */
/*               also see if switch two has been pressed to select   */
/*               a different test                                    */
/*********************************************************************/
void wdt_monitor(void)
{
    unsigned char swval;
    char timer_interval;

    timer_interval = 1;

    while(wdt_done == 0)
    {
        swval = sw_get();
        if(swval == SW_LD_RU)
        {
            /* turn off watchdog, user selected a different test */
            WDTM1 = 0x00;  /* stop counting */
            TM00_User_Init(0, 0);
            while(sw_get() == SW_LD_RU){};  /* wait for release */
            /* display test number */
            led_dig_left(test_number);
```

```
            led_out_right(LED_PAT_EQUAL);
            return;
        }
        if(swval == SW_LU_RD)
        {
            while(sw_get() == SW_LU_RD){};  /* wait for release */
            timer_interval++;
            TM00_User_Init(test_number, timer_interval);
        }
    }
    /* turn off watchdog, interrupt happened */
    WDTM1 = 0x00;  /* stop counting */

    TM00_User_Init(0,0);  /* stop periodic display */
    display_reset(WDT_INTERRUPT); /* display the wdt interrupt reason */
}
```

6.4 int.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KF1,
**  V850ES/KG1, V850ES/KJ1 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation  2002-2004
**  All rights reserved by NEC Electronics Corporation .
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation  assumes no responsibility for any losses incurred
**  by customers or third parties arising from the use of this file.
**
**  Filename : int.c
**  Abstract : This file implements a device driver for the INT module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
/*
** ********************************************************************
** Include files
** ********************************************************************
*/
#include "macrodriver.h"
#include "int.h"


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

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**  Init the external INT, including enable or disable,
**  priority setting
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-----------------------------------------------------------------------
*/
```

```
void  INT_Init( void )
{
    INT_User_Init ( );
    return;
}
```

6.5 int_user.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KF1,
**  V850ES/KG1, V850ES/KJ1 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation  2002-2004
**  All rights reserved by NEC Electronics Corporation .
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation  assumes no responsibility for any losses incurred
**  by customers or third parties arising from the use of this file.
**
**  Filename : int_user.c
**  Abstract : This file implements a device driver for the INT
**             interrupt service routine
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/

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

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


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

```
**-----------------------------------------------------------------------------
*/
void INT_User_Init( void )
{
    /* TODO. Add user code here */
}
```

6.6 LVI.s

```
--/*
--*************************************************************************
--**
--**  This device driver was created by Applilet for the V850ES/KX1+
--**  32-Bit Single-Chip Microcontrollers
--**
--**  Copyright(C) NEC Electronics Corporation 2002-2004
--**  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.s
--**  Abstract : This file implements a device driver for the LVI module
--**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
--**
--**  Device:  uPD70F3318Y
--**
--**  Compiler:  NEC/CA850
--**
--************************************************************************
--*/
    .globl  _LVI_Init
    .globl  _LVI_Start
    .align  4
--/*
--**-------------------------------------------------------------------------
--**
--**  Abstract:
--**       This function initialises the Low-Voltage Detector.
--**       Inital LVI function according to the configurator setting .
--**
--**  Parameters:
--**        None
--**
--**  Returns:
--**        None
--**
--**-------------------------------------------------------------------------------------------------------------------
-------
--*/
_LVI_Init:

    --push
    add -8, sp
    st.w    r11, 0[sp]
    st.w    r12, 4[sp]
    set1    6, LVIIC[r0]            --mask LVI interrupt
    mov 0x4, r11
    st.b    r11, LVIS[r0]           --compare with 3.5V
    st.b    r0, PRCMD[r0]
```

```
    set1    7, LVIM[r0]             --enable LVI operation
    nop
    nop
    nop
    nop
    nop

    --wait 100usec(TYP)
    movea   0x400, r0, r11
__LVI_LOOP1:
    nop
    nop
    nop

    addi    -1, r11, r11
    cmp r0, r11
    bnz __LVI_LOOP1

__LVI_LOOP2:
    -- Check LVIF
    tst1    0, LVIM[r0]
    bnz __LVI_LOOP2

    st.b    r0, PRCMD[r0]
    clr1    1, LVIM[r0]             --internal interrupt mode
    nop
    nop
    nop
    nop
    nop
    ld.b    LVIIC[r0], r12
    andi    0xf8, r12, r12
    st.b    r12, LVIIC[r0]      -- set priority highest
    clr1    6, LVIIC[r0]        -- unmask(enable) LVI interrupt
    --pop
    ld.w    0[sp], r11
    ld.w    4[sp], r12
    add 8, sp

    jmp [lp]

--/*
--**-------------------------------------------------------------------------
--**
--**  Abstract:
--**        This function starts the Low-Voltage Detector.
--**        (but it was already enabled in _LVI_init)
--**  Parameters:
--**    None
--**  Returns:
--**        MD_OK
--**    MD_ERROR    Illegal input parameter
--**
--**-------------------------------------------------------------------------
--*/
_LVI_Start:
```

```
clr1    6, LVIIC[r0]            -- enable LVI interrupt

st.b    r0, PRCMD[r0]
set1    7, LVIM[r0]             -- enable LVI operation
nop
nop
nop
nop
nop

jmp [lp]
```

6.7 LVI_user.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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 a device driver for the
**             LVI interrupt service routine
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
/*
************************************************************************
** Include files
************************************************************************
*/
#include "macrodriver.h"
#pragma interrupt   INTLVI  MD_INTLVI

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**  INTLVI Interrupt service routine
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-----------------------------------------------------------------------
*/
extern int lvd_done;
__multi_interrupt void MD_INTLVI( void )
{
    __asm("ei");
    lvd_done = 1;  /* indicate LVI interrupt received */
```

```
}
```

6.8 port.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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 a device driver for the PORT module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/

/*
**========================================================================
** Include files
**========================================================================
*/
#include "macrodriver.h"
#include "port.h"

/*
**========================================================================
** Constants
**========================================================================
*/
/*
**------------------------------------------------------------------------
**
**  Abstract:
**      Initialises the I/O module
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**------------------------------------------------------------------------
*/
```

```
void PORT_Init( void )
{
    /* initialize the port registers */
    P0 = PORT_P0;
    P1 = PORT_P1;
    P3 = PORT_P3;
    P4 = PORT_P4;
    P5 = PORT_P5;
    P6 = PORT_P6;
    P8 = PORT_P8;
    P9 = PORT_P9;
    PCD = PORT_PCD;
    PCM = PORT_PCM;
    PCS = PORT_PCS;
    PCT = PORT_PCT;
    PDH = PORT_PDH;
    PDL = PORT_PDL;

    /* initialize the function registers */
    PF3H = PORT_PF3;
    PF4 = PORT_PF4;
    PF5 = PORT_PF5;
    PF6 = PORT_PF6;
    PF8 = PORT_PF8;
    PF9H = PORT_PF9;

     /* initialize the Pull-up resistor option registers */
    PU0 = PORT_PU0;
    PU1 = PORT_PU1;
    PU3 = PORT_PU3;
    PU4 = PORT_PU4;
    PU5 = PORT_PU5;
    PU6 = PORT_PU6;
    PU8 = PORT_PU8;
    PU9 = PORT_PU9;
    PUCD = PORT_PUCD;
    PUCM = PORT_PUCM;
    PUCS = PORT_PUCS;
    PUCT = PORT_PUCT;
    PUDH = PORT_PUDH;
    PUDL = PORT_PUDL;

     /* initialize the mode registers */
    PM0 = PORT_PM0;
    PM1 = PORT_PM1;
    PM3 = PORT_PM3;
    PM4 = PORT_PM4;
    PM5 = PORT_PM5;
    PM6 = PORT_PM6;
    PM8 = PORT_PM8;
    PM9 = PORT_PM9;
    PMCD = PORT_PMCD;
    PMCM = PORT_PMCM;
    PMCS = PORT_PMCS;
    PMCT = PORT_PMCT;
    PMDH = PORT_PMDH;
```

```
    PMDL = PORT_PMDL;

     /*--- initialize the mode control registers ---*/
    PMC0 &= ~PORT_PMC0;
    PMC3 &= ~PORT_PMC3;
    PMC4 &= ~PORT_PMC4;
    PMC5 &= ~PORT_PMC5;
    PMC6 &= ~PORT_PMC6;
    PMC8 &= ~PORT_PMC8;
    PMC9 &= ~PORT_PMC9;
    PMCCM &= ~PORT_PMCCM;
    PMCCS &= ~PORT_PMCCS;
    PMCCT &= ~PORT_PMCCT;
    PMCDH &= ~PORT_PMCDH;
    PMCDL &= ~PORT_PMCDL;

    PORT_User();
}
```

6.9 port_user.c

```
/* functions making use of the I/O ports               */
/*                                                     */
/*  routines for LED display                           */
/*  for M-V850ES-KJ1 CPU board on M-Station base board */
/*      PDL8-PDL15 = output to right digit (LED2)      */
/*      PDH0-PDH7  = output to left digit  (LED1)      */
/*  For M-Station V2.1, assumes default SBs inserted   */
/*      SB27-SB33 inserted to connect                  */
/*       J1.27-J1.33 (PDL8-PDL15) to LED2 segments A-G */
/*      SB35-SB42 inserted to connect                  */
/*       J1.35-J1.42 (PDH0-PDH7) to LED1 segments A-DP */
/*  For M-Station V1.1, insert jumpers connecting:     */
/*       ROW1.25-32 (PDL8-PDL15) to ROW2.25-32 (LED2)  */
/*       ROW1.17-24 (PDH0-PDH7) to ROW2.17-20 (LED1)   */

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

#include "macrodriver.h"
#include "int.h"
#include "port.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 */
};

/***********************************************************************/
/* Function:    PORT_User()                                            */
/* Description: user initialization done after system initialization */
/*              of the ports.                                          */
/* Input:       none                                                   */
/* Return:      none                                                   */
/***********************************************************************/
void PORT_User (void)
{
```

```
    sw_init();              /* set up switches */
    sw_set_debounce(64);    /* initial debounce value */
    led_init();             /* initialize LED control ports */
}

/*********************************************************************/
/* Function:    led_init()                                           */
/* Description: set up ports for display of LED digits               */
/* Input:       none                                                 */
/* Return:      none                                                 */
/*********************************************************************/
void led_init(void)
{
    PMCDH = 0x00;       /* set port DH to port mode */
    PMDH = 0x00;        /* set port DH to output    */
    PDH  = 0xFF;        /* all LED's off            */

    PMCDLH = 0x00;      /* set port DL high 8-bits to port mode*/
    PMDLH = 0x00;       /* set port DL high 8-bits to output   */
    PDLH  = 0xFF;       /* all LED's off            */
}

/*********************************************************************/
/* Function:    led_out_right()                                      */
/* Description: output raw data to right LED                         */
/* Input:       unsigned char val - value to output to right segment */
/*              led port                                             */
/* Return:      none                                                 */
/*********************************************************************/
void led_out_right(unsigned char val)
{
    PDLH = val;
}

/*********************************************************************/
/* Function:    led_out_left()                                       */
/* Description: output raw data to left LED                          */
/* Input:       unsigned char val - value to output to left segment  */
/*              led port                                             */
/* Return:      none                                                 */
/*********************************************************************/
void led_out_left(unsigned char val)
{
    PDH = val;
}

/*********************************************************************/
/* Function:    led_dig()                                            */
/* Description: display input number as two hex digits               */
/* Input:   num - number to display                                  */
/*          bits 0-3 in right digit, in P7                           */
/*          bits 4-7 in left digit, in P0                            */
/* Return:  none                                                     */
/*********************************************************************/
void led_dig(unsigned char num)
{
```

```
    led_out_right(dig_tab[num & 0x0F]);         /* lower nibble*/
    led_out_left(dig_tab[(num >> 4) & 0x0F]);   /* upper nibble*/
}

/*********************************************************************/
/* Function: led_dig_bcd()                                           */
/* Description: display two digits of BCD coded bcdnum               */
/* Input:   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                           */
/* Return:  none                                                     */
/*********************************************************************/
void led_dig_bcd(unsigned char bcdnum)
{
unsigned char tens_dig;
    if (bcdnum > 99)
    {
        led_clear();    /* display both digits 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 += 0x10; /* while counting up the tens digit    */
    } while (bcdnum >= 10);
    /* now tens_dig has ten's place */
    /* and bcdnum has remainder     */
    led_dig(tens_dig+bcdnum);
}

/*********************************************************************/
/* Function:    led_dig_right()                                      */
/* Description: display number in right LED seven segment display    */
/* Input:       num  - number to display(0..0xf) in right digit      */
/* Return:      none                                                 */
/*********************************************************************/
void led_dig_right(unsigned char num)
{
    if (num > 0x0F)
        led_out_right(LED_PAT_BLANK);
    else
        led_out_right(dig_tab[num]);
}

/*********************************************************************/
/* Function:    led_dig_left()                                       */
```

```
/* Description: display number in left LED segment display          */
/* Input:       unsigned char num (0..0x0f) number to display       */
/* Return:      none                                                */
/**********************************************************************/
void led_dig_left(unsigned char num)
{
    if (num > 0x0F)
        led_out_left(LED_PAT_BLANK);
    else
        led_out_left(dig_tab[num]);
}

/**********************************************************************/
/* Function:    led_clear()                                           */
/* Description: blank the LED seven segment display                   */
/* Input:       none                                                  */
/* Return:      none                                                  */
/**********************************************************************/
void led_clear(void)
{
    led_out_right(LED_PAT_BLANK);   /* lower nibble*/
    led_out_left(LED_PAT_BLANK);    /* upper nibble*/
}

/**********************************************************************/
/* Description: display the cause of the reset based on RESF register */
/*   "oo" - manual reset or power-on-clear                            */
/*   "Lr" - low voltage reset                                         */
/*   "L1" - low voltage interrupt                                     */
/*   "dr" - watchdog reset                                            */
/*   "d1" - watchdog interrupt                                        */
/*   "cL' - clock monitor                                             */

/* MH_027  Seg_A2         aaaaa        MH_035  Seg_A1        */
/* MH_028  Seg B2        f     b       MH_036  Seg_B1        */
/* MH_029  Seg C2        f     b       MH_037  Seg_C1        */
/* MH_030  Seg D2         ggggg        MH_038  Seg_D1        */
/* MH_031  Seg E2        e     c       MH_039  Seg_E1        */
/* MH_032  Seg F2        e     c       MH_040  Seg_F1        */
/* MH_033  Seg G2         dddd         MH_041  Seg_G1        */
/* MH_034  Seg_dp2             o       MH_042  Seg_dp1       */
/* J1_B                                                      */
/* these go to P1_27 .. P1_42                                */
/* P1_27 - PDL08                       P1_35 - PDH0          */
/* P1_28 - PDL09                       P1_36 - PDH1          */
/* P1_29 - PDL10                       P1_37 - PDH2          */
/* P1_30 - PDL11                       P1_38 - PDH3          */
/* P1_31 - PDL12                       P1_39 - PDH4          */
/* P1_32 - PDL13                       P1_40 - PDH5          */
/* P1_33 - PDL14                       P1_41 - PDH6          */
/* P1_34 - PDL15                       P1_42 - PDH7          */
/**********************************************************************/

/**********************************************************************/
/* Function:    display_reset()                                       */
/* Description: display a symbol on the seven segment display which   */
```

```
/*              represents the reason for the reset                  */
/* Input:       int mode - index into the table of mode symbols      */
/*********************************************************************/
void display_reset(unsigned char reason)
{
    /* upper and lower segment to display for each reason          */
static unsigned short led_seg[] = {0xa3a3, /* reset and power on clear */
                                   0xc7af, /* LVI reset               */
                                   0xc7cf, /* LVI interrupt           */
                                   0xa1af, /* watchdog reset          */
                                   0xa1cf, /* watchdog interrupt      */
                                   0xa7c7}; /* clock monitor          */
int index;

    /* just write the bits to the led control port */
    if(reason & LVI_RESET)           /* LVIRF set */
        index = 1;
    else if (reason & CLM_RESET) /* CLMRF     */
        index = 5;
    else if (reason & LVI_INTERRUPT) /* low voltage interrupt */
        index = 2;
    else if (reason & WDT_INTERRUPT) /* watchdog interrupt */
        index = 4;
    else if (reason & (WDT1_RESET | WDT2_RESET)) /* WDT2RF or WDT1RF */
        index = 3;
    else
        index = 0;

    /* write the data to the LED ports */
    led_out_left( led_seg[index]>>8);
    led_out_right( led_seg[index]);
}

/*  routines for switch input                           */
/*  for M-V850ES-KJ1 CPU board on M-Station base board  */
/*      P94 = input for left switch  (SW2)              */
/*      P95 = input for right switch (SW3)              */
/*  For M-Station 2.1, assumes default SBs inserted     */
/*      SB7 connecting J1.7 (P94) to SW2                */
/*      SB8 connecting J1.7 (P95) to SW3                */
/*  For M-Station 1.1, insert jumpers connecting:       */
/*      ROW1.5 (P94) to ROW2.5 (SW2)                    */
/*      ROW1.6 (P95) to ROW2.6 (SW3)                    */


/* 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            */

/*********************************************************************/
/* Function:    sw_init()                                            */
/* Description: set up ports for user switch input                   */
/* Input:       none                                                 */
/* Return:      none                                                 */
```

```
/*********************************************************************/
void sw_init(void)
{
    /* set P94 and P95 to port mode */
    PMC9L &= 0xCF;
    /* set P94 and P95 to inputs    */
    PM9L |= 0x30;
    /* set pullups on P94 and P95   */
    PU9L |= 0x30;
    /* 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*/
}

/*********************************************************************/
/* Function:    sw_chk()                                             */
/* Description: return input from switches, undebounced              */
/* Input:       none                                                 */
/* Return:      unsigned char value of switches, not shifted         */
/*********************************************************************/
unsigned char sw_chk(void)
{
    return P9L & 0x30;
}

/*********************************************************************/
/* Function:    sw_set_debounce()                                    */
/* Description: set the debounce counter value                       */
/* Input:       unsigned char count - number of times to check the   */
/*              switch has not changed, before returning it          */
/* Return:      none                                                 */
/*********************************************************************/
void sw_set_debounce(unsigned char count)
{
    sw_deb_value = count;   /* set new debounce counter value*/
    sw_deb_count = count;   /* set counter to max*/
}

/*********************************************************************/
/* Function:    sw_get()                                             */
/* Description: return debounced switch input                        */
/*        This routine should be called periodically; on the nth call*/
/*        with a new switch value, will return the debounced value.  */
/*        Should be called often enough to rapidly poll switches;    */
/*        debounce count can be adjusted to filter out bounces.      */
/*        This method of debouncing requires no timers, returns quickly*/
/* Input:       none                                                 */
/* Return:      unsigned char - the current debounced switch values  */
/*********************************************************************/
unsigned char sw_get(void)
{
unsigned char val;

    val = sw_chk();     /* get current value*/
    /* if we have seen this before, just return it*/
```

```
    if (val == sw_last) {
        sw_new = sw_last;
        sw_deb_count = sw_deb_value; /* reset debounce counter to max*/
        return val;
    }

    /* 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 */
    if (val != sw_new) {
        sw_new = val;
        sw_deb_count = sw_deb_value;
        return sw_last;
    }

    /* 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 val;
    }

    /* if still debouncing, return the last value */
    return sw_last;
}
```

6.10 timer.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+, V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/

/*
** ************************************************************************
** Include files
** ************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"
/*
** ************************************************************************
**MacroDefine
** ************************************************************************
*/
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  Initiate TM00, select founction and input parameter
**  count clock selection, INT init
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
```

```
void TM00_Init( void )
{
    TMC00 = 0x0;                        /* stop TM00 */
    ClrIORBit(PRM00, 0x3);
    ClrIORBit(SELCNT1, 0x1);

    SELCNT1 |= ( TM00_Clock&0x4)>>2;    /* internal count clock */
    PRM00 |=( TM00_Clock&0x3);

    /* INTTM000 setting */
    TM0IC00 = Lowest;
    SetIORBit(TM0IC00, 0x40);
    /* TM00 interval */
    ClrIORBit(CRC00, 0x01);
    CR000 = TM00_INTERVALVALUE;
    CR001 = 0xffff;
    TM00_User_Init(0,0);               /* For user code */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  Initiate TMP0, select function and input parameter,
**  count clock selection, INT init.
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
void TMP0_Init( void )
{
                                        /* internal count clock */
    TP0CTL0 |= TM_TMP0_CLOCK;
    /* interrupt INTTP0CC0 */
    TP0CCIC0 = Lowest;
    SetIORBit(TP0CCIC0, 0x40);
    /* TMP0 interval */
    ClrIORBit(TP0CTL1, 0x7);
    TP0CCR0 = TM_TMP0_INTERVALVALUE;
    TP0CCR1 = 0xffff;
    TMP0_User_Init();                   /* For user code */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  start the TM00 counter
**
**  Parameters:
**  None
```

```
**
**  Returns:
**  None
**
**
**-------------------------------------------------------------------------
*/
void TM00_Start( void )
{
    TMC00 = 0x0c;               /* interval timer start */
    ClrIORBit(TM0IC00, 0x40);       /* enable INTTM000 */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  Start TMP0 counter.
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
void TMP0_Start( void )
{
    SetIORBit(TP0CTL0, 0x80);
    ClrIORBit(TP0CCIC0, 0x40);  /* enable interrupt INTTP0CC0 */
    return;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  stop the TM00 counter and clear the count register
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
void TM00_Stop( void )
{
    TMC00 = 0x0;                    /* stop TM00 */
    SetIORBit(TM0IC00, 0x40);       /* disable INTTM000 */
}

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

```
**
**  Abstract:
**  Stop the TMP0 counter and clear the count register.
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**--------------------------------------------------------------------------
*/
void TMP0_Stop( void )
{
    ClrIORBit(TP0CTL0, 0x80);
    SetIORBit(TP0CCIC0, 0x40);  /* disable interrupt INTTP0CC0 */
    return;
}

/*
**--------------------------------------------------------------------------
**
**  Abstract:
**  Change 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:
                  CR001=*(array_reg + 1);
         case 1:
                  CR000=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
      return MD_OK;
}
/*
**--------------------------------------------------------------------------
**
**  Abstract:
**  Change TMP0 condition.
**
**  Parameters:
**  USHORT*:     array_reg
**  USHORT:   array_num
```

```
**  Returns:
**  MD_OK
**  MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TMP0_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
   switch (array_num){
         case 2:
                  TP0CCR1=*(array_reg + 1);
         case 1:
                  TP0CCR0=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
      return MD_OK;
}
```

6.11 timer_user.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+, V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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
**             interrupt service routine
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
/*
************************************************************************
**Include files
************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"
#include "port.h"
#include "watchdogtimer.h"

#pragma interrupt INTTM000  MD_INTTM000
#pragma interrupt INTTP0CC0 MD_INTTP0CC0

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

volatile int timer_count;
int display_flip;
volatile char test_number_blink;
volatile char test_interval;
volatile char timer_tick;

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

```
**
**  Abstract:
**  This function is an empty function for user code when TM00 initializing
**
**  Parameters:
**       blink   - test number to blink, if zero, do not blink
**       interval- test interval number
**
**  Returns:
**  None
**
**--------------------------------------------------------------------------
*/
void TM00_User_Init(char blink, char interval )
{
    timer_count = 0;
    test_number_blink = blink & 0x0f;
    test_interval = interval & 0x0f;
    display_flip = 0;
}

/*
**--------------------------------------------------------------------------------
**
**  Abstract:
**  TM00 INTTM000 Interrupt service routine. the interrupt happens at 9.6 msec
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**--------------------------------------------------------------------------------
*/
__multi_interrupt void MD_INTTM000( void )
{
    __asm("ei");

    WDTE = 0xac;   /* reset watch dog timer 2 (used by clock monitor test) */
    timer_tick++;
    if(timer_tick >= test_interval)
    {
        timer_tick = 0;

        /* reset watch dog timer 1 */
        write_RUN1();

        /* increment count, do LED blink */
        if(test_number_blink)
        {
            timer_count++;
            /* if count is mod 4, display the test number and interval */
            if(timer_count > 4)
            {
                timer_count = 0;
```

```
                if(display_flip)
                {
                    led_dig_right(test_interval);
                    display_flip = 0;
                }
                else
                {
                    led_dig_right(LED_PAT_BLANK);
                    display_flip = 1;
                }
            }
        }
    }
}


volatile int TMP0_count;

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  This function is an empty function for user code when TMP0 initializing
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
/* do initialization for timer / event counter P */
void TMP0_User_Init(void)
{
    /* Add user code here */
    TMP0_count = 0;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  TMP0 INTTP0CC0 interrupt service routine.
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
__multi_interrupt void MD_INTTP0CC0( void )
{
    __asm("ei");
    TMP0_count++;
```

```
    /* used to show activity of the clock              */
    /* toggle the decimal point of the lower digit led */
    if(PDLH & 0x80)
        PDLH &= 0x7f;  // clear the decimal point
    else
        PDLH |= 0x80;  // set the decimal point
}
```

6.12 watchdogtimer.c

```
/*
*********************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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 a device driver for the WATCHDOGTIMER module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*********************************************************************************
*/
/*
** ***************************************************************************
** Include files
** ***************************************************************************
*/
#include "macrodriver.h"
#include "watchdogtimer.h"


/*
**-------------------------------------------------------------------------------
**
**  Abstract:
**  Watchdog timer 1 initiate in this function,
**  including mode selection clock selection, oscilliation stablization
**  selection.
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------------
*/
void WDT1_Init( void )
{
    wdt_done = 0;
```

```
    wdt_tick = 0;
    wdt_count = 0;
    wdt_rollover = 0;

     __asm(".set  _PRCMD, -0xe04");
     __asm(".set  _WDTM1, -0x93e");  /* 0xfffff6c2 */

    SetIORBit(WDT1IC, 0x40);    /* disable INTWDTM1 */
    WDCS = 0x07;            /* set watchdog timer 1  clock:2^21/fxw */

     /* watchdog timer mode 1:
        Upon overflow, non-maskable interrupt INTWDT1 is generated */
     __asm("movea 0x10, r0, r10");
     __asm("st.b  r10, _PRCMD[r0]");
     __asm("st.b  r10, _WDTM1[r0]");
     __asm("nop");
     __asm("nop");
     __asm("nop");
     __asm("nop");
     __asm("nop");
}
```

6.13 watchdogtimer_user.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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_user.c
**  Abstract : This file implements a device driver for the WATCHDOGTIMER
**             interrupt service routine
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
/*
** ************************************************************************
** Include files
** ************************************************************************
*/
#include "macrodriver.h"
#include "watchdogtimer.h"
#pragma interrupt INTWDT1 MD_INTWDT1 /* vector set */


extern volatile int wdt_done;

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  INTWDT1 interrupt service routine.
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTWDT1( void )
```

```
{
    __asm("ei");   /* enable interrupts */
    wdt_done = 1;  /* indicate watch dog timer interrupt received */
    wdt_done = 1;
}
```

6.14 watchtimer.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  All rights reserved by NEC Electronics Corporation
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses incurred
**  by customers or third parties arising from the use of this file.
**
**  Filename : watchtimer.c
**  Abstract : This file implements a device driver for the WATCHTIMER module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
/*
** ************************************************************************
** Include files
** ************************************************************************
*/
#include "macrodriver.h"
#include "watchtimer.h"
/*
** ************************************************************************
** MacroDefine
** ************************************************************************
*/

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  Initiate the watch timer, select its working mode, count clock,
**  enable/disable interrupt etc through Configurator.
**
**  Parameters:
**  None
**
**  Returns:
**  None

**
**-------------------------------------------------------------------------
```

```
*/
void WT_Init( void )
{
    PRSM = 0x00;  /* stop interval timer baud rate generator */
                  /* which is one source of watchtimer clock */
    /* disable watchtimer before change the setting */
    WTM.1 = 0;
    WTM.0 = 0;
    PRSM.4 = 0;
    /* disable all the interrupts of watchtimer */
    SetIORBit(WTIC, 0x40);  /* set interrupt mask to disable */
    SetIORBit(WTIIC, 0x40); /* set interrupt mask to disable */
    SetIORBit(BRGIC, 0x40); /* set interrupt mask to disable */
    SetIORBit(WTIC, 0x07);  /* select lowest interrupt priority */
    SetIORBit(WTM, 0x80);   /* select watchtimer clock:fw=fBRG */
    WTM &= 0xf3;        /* watchtimer flag setting:2^5/fw(977us) */
    WTM |= 0x08;        /* set prescale  WTM3 WTM2 */
    WTM &= 0x8f;        /* interval timer prescaler setting:2^4/fw */

    /* set the dedicated baud rate generator */
    PRSM |= WT_PRSM_BGCS;
    PRSCM = WT_PRSCM;   /* baud rate generator compare reg */
    PRSM.4 = 1;
    WT_User_Init();
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  Restart the watch timer after stopping. Enable the interrupt.
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
void  WT_Start( void )
{
    WTM.1 = 1;
    WTM.0 = 1;
    ClrIORBit(WTIC, 0x40);      /* enable INTWT */
    return;
}
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**  stop the watch timer.
**
**  Parameters:
**  None
**
```

```
**  Returns:
**  None
**
**--------------------------------------------------------------------------
*/
void WT_Stop( void )
{
    WTM &= 0xfc;                /* stop watch timer */
    SetIORBit(WTIC, 0x40);      /* disable INTWT    */
    return;
}
```

6.15 watchtimer_user.c

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/Kx1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  All rights reserved by NEC Electronics Corporation
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses incurred
**  by customers or third parties arising from the use of this file.
**
**  Filename : watchtimer_user.c
**  Abstract : This file implements a device driver for the watchtimer
**             interrupt service routine
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/
/*
** ********************************************************************
** Include files
** ********************************************************************
*/
#include "macrodriver.h"
#include "watchtimer.h"

#pragma interrupt INTWT MD_INTWT

#define TICKS_PER_MS    1 /* tbd */

extern volatile int watch_tick;
extern volatile int watch_rollover;
extern unsigned char watch_count;
/*
** ********************************************************************
** MacroDefine
** ********************************************************************
*/

#define TICKS_PER_MS    1 /* tbd */

volatile int end;
/*
**------------------------------------------------------------------------
**
```

```
**  Abstract:
**  This function is an empty function for user code when WT initializing
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**--------------------------------------------------------------------
*/
void WT_User_Init( void )
{
    watch_tick = 0;
    watch_count = 0;
    watch_rollover = 0;
}

/*********************************************************************/
/* Function:    delay_wt_ticks()                                     */
/* Description: delay for specified number of watch timer interrupts  */
/* Input:       int count - number of ticks to delay                 */
/* Return:      none                                                 */
/*********************************************************************/
void delay_wt_ticks(int count)
{
    int i;
    __asm("di");
    end = count;
    __asm("ei");

    while(end > 0){};  // wait for watch timer to count down to end
}

/*********************************************************************/
/* Function:    delay_ms()                                           */
/* Description: delay for specified number of counts, software only   */
/* Input:       int count - number of 1K iterations to delay         */
/* Return:      none                                                 */
/*********************************************************************/

void delay_ms(int count)
{
    int i;

    for(i=0; i<count*1000; i++)
    {
        __asm("nop");
    }
}

/*
**--------------------------------------------------------------------
**
**  Abstract:
**  INTWT interrupt service routine.
```

```
**
**  Parameters:
**  None
**
**  Returns:
**  None
**
**-------------------------------------------------------------------------
*/
__multi_interrupt void MD_INTWT( void )
{
    __asm("ei");
    /* user defined interrupt service routine */
    watch_tick++;
    end--;

    if(watch_tick >= 100)
    {
      watch_tick = 0;
      watch_count++;
      watch_rollover = 1;
    }
}
```

6.16 write_special.s

```
--/*
--************************************************************************
--**
--**  Filename : write_pcc.s85
--**  Abstract : This file implements a service routine to write the PCC
--**             (Processor Clock Control) register which is protected
--**
--**  Device:  uPD70F3318Y
--**
--**  Compiler:  NEC/C Compiler
--**
--************************************************************************
--*/

    .text       --RSEG CODE
    .globl   _write_PSC
    .globl   _write_PCC
    .globl   _write_CLM
    .globl   _write_WDTM1
    .globl   _write_LVIM
    .globl   _write_RUN1
    .align   4

#define        DCHC0           0xFFFFF0E0
#define        DCHC1           0xFFFFF0E2
#define        DCHC2           0xFFFFF0E4
#define        DCHC3           0xFFFFF0E6
#define        PRCMD           0xFFFFF1FC
#define        PSC             0xFFFFF1FE

#define        PSMR            0xFFFFF820
#define        PCC             0xFFFFF828
#define        CLM             0xFFFFF870


--/*
--**-------------------------------------------------------------------
--**
--** Abstract:
--**        This function sets the Power Save Control register
--**
--**  Parameters:
--**        R6 = value to set Power Save Control to
--**
--**  Returns:
--**         None
--**
--**-------------------------------------------------------------------
--*/
_write_PSC:
        st.b    r6, PRCMD[r0]  -- PRCMD register write
```

```
        st.b    r6, PSC[r0]    -- PSC register setting
        nop
        nop
        nop
        nop
        nop
        jmp [lp]

--/*
--**-------------------------------------------------------------------
--**
--**  Abstract:
--**        This function sets the Processor Clock Control register
--**
--**  Parameters:
--**        R6 = value to set clock control to
--**
--**  Returns:
--**         None
--**
--**-------------------------------------------------------------------
--*/
_write_PCC:
        st.b    r6, PRCMD[r0]  -- PRCMD register write
        st.b    r6, PCC[r0]    -- PCC register setting
        nop
        nop
        nop
        nop
        nop
        jmp [lp]

--/*
--**-------------------------------------------------------------------
--**
--**  Abstract:
--**        This function sets the Clock Monitor Control register
--**
--**  Parameters:
--**        R6 = value to set clock monitor control to
--**
--**  Returns:
--**         None
--**
--**-------------------------------------------------------------------
--*/
_write_CLM:
        nop
        st.b    r6, PRCMD[r0]  -- PRCMD register write
        st.b    r6, CLM[r0]    -- CLM register setting
        nop
        nop
        nop
        nop
        nop
        jmp [lp]
```

```
--/*
--**-------------------------------------------------------------------
--**
--**  Abstract:
--**        This function sets the Watchdog timer mode register 1
--**
--**  Parameters:
--**        R6 = value to set register to
--**
--**  Returns:
--**         None
--**
--**-------------------------------------------------------------------
--*/
_write_WDTM1:
        nop
        st.b    r6, PRCMD[r0]  -- PRCMD register write
        st.b    r6, WDTM1[r0]  -- WDTM1 register setting
        nop
        nop
        nop
        nop
        nop
        jmp [lp]

--/*
--**-------------------------------------------------------------------
--**
--**  Abstract:
--**        This function writes the Watchdog timer mode register 1
--**        RUN1 bit to reset the watchdog timer.
--**
--**  Parameters:
--**        none
--**
--**  Returns:
--**         None
--**-------------------------------------------------------------------
--*/
_write_RUN1:
        --push
        add -4, sp
        st.w    r11, 0[sp]
        ld.b    WDTM1[r0],r11  -- read watchdog mode register
        st.b    r6, PRCMD[r0]  -- PRCMD register write
        st.b    r11, WDTM1[r0] -- WDTM1 register run/reset bit
        -- pop
        ld.w    0[sp], r11
        add 4, sp

        jmp [lp]

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

```
--**  Abstract:
--**        This function sets the Low Voltage Detection register
--**
--**  Parameters:
--**        R6 = value to set register to
--**
--**  Returns:
--**         None
--**-------------------------------------------------------------------------
--*/
_write_LVIM:
        nop
        st.b    r6, PRCMD[r0]  -- PRCMD register write
        st.b    r6, LVIM[r0]   -- LVIM register setting
        nop
        nop
        nop
        nop
        nop
        jmp [lp]

--  END
```

6.17 int.h

```
/*
*********************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KF1,
**  V850ES/KG1, V850ES/KJ1 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation  2002-2004
**  All rights reserved by NEC Electronics Corporation .
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation  assumes no responsibility for any losses incurred
**  by customers or third parties arising from the use of this file.
**
**  Filename : int.h
**  Abstract : This file implements a device driver for the INT module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**
**  Compiler:  NEC/CA850
**
*********************************************************************************
*/
#ifndef _MDINT_
#define _MDINT_
/*
** ***************************************************************************
** MacroDefine
** ***************************************************************************
*/
#define IC_BASE     0xfffff110   /* interrupt control register base address */

#define LVI_INTERRUPT 0x08       /* bit indicating low voltage interrupt */
#define WDT_INTERRUPT 0x20       /* bit indicating watch dog timer interrupt */

enum ExternalINT { EX_NMI, EX_INTP0, EX_INTP1, EX_INTP2, EX_INTP3, EX_INTP4, EX_INTP5, EX_INTP6, EX_INTP7 };
enum MaskableSource{
        INT_WDT1, INT_INTP0, INT_INTP1, INT_INTP2, INT_INTP3, INT_INTP4, INT_INTP5, INT_INTP6,
        INT_TM000, INT_TM001, INT_TM010, INT_TM011, INT_TM50, INT_TM51, INT_CSI00, INT_CSI01,
        INT_SRE0, INT_SR0, INT_ST0, INT_SRE1, INT_SR1, INT_ST1, INT_TMH0, INT_TMH1,
        INT_CSIA0, INT_IIC0, INT_AD, INT_KR, INT_WTI, INT_WT, INT_BRG, INT_TM020,
        INT_TM021, INT_TM030, INT_TM031, INT_CSIA1, INT_TM040, INT_TM041, INT_TM050, INT_TM051,
        INT_CSI02, INT_SRE2, INT_SR2, INT_ST2, INT_IIC1,
        INT_LVI=48, INT_INTP7, INT_TP0OV,INT_TP0CC2, INT_TP0CC1, INT_DMA0, INT_DMA1, INT_DMA2,
        INT_DMA3
        };
void INT_Init( void );
void INT_User_Init ( void );

#endif
```

6.18 lvi.inc

```
--/*
--*****************************************************************************
--**
--**  This device driver was created by Applilet for the V850ES/KX1+
--**  32-Bit Single-Chip Microcontrollers
--**
--**  Copyright(C) NEC Electronics Corporation 2002-2004
--**  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.inc
--**  Abstract : This file implements a device driver for the LVI module
--**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
--**
--  Device:  uPD70F3318Y
--**
--**  Compiler:  NEC/CA850
--**
--*****************************************************************************
--*/
```

6.19 macrodriver.h

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+, V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/

#ifndef _MDSTATUS_
#define _MDSTATUS_
#pragma ioreg       /*enable use the register directly in ca850 compiler*/

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

#define MD_ON       1
#define MD_OFF      0

#define MD_TRUE         1
#define MD_FALSE        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_ADDRESSMATCH     MD_STATUSBASE+0x3   /*IIC slave address match*/
#define MD_OVF              MD_STATUSBASE+0x4   /*timer count overflow*/
#define MD_DMA_END          MD_STATUSBASE+0x5   /*DMA transfer end*/
#define MD_DMA_CONTINUE     MD_STATUSBASE+0x6   /*DMA transfer continue*/
#define MD_SPT              MD_STATUSBASE+0x7   /*IIC stop*/
#define MD_NACK             MD_STATUSBASE+0x8   /*IIC no ACK*/
#define MD_SLAVE_SEND_END   MD_STATUSBASE+0x9   /*IIC slave send end*/
```

```
#define MD_SLAVE_RCV_END    MD_STATUSBASE+0x0   /*IIC slave receive end*/
#define MD_MASTER_SEND_END  MD_STATUSBASE+0x11  /*IIC master send end*/
#define MD_MASTER_RCV_END   MD_STATUSBASE+0x12  /*IIC master receive end*/


/*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 n=0,1,2*/
#define MD_OVERRUNERROR     MD_ERRORBASE+0x3    /*UARTn overrun error n=0,1,2*/
#define MD_FRAMEERROR       MD_ERRORBASE+0x4    /*UARTn frame error n=0,1,2*/
#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_ODDBUF           MD_ERRORBASE+0x8    /*in 16bit transfer mode,buffer size should be even*/
#define MD_DATAEXISTS       MD_ERRORBASE+0x9    /*Data to be transferred next exists in TXBn register*/

/* macro fucntion definiton */
#define LockInt( ) { __asm("stsr 5,r10"); __asm("push r10"); __asm("di"); }
#define UnlockInt( ) { __asm("pop r10"); __asm("ldsr r10,5"); }

/*main clock and subclock as clock source*/
enum ClockMode { MainClock, SubClock };
void  Clock_Init( void );
/*clear IO register bit and set IO register bit */
#define ClrIORBit(Reg, ClrBitMap)   Reg &= ~ClrBitMap
#define SetIORBit(Reg, SetBitMap)   Reg |= SetBitMap

enum INTLevel{Highest,Level1,Level2,Level3,Level4,Level5,Level6,Lowest};
enum TrigEdge { None, RisingEdge,FallingEdge, BothEdge };


#define SYSTEMCLOCK 20000000
#define SUBCLOCK    32768
#define MAINCLOCK   5000000

#define LVI_RESET     0x01       /* bit indicating low voltage reset */
#define CLM_RESET     0x02       /* bit indicating clock monitor reset */
#define WDT2_RESET    0x10       /* bit indicating watch dog timer 2 reset */
#define WDT1_RESET    0x80       /* bit indicating watch dog timer 1 reset */

#endif
```

6.20 port.h

```
/*
*********************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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 a device driver for the PORT module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*********************************************************************************
*/

#ifndef _MDPORT_
#define _MDPORT_
/*
** *****************************************************************************
** MacroDefine
** *****************************************************************************
*/
#define PORT_PMC0   0x0
#define PORT_PM0    0xff
#define PORT_PU0    0x0
#define PORT_P0     0x0
#define PORT_PU1    0x0
#define PORT_PM1    0xff
#define PORT_P1     0x0
#define PORT_PMC3   0x0
#define PORT_PM3    0xffff
#define PORT_PU3    0x0
#define PORT_P3     0x0
#define PORT_PF3    0x0
#define PORT_PMC4   0x0
#define PORT_PM4    0xff
#define PORT_PU4    0x0
#define PORT_P4 0x0
#define PORT_PF4    0x0
#define PORT_PMC5   0x0
#define PORT_PM5    0xff
#define PORT_PU5    0x0
#define PORT_P5 0x0
```

```
#define PORT_PF5    0x0
#define PORT_PMC6   0x0
#define PORT_PM6    0xffff
#define PORT_PU6    0x0
#define PORT_P6 0x0
#define PORT_PF6    0x0
#define PORT_PMC8   0x0
#define PORT_PM8    0xff
#define PORT_PU8    0x0
#define PORT_P8 0x0
#define PORT_PF8    0x0
#define PORT_PMC9   0x0
#define PORT_PM9    0xffff
#define PORT_PU9    0x0
#define PORT_P9 0x0
#define PORT_PF9    0x0
#define PORT_PMCD   0xff
#define PORT_PCD    0x0
#define PORT_PMCM   0xff
#define PORT_PCM    0x0
#define PORT_PMCCM  0x0
#define PORT_PMCS   0xff
#define PORT_PCS    0x0
#define PORT_PMCCS  0x0
#define PORT_PMCT   0xff
#define PORT_PCT    0x0
#define PORT_PMCCT  0x0
#define PORT_PMDH   0x0
#define PORT_PDH    0x0
#define PORT_PMCDH  0xff
#define PORT_PMDL   0xff
#define PORT_PDL    0x0
#define PORT_PMCDL  0xff00
#define PORT_PUCD   0x0
#define PORT_PUCM   0x0
#define PORT_PUCS   0x0
#define PORT_PUCT   0x0
#define PORT_PUDH   0x0
#define PORT_PUDL   0x0

void PORT_Init( void );
void PORT_User( void );

/*  header for M-V850ES-KJ1 CPU board for LED digit display */

/*****************************************************************************/
/* 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_dig(unsigned char num);        /* display number as hex        */
extern void led_dig_bcd(unsigned char bcdnum); /* display number as BCD        */
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_out_right(unsigned char val);  /* output value to right LED    */
extern void led_out_left(unsigned char val);   /* output value to left LED     */

/*  header for M-V850ES-KJ1 CPU board for base board switch reading */

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

/* symbolic definitions for switch inputs */
/* SW2 = left switch = P94 */
/* SW3 = right switch = P95 */
/*                                              P95     P94  */
#define SW_LU_RU    0x30    /* left up, right up     1       1   */
#define SW_LD_RU    0x20    /* left down, right up   1       0   */
#define SW_LU_RD    0x10    /* left up, right down   0       1   */
#define SW_LD_RD    0x00    /* left down, right down 0       0   */

#define SW_DEF_DEB_COUNT    16  /* default debounce counter */
```

```
/*****************************************************************************/
/* Export functions                                                          */
/*****************************************************************************/
extern void PORT_User (void);
extern void led_init(void);
extern void led_out_right(unsigned char val);
extern void led_out_left(unsigned char val);
extern void led_dig(unsigned char num);
extern void led_dig_bcd(unsigned char bcdnum);
extern void led_dig_right(unsigned char num);
extern void led_dig_left(unsigned char num);
extern void led_clear(void);
extern void display_reset(unsigned char reason);
extern void sw_init(void);          /* init ports 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 debounce cound*/

#endif
```

6.21 timer.h

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KJ1+, V850ES/KG1+,
**  V850ES/KF1+ and V850ES/KE1+ 32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_

/*
** ************************************************************************
**MacroDefine
** ************************************************************************
*/
#define TM_TMP0_CLOCK   0x7
#define TM_TMP0_INTERVALVALUE   0x3d08
#define TM_TMP0_INTERVALVALUE2  0x1e83
#define TM_TMP0_ONESHOTOUTPUTCYCLE  0x3d08
#define TM_TMP0_ONESHOTOUTPUTDELAY  0x1e83
#define TM_TMP0_EXTTRIGGERCYCLE 0x3d08
#define TM_TMP0_EXTTRIGGERDELAY 0x1e83
#define TM_TMP0_PWMCYCLE    0x3d08
#define TM_TMP0_PWMWIDTH    0x1e83
#define TM_TMP0_CCROCOMPARE 0x3d08
#define TM_TMP0_CCR1COMPARE 0x1e83
#define TM00_Clock  0x5
#define TM00_INTERVALVALUE  0xbb7
#define TM00_SQUAREWIDTH    0xbb7
#define TM00_PPGCYCLE   0xbb7
#define TM00_PPGWIDTH   0x00
#define TM00_ONESHOTCYCLE   0xbb7
#define TM00_ONEPULSEDELAY  0x00
#define TM01_Clock  0x0
#define TM01_INTERVALVALUE  0x00
```

```
#define TM01_SQUAREWIDTH    0x00
#define TM01_PPGCYCLE   0x00
#define TM01_PPGWIDTH   0x00
#define TM01_ONESHOTCYCLE   0x00
#define TM01_ONEPULSEDELAY  0x00
#define TM02_Clock  0x0
#define TM02_INTERVALVALUE  0x00
#define TM02_SQUAREWIDTH    0x00
#define TM02_PPGCYCLE   0x00
#define TM02_PPGWIDTH   0x00
#define TM02_ONESHOTCYCLE   0x00
#define TM02_ONEPULSEDELAY  0x00
#define TM03_Clock  0x0
#define TM03_INTERVALVALUE  0x00
#define TM03_SQUAREWIDTH    0x00
#define TM03_PPGCYCLE   0x00
#define TM03_PPGWIDTH   0x00
#define TM03_ONESHOTCYCLE   0x00
#define TM03_ONEPULSEDELAY  0x00
#define TM04_Clock  0x0
#define TM04_INTERVALVALUE  0x00
#define TM04_SQUAREWIDTH    0x00
#define TM04_PPGCYCLE   0x00
#define TM04_PPGWIDTH   0x00
#define TM04_ONESHOTCYCLE   0x00
#define TM04_ONEPULSEDELAY  0x00
#define TM05_Clock  0x0
#define TM05_INTERVALVALUE  0x00
#define TM05_SQUAREWIDTH    0x00
#define TM05_PPGCYCLE   0x00
#define TM05_PPGWIDTH   0x00
#define TM05_ONESHOTCYCLE   0x00
#define TM05_ONEPULSEDELAY  0x00
#define TM50_Clock  0x5
#define TM50_INTERVALVALUE  0x1e
#define TM50_SQUAREWIDTH    0x1e
#define TM50_PWMACTIVEVALUE 0x1e
#define TM51_Clock  0x5
#define TM51_INTERVALVALUE  0x1e
#define TM51_SQUAREWIDTH    0x1e
#define TM51_PWMACTIVEVALUE 0x1e
#define TMH0_Clock  0x3
#define TMH0_INTERVALVALUE  0x7c
#define TMH0_SQUAREWIDTH    0x7c
#define TMH0_PWMCYCLE   0x7c
#define TMH0_PWMDELAY   0x3d
#define TMH0_CARRIERDELAY   0x7c
#define TMH0_CARRIERWIDTH   0x3d
#define TMH1_Clock  0x4
#define TMH1_INTERVALVALUE  0xf9
#define TMH1_SQUAREWIDTH    0xf9
#define TMH1_PWMCYCLE   0xf9
#define TMH1_PWMDELAY   0xe
#define TMH1_CARRIERDELAY   0xf9
#define TMH1_CARRIERWIDTH   0xe
```

```
void TMP0_Init( void );
void TMP0_Start( void );
void TMP0_Stop( void );
MD_STATUS TMP0_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);

/*timer00 configurator initiation*/
void TM00_Init( void );

/*timer00 to 05 free running start,50,51,H0,H1 timer start*/
void TM00_Start( void );

/*timer00 to 05,50,51,H0,H1 timer stop*/
void TM00_Stop( void );
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
__multi_interrupt void MD_INTTM000( void );
__interrupt void MD_INTTMH1( void );
void TM00_User_Init(char test_number, char interval );

#endif
```

6.22 watchdogtimer.h

```
/*
*********************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  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 a device driver for the WATCHDOGTIMER module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
**  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*********************************************************************************
*/
#ifndef _MDWTACHDOGTIMER_
#define _MDWTACHDOGTIMER_
/*
** ***************************************************************************
**MacroDefine
** ***************************************************************************
*/
volatile int wdt_done;          /* flag for watch dog timer interrupt */
unsigned char wdt_count;        /* counter for display */
volatile int wdt_tick;          /* interrupt counter   */
volatile int  wdt_rollover;     /* flag indicating count has reached some limit */
/* external prototypes */
extern void write_RUN1(void);   /* the watch dog timer reset in write_special.s */

void  WDT1_Init( void );
__interrupt void MD_INTWDT1( void );
#endif
```

6.23 watchtimer.h

```
/*
*********************************************************************************
**
**  This device driver was created by Applilet for the V850ES/KX1+
**  32-Bit Single-Chip Microcontrollers
**
**  Copyright(C) NEC Electronics Corporation 2002-2004
**  All rights reserved by NEC Electronics Corporation
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses incurred
**  by customers or third parties arising from the use of this file.
**
**  Filename : watchtimer.h
**  Abstract : This file implements a device driver for the WATCHTIMER module
**  APIlib:  V850ESKX1H.lib V1.33 [24 Sep 2004]
**
  Device:  uPD70F3318Y
**
**  Compiler:  NEC/CA850
**
*********************************************************************************
*/
#ifndef _MDWATCHTIMER_
#define _MDWATCHTIMER_
/*
** ***************************************************************************
**MacroDefine
** ***************************************************************************
*/
#define WT_PRSM_BGCS    0x01
#define WT_PRSCM        0x98

extern void delay_wt_ticks(int count);

void WT_Init( void );
void WT_Start( void );
void WT_Stop( void );
void delay_ms(int count);
__multi_interrupt void MD_INTWT( void );
#endif
```