To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

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

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

RENESAS

Application Note

Using 8- and 16-Bit Timers

Timer/Counter Techniques for NEC Electronics Microcontrollers

Document no. U18269EU1V0AN00

The information in this document is current as of July 2006. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.

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

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

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

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

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

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

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

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

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

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

Notes:

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

M8E 02.10

Revision History

Date	Revision	Section	Description
July 2006	—	—	First release

Contents

1. Introduction

This application note gives simple examples that show a variety of uses for the counter/timer peripherals in NEC Electronics microcontrollers. The purpose is to help you better understand the use of these peripherals and provide basic routines you can use in more complex applications.

Each section of this application note describes a different counter/timer technique and includes the following information:

- Description of peripheral features
- Example program descriptions and specifications
- Software flow charts
- Applilet reference drivers
- Description of the 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 your code.

For details on using the Applilet and NEC Electronics microcontrollers, please consult the appropriate User's Manuals and related documents.

1.1 Overview of 8- and 16-Bit Timers

Most NEC Electronics microcontrollers incorporate 8- and/or 16-bit timers, and many devices offer multiple timers. The timers provide basic measurements or counts for aiding decisions involved in many control functions. Most timers have multiple functions, and you can configure them to meet many different requirements. In addition, all the timers have a flexible clock-selection scheme, which allows you to specify different clock sources and clock divisions.

A typical timer in an NEC Electronics microcontroller offers the following features:

- Interval timer
- External event counter
- Square-wave generator
- One-shot generator
- Programmable pulse generator (PPG)

- Programmable pulse-width modulation (PWM)
- Pulse-width measurement

To configure timer operations and output modes, you use the microcontroller's operation control registers. A clock selection register selects the base clock for the timer operation. You can select from several types of clock sources, including the main clock, divisions of the main clock, the subclock, and external clock inputs. The timers can operate in free-running mode, with repeated output changes or interrupts, or can be started and stopped for timing of single intervals or pulses.

The interval timer generates an interrupt signal after a preset time interval. The timer generates the interrupt when the counter value matches the preset compare register value. you can use this feature for checking status or performing tasks at preset intervals.

In External Event Counter mode, the event counter receives signals of external events. The event counter value is compared with a preset event monitor value. An interrupt is generated when the event count value and event monitor value match. You can use this feature to monitor external events and perform tasks when a preset number of external events is detected.

You can program free-running timers to create output pulses of varying shapes and durations. To generate a square wave, for example, create equal time periods for high and low outputs. For PWM, you set a particular frequency for the output on/off cycle and vary the relative duty cycle of the on and off times.

You can use programmable pulse generation to make custom pulses, varying both the frequency of the pulse and the duty cycle of the high/low time.

You can use one-shot pulse generation to create a custom single pulse of variable width and delay for start time, based on either an external trigger or software trigger. In some selected NEC Electronics microcontrollers, a combination of timers can generate carrier-pulse waveforms for remote control applications.

The timers in NEC Electronics 78K0-family microcontrollers include the functional blocks shown in Figure 1. This figure provides a generic overview of typical timer elements. Specific timers may have only some of these functional blocks or may have additional specialized blocks for functions such as carrier generation.

Figure 1. Typical Timer Functional Blocks

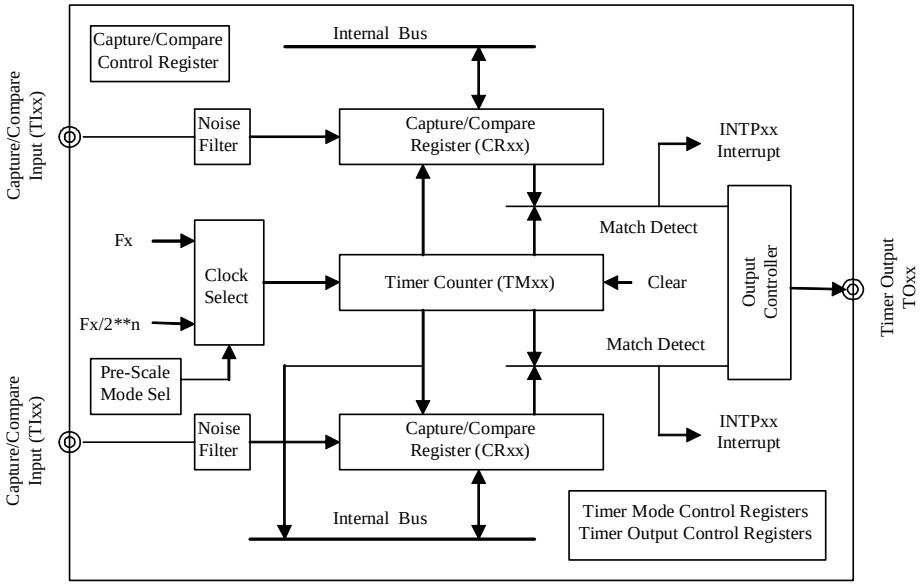


Table 1. Operation Registers

Register	Name	Description of Functions
Timer/Counter	TMxx	Timer/counter is incremented in synchronization with count clock
		Timer/counter is read-only register
Capture/Compare Register	CRxx/CRyy	These registers can be used as capture or compare register
	Compare	The value of register is constantly compared with timer/counter
		When match occurs, an interrupt is generated
	Capture	The timer/counter value is captured into capture/compare register
		TIxx, TIyy are used as capture-trigger input
Capture-Trigger Input	TIxx/TIyy	External input to capture timer/counter value into capture register
		When captured, input clears timer/counter (depending on mode setting)
		Timer value is captured on rising, falling or both edges of trigger

Table 2. Control Registers

Register	Name	Description of Functions
Mode Control Register	TMCxx	Enable/disable timer/counter
		Mode of operation - free-running, clear-and-start mode
		Timer/counter overflow flag
Capture/Compare Control Register	CRCxx	Select compare operation or capture operation
		Capture-trigger select
Output Control Register	TOCxx	Enable/disable output, inversion of output, initial value of output
Prescaler Mode Control Register	PRMxx	Clock select, capture-trigger valid edge select
Port mode Register	PMx	Input or output mode select for TIxx inputs and TOxx outputs

2. Interval Timer Using 16-Bit Timer 00 (TM00)

Most of the timers/counters in NEC Electronics 78K0 family support interval timer functions. You can use these functions to measure time intervals and take action after an elapsed time.

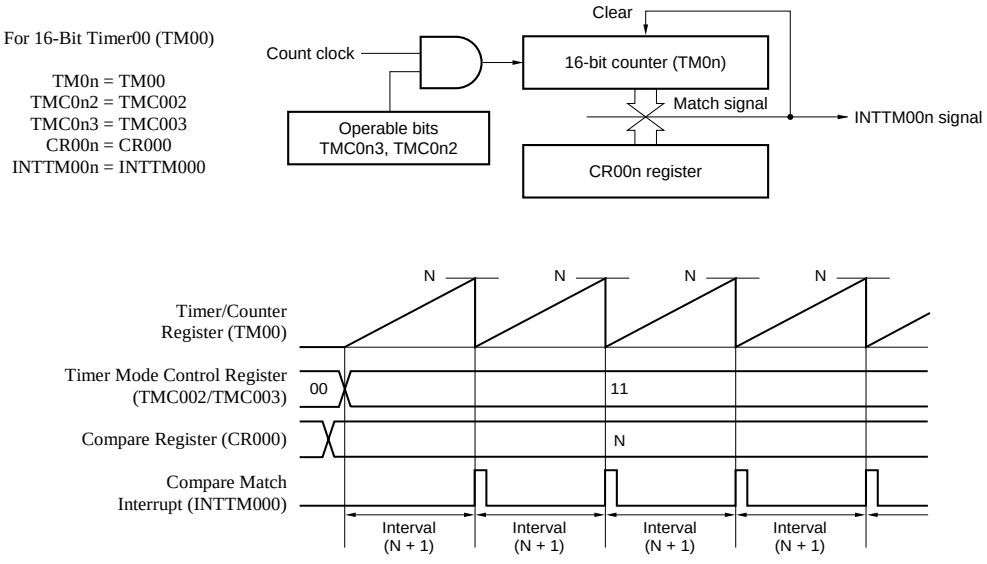
2.1 16-Bit Interval Timer Features

In interval timer mode, the 16-bit timers offer the following features:

- Variable time base based on divisions of the peripheral clock, using the PRM00 register
- Variable interval time using CR000 to count time-base clocks
- Optional interrupt-on-comparison between TM00 and CR000

To use the interval timer feature, begin by setting the timer/counter mode control register to clear-and-start mode. In this mode, a match between the timer/counter register TM00 and the compare register CR000 clears the timer/counter and starts the timer/counter counting in synchronization with the count clock. When timer/counter register TM00 matches compare register CR000, the timer/counter clears to zero and generates a match interrupt.

Figure 2. Characteristics of Timer 00 and Timer 01 (with Generic Register Names)



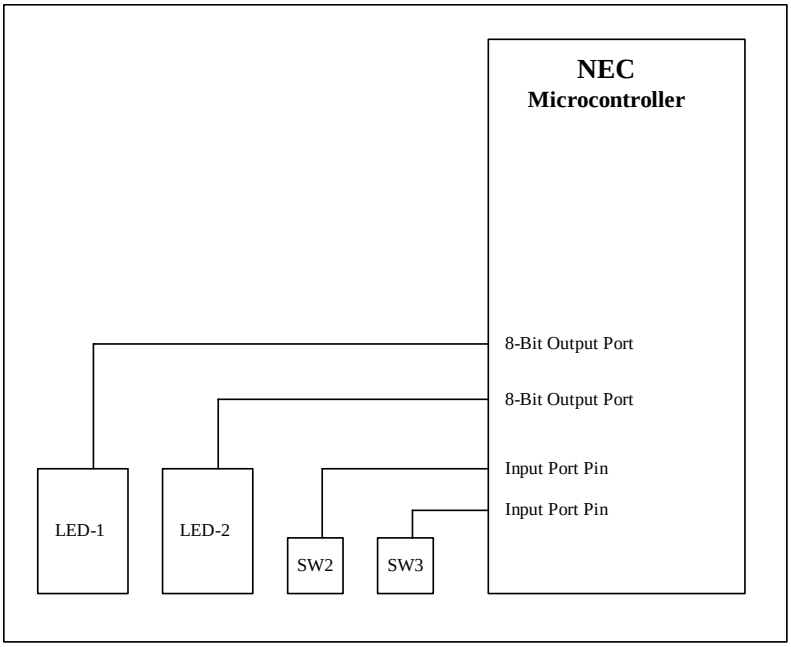
To configure the timer for interval timer operations, use the following steps:

- Set the time base using PRM00, and set the interval duration in CR000
- Set the priority for the interrupt, clear the interrupt flag, and unmask the interrupt, if used
- Enable the timer by setting the Timer Mode Register TMC00 for clear-and-start mode on compare of TM00 and CR000

2.2 Program Description and Specification

The example program demonstrates the use of an interval timer to eliminate noise on a switch input. The demonstration uses two switches to show methods for decrementing and incrementing counter values. When the first switch is pressed, it decrements a counter value and, after switch debouncing, displays the number of times the switch has been pressed on the LEDs. When the second switch is pressed, it increments the count value and, after switch debouncing, displays the number on the LEDs.

Figure 3. Switch Debounce Demonstration Setup



The MCU accomplishes switch debouncing by sampling the switch input at a preset time interval after the switch is pressed. The sampling period is programmed using interval timer feature. If you configure the timer/counter in clear-and-start mode and load the desired value into the compare register, every time the timer register and compare register match, the timer generates an interrupt. The timer register then clears and starts counting in synchronization with the selected clock.

At the preset interval, when the timer register and the compare register match, the MCU samples the state of the switch. If the switch is pressed for a preset number of cycles and the state does not change during this time, the program recognizes this state as "switch pressed." You can program the number of cycles to sample. If, however, the switch state changes during the sampling period, the change is considered as switch noise and will not be recognized as a switch input.

For the demonstration, the switches connect to port pins of a 78K0 microcontroller. The LEDs connect to 8-bit ports. The demonstration sequence is:

- Initialization Set timer/counter mode, compare register value, and set port mode

- Press SW2 Display number of times SW2 has been pressed on the LEDs (decrement)
- Press SW3 Display number of times SW3 has been pressed on the LEDs (increment)

Specifications:

- Timer 00 is set to give a periodic interrupt every 1 millisecond
- Switches must be seen stable for 10 consecutive inputs (10 milliseconds) to be debounced

2.3 Software Flow Charts

The demonstration program consists of the following major sections:

- Initialization code for the program, called before the main() program starts
- The main program loop, which checks the status of switches and displays the count on the LEDs
- Subroutines generated by the Applilet to handle Timer 00 starting, stopping and changing interval
- Subroutines with user code for handling Timer 00 interrupts (Applilet-generated stub interrupt service routines, with user code added)
- Subroutines for pushbutton input, LED display output

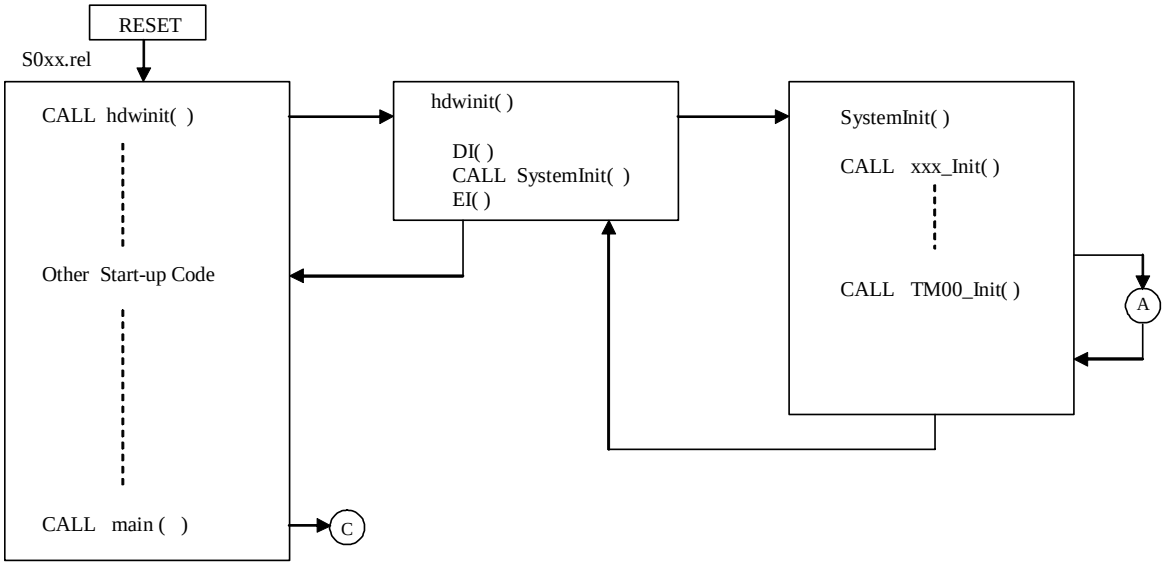
Flowcharts presented below describe initialization, the main program, and Timer 00-related subroutines and interrupt service routines.

2.3.1 Program Startup and Initialization

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

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

Figure 4. Flow Chart - Program Startup

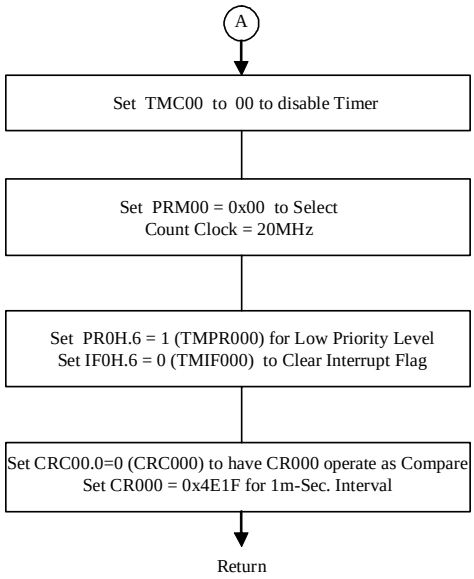


After the hdwinit( ) function finishes, the startup code calls the main( ) function of the user program. So at the start of the main( ) function, all peripheral initialization has been done. The main( ) function does not need to call the individual peripheral initialization routines.

2.3.2 TM00_Init( ) - Timer 00 Initialization

The TM00_Init() routine, called by SystemInit(), sets the Timer 00 registers to prepare for interval timer operation. First the routine disables the timer, which is recommended when changing register settings.

Figure 5. Initializing Timer 00



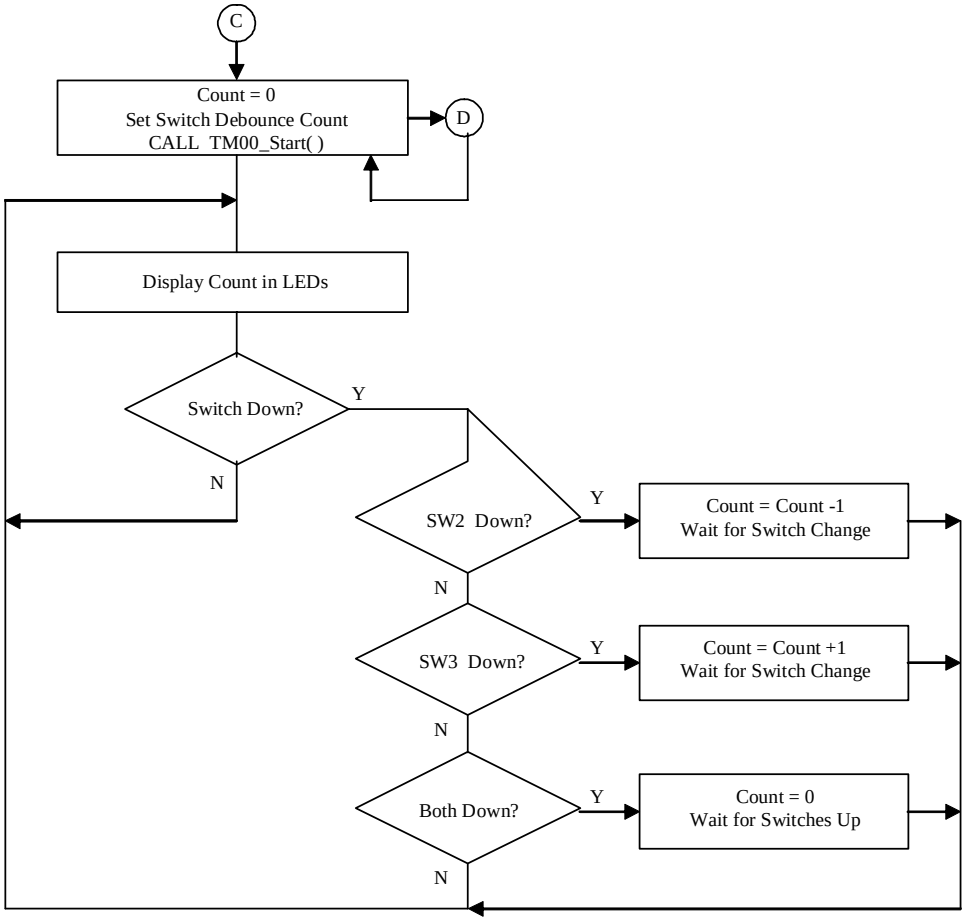
The prescaler mode register, PRM00, controls the timer clock and is set to 0x00. With this setting, Timer 00 uses fprs, the main system clock of 20 MHz, as its count clock. As a result, the count register TM00 counts up once every 50 nanoseconds.

The registers related to the INTTM000 interrupt are set for the selected low priority, and the interrupt flag is cleared. The mask register controlling the enabling of the timer interrupt is left in the default disabled state.

The CRC00 register is set to have CR000 act as a compare register, and the compare value of 0x4E1F is set. This value has CR000 match TM00 every (0x4E1F + 1), or 20,000 counts. This match thus occurs once every 20,000 * 50 nanoseconds = 1,000,000 nanoseconds = 1 millisecond. The Applilet calculated the value 0x4E1F to provide the 1 millisecond interval (specified in an Applilet detail dialog box; more on this later).

2.3.3 Main( ) - Main Program – Interval Timer for Switch Debouncing

Figure 6. Main Program Flow



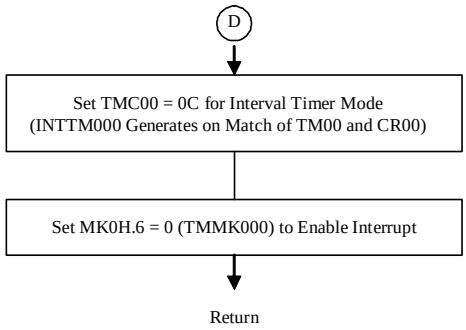
The main program sets the initial value for its count variable, calls a routine to initialize switch debouncing, and calls TM00_Start() to start Timer 00 counting. The program then enters an endless loop.

During this loop, the main program displays the current count value on the LEDs. It then checks whether either switch is down by calling a routine to get the current debounced state of the switches. If the program sees that the switches are up, it goes back to the top of the loop and displays the count. The program takes no other action until it sees a debounced switch value. If the program sees a switch pressed, the program changes the count variable and updates the display with the new count.

During the program loop, INTTM000 interrupts are occurring every 1 millisecond, and the MD_INTTM000() interrupt service routine is invoked.

2.3.4 TM00_Start( ) – Start Timer 00 Operation

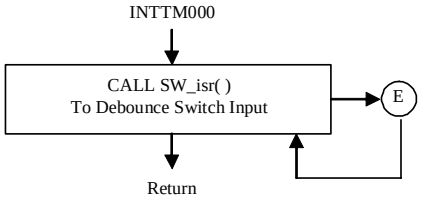
Figure 7. Starting Timer



The TM00_Start() routine sets the TMC00 register to start interval timer operation and clears the TMMK000 bit in the MK0H interrupt mask register to enable the INTTM000 interrupt.

2.3.5 MD_INTTM000( ) - Interrupt Service Routine For INTTM000

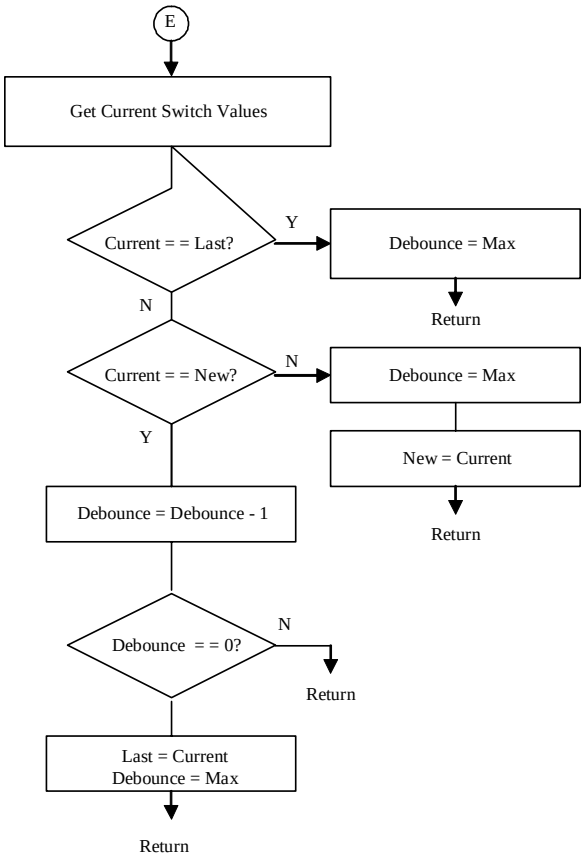
Figure 8. Interrupt Service Routine



The MD_INTTM000() interrupt service routine is called when the INTTM000 interrupt occurs and in turn calls the sw_isr() function in sw_0537.c to debounce the switches.

2.3.6 SW_isr( ) - Switch Debounce Interrupt Service

Figure 9. Switch Debounce Interrupt Flow



This switch debounce interrupt routine (not generated by the Applilet), is called once every 1 millisecond. The routine compares the current value of the switches to previous values. Once the values have been stable for the specified number of debounce counts, the routine updates the sw_new variable to provide a debounced switch setting. The main program checks this debounced value by calling the routine sw_get().

2.4 Applilet's Reference Driver

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

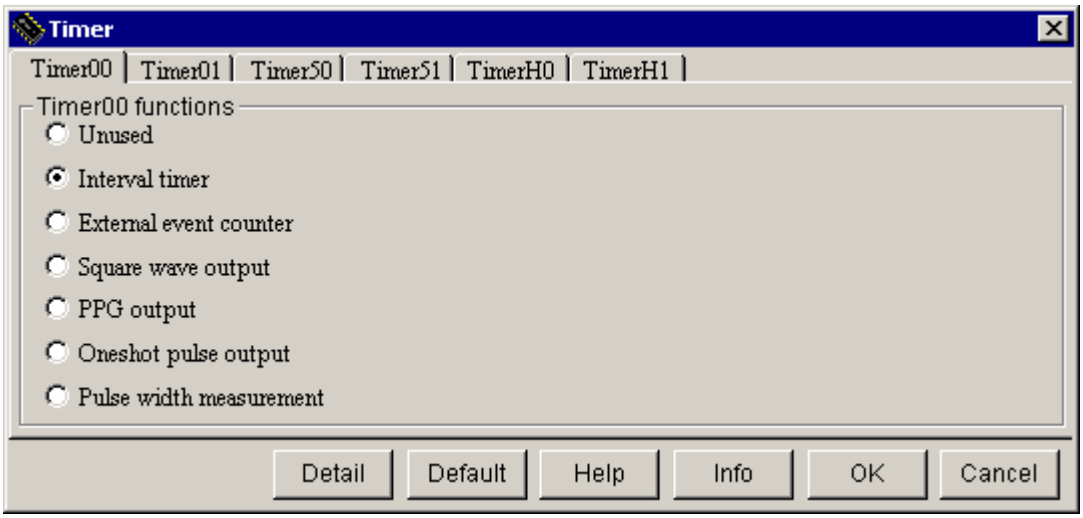
The Applilet is used to produce the basic initialization code and driver code for Timer 00, as well as code to manage the I/O ports used for switch input and LED display output. After the Applilet produces the basic code, you can add code to customize the program.

The information that follows describes how to set up the Applilet to produce code for Timer 00 and lists the routines produced. In general, the procedure is this: When you start the Applilet and select the target device, save your settings to a new project (.prx) file. The Applilet displays a dialog box that lets you select the peripheral blocks you want to set up.

2.4.1 Configuring Applilet for Timer 00

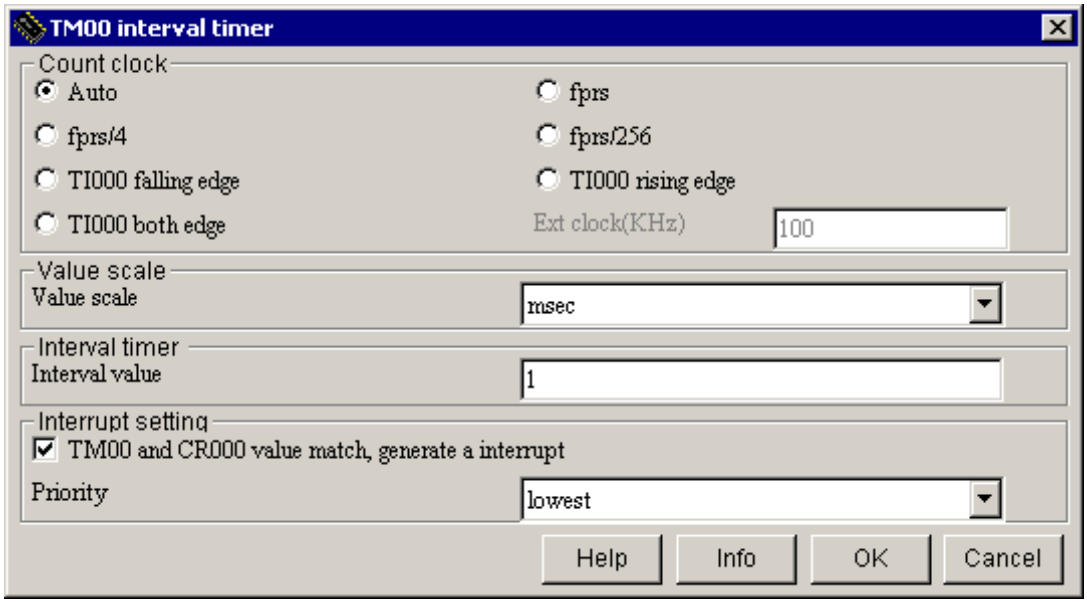
Selecting **Timer** in the Applilet main dialog box brings up a dialog box showing the various timer blocks. For this demonstration, select the **Timer00** tab and click **Interval timer**.

Figure 10. Timer Peripherals Shown in Applilet Setup



Once you select Timer 00, the **Detail** button is available for Timer 00 settings in the selected mode. Clicking **Detail** brings up the **TM00 interval timer** dialog box.

Figure 11. Detail Settings for Timer 00 Interval Timer



To generate an interrupt every 1 millisecond, set the value scale to **msec** (milliseconds) and the interval value to **1**. Leave the setting for the count clock as **Auto**. This setting causes the Applilet to select an appropriate setting for the timer clock selection register that allows a 1 millisecond interval.

To generate an interrupt when the timer interval is done, select the check box under **Interrupt setting** to generate an interrupt when TM00 (the timer count register) and CR000 (the timer compare register) match. Set the priority for this interrupt to **lowest**, since this application does not require a precise interval.

2.4.2 Generating Code With Applilet

Once the **TM00 interval timer** dialog box is set up, click OK. Select **Generate code** from the Applilet's main dialog box. The Applilet shows the peripherals and functions to be generated and allows you to select a directory in which to store the source code. When you click **Generate**, the Applilet generates the necessary initialization code, interrupt service routines, and driver subroutines for Timer 00.

The Applilet creates this code in several C-language source files (extension .c) and header files (extension .h) and shows the list of files created in a dialog box. To support Timer 00, the Applilet generates timer.h, timer.c and timer_user.c, as well as several other files not specifically related to Timer 00.

2.4.3 Applilet-Generated Timer 00 Files and Functions

As mentioned, the code generated for Timer 00 support is in the files timer.h, timer.c and timer_user.c. The following sections describe the contents of these files.

2.4.3.1 Timer.h

The header file timer.h contains declarations for the functions controlling Timer 00 and definitions of values for Timer 00 initialization. The header file macrodriver.h, used for all Applilet generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

2.4.3.2 Timer.c

The source file Timer.c contains the following functions generated by the Applilet for Timer 00:

void TM00_Init(void)
The TM00_Init() routine initializes the Timer 00 peripheral as specified in the Applilet Timer 00 detail dialog.

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

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

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

The array_reg parameter is a pointer to an array of values to be put in one of the compare registers; the array_num parameter is either 1 (to select CR000) or 2 (to select both CR010 and CR000). The demonstration program does not use this routine because the demonstration does not involve changing the interval time for Timer 00.

2.4.3.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. When the Applilet generates code, these functions are therefore empty to allow you to add application-specific code.

The interrupt service routine for the Timer 00 interrupt INTTM000 is:

__interrupt void MD_INTTM000(void)
The MCU generates this interrupt when TM00 and CR000 values match. Once the timer starts, the MCU generates this interrupt every millisecond.

As mentioned, this interrupt service routine is blank when the Applilet generates it. To use the interval timer to debounce the pushbutton switches, you need to place a call to the function sw_isr() in MD_INTTM000().

The sw_isr() routine is located in sw_0537.c and has code to check the value of the switches every millisecond. Once the value remains stable for the required number of milliseconds, the routine reports the new value as the debounced setting of the switches. After the routine reports this value, sw_isr() returns to MD_INTTM000(), which then returns to the main program where the interrupt occurred.

The sw_isr() code could be placed directly in MD_INTTM000(). From this location the code would still save the time of the call and return to sw_isr(). However, placing sw_isr() in sw_0537.c allows all switch-related functions to reside in the sw_0537.c source file for clarity.

2.4.4 Applilet-Generated Files Not Related to Timer 00

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

Table 3. Applilet-Generated Files Not Related to Timer 00

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
System.h	Clock-related definitions
System.c	Clock_Init() function
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.asm	Definitions of the option byte and security bytes
Option.inc	Definition of settings for the option byte and security settings

2.4.5 Demonstration Program Files Not Generated By Applilet

The demonstration program includes several files not generated by the Applilet, as shown below.

Table 4. Demonstration files not generated by Applilet

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

2.5 Demonstration Platform

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

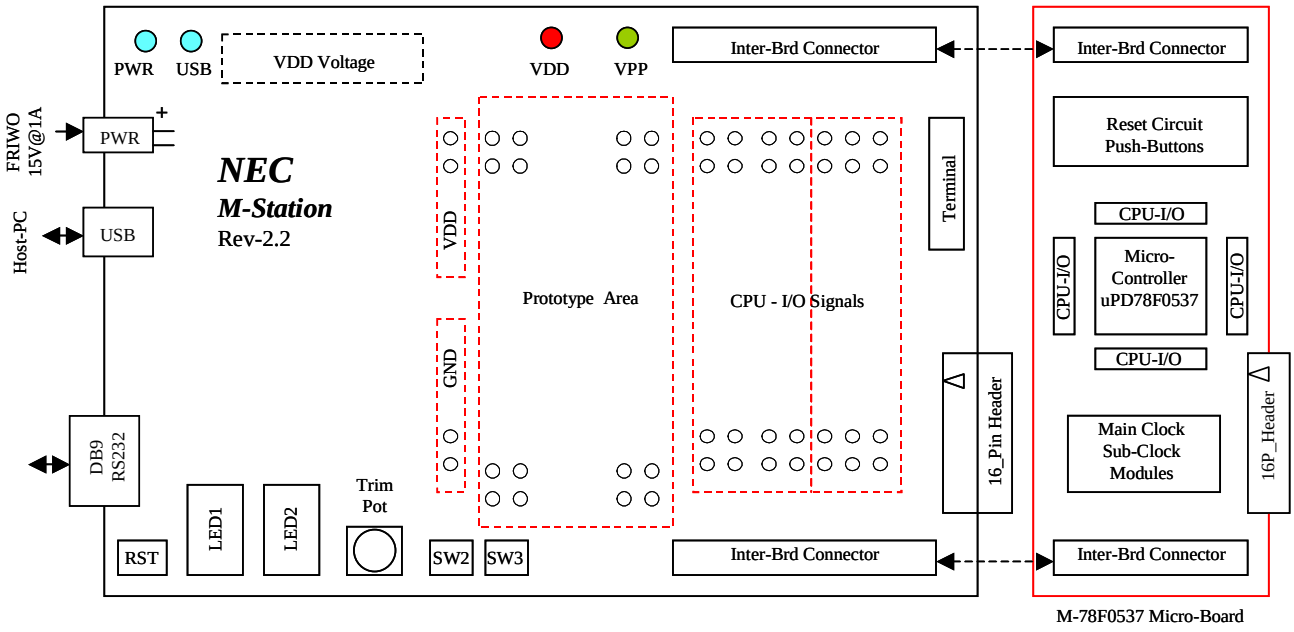
2.5.1 Resources

To run the demonstration program, the following resources are used:

- M-78F0537 Micro-Board with uPD78F0537 8-bit microcontroller mounted
- M-Station II Evaluation System using M-Station II resources:
  - 7-Segment LEDs LED1 and LED2
  - Pushbutton switches SW2 and SW3

For details on this hardware, please refer to the appropriate User's Manual, available from NEC Electronics.

Figure 12. Micro-Board for Interval Timer Demonstration



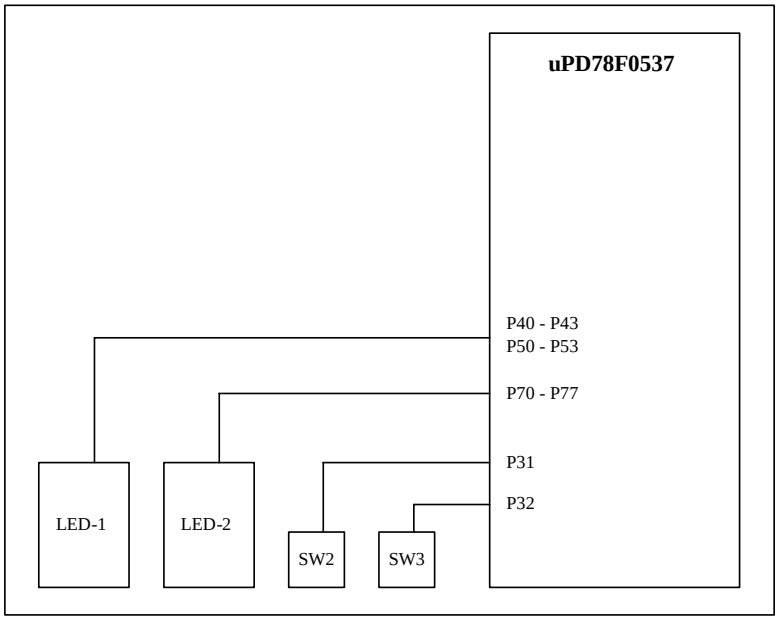
2.5.2 Demonstration of Program

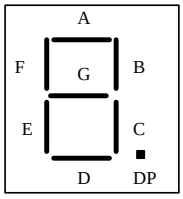
With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Press SW2 Observe decrementing numbers
- Press SW3 Observe incrementing numbers

2.6 Hardware Block Diagram

Figure 13. Hardware Layout for Interval Timer Demonstration





LED-1 and LED-2

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

2.7 Software Modules

The following files make up the software modules for the demonstration program. The table below shows which files are generated by the Applilet and which of those need modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 5. Files for Interval Timer Demonstration

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	Yes
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Led_0537.h	LED display definitions	--	Yes
Led_0537.c	LED display functions	--	Yes
Sw_0537.h	Switch input definitions	--	Yes
Sw_0537.c	Switch input functions	--	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

3. External Event Counter Using 16-Bit Timer 00 (TM00)

You can use timer/counters to monitor external events, either counting the number of events or measuring times between events. An external event is the edge of a signal. You can choose either a negative (1 -> 0), or positive edge (0 -> 1), or both, present at an input pin to act as the event.

This demonstration uses 16-bit Timer/Counter 00 (TM00) to count external events and 8-bit timer/counter TM50 as a square-wave generator. Connecting the square-wave output to the input of TM00 allows the timer to treat the square-wave's edges as events. TM00 detects these events and counts them, producing an interrupt after a specified number of counts. The program displays a count of how many interrupts have occurred, providing a visual representation of the event frequency.

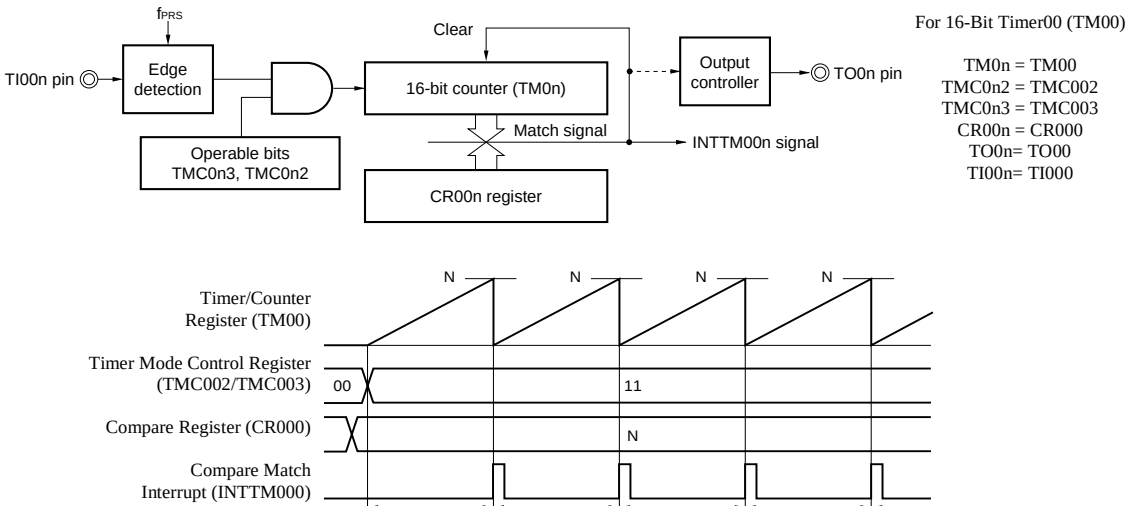
3.1 External Event Counter Features

When configured as an external-event counter the 16-bit timer offers:

- Counting of external events on falling edge, rising edge, or both edges
- Counts from 1 to 65535 events
- Optional interrupt after a specified number of events

If you select TI000 input as an external clock input and set the timer/counter-mode register to clear-and-start, the counter begins counting in synchronization with the clock at the input TI000. The counter treats TI000 as a source of external events. After each count, the counter compares the value in the timer/counter register with that in the capture/compare register (CR000) and generates an interrupt when the two register values match. The interrupt-service routine clears the counter and starts it counting from zero again. Thus, the value you store in the compare register represents the number of events you want to detect.

Figure 14. Configuring Timer 00 As An External-Event Counter



The important facts about this use of the counter are:

- Prescaler mode register = 11 — Select TI000 as the count clock
- Timer/counter mode register = 11 — Clear-and-start mode
- Compare register value — Contains the number of external events to detect
- Compare match interrupt — An interrupt is generated when match occurs

To configure the timer as an external-event counter:

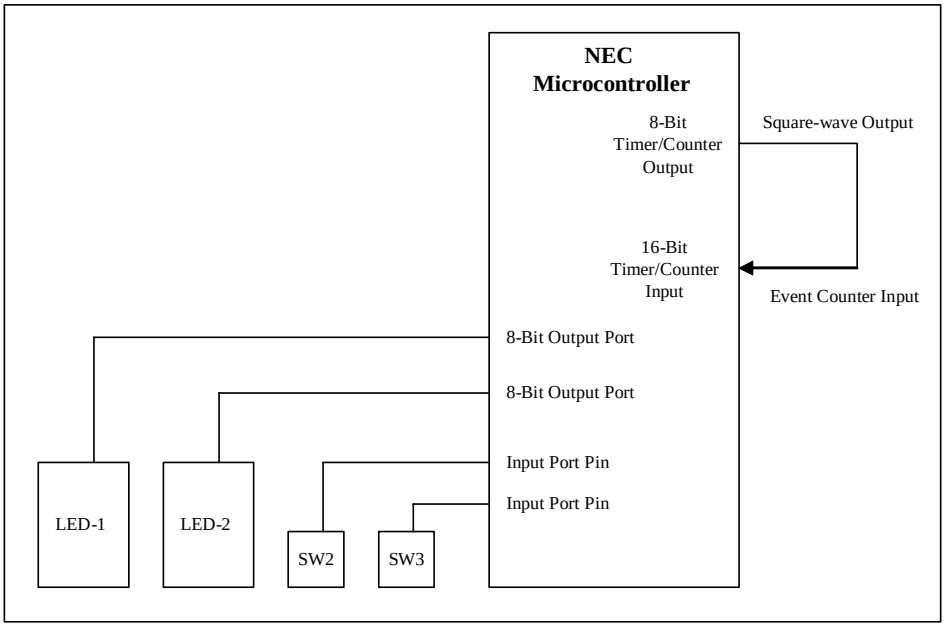
- Set the edge(s) to be counted, and TI000 edge input, in the PRM00 register
- Set the number of events in CR000; set CRC00 to use CR000 as a compare register
- Set the priority for the interrupt, clear the interrupt flag, and unmask the interrupt
- Enable the timer by setting the timer-mode register TMC00 for clear-and-start mode on match of TM00 and CR000

3.2 Program Description and Specification

The demonstration program uses 8-bit timer TM50 to generate a square-wave output, which feeds to the input of 16-bit Timer/Counter TM00. TM00 sees the edges of the square wave as a series of external events. It detects them, counting up to a preset number, at which time it generates an interrupt that resets and restarts the event count.

A 2-digit, 7-segment LED displays the number of times the interrupt occurs, providing a visual indication of the rate of the events. The square-wave generator reads two external switches that determine the square-wave frequency. Pressing one switch sets the square-wave output to a low frequency (50 Hz), resulting in a slow count, while pressing the other sets the frequency higher (100 Hz).

Figure 15. Demonstration Block Diagram



This demonstration uses the 8-bit timer/counter TM50 as follows:

- TM50 generates external events.
- TM50's ouput connects to the timer/counter TI000 input.
- TM50 changes output when the timer/counter overflows (treated as an external event).
  - Selecting the fast clock produces more frequent events.
  - Selecting a slow clock reduces event frequency.

Switch inputs perform the following functions:

- Pressing SW2 produces a higher frequency (100 Hz).
- Pressing SW3 produces a lower frequency (50 Hz).

Use the 16-bit timer/counter TM00 in the following way:

- Select TI000 as an external-event input clock by setting the prescale-mode register.
- Set the timer/counter mode register to clear-and-start mode.
- Set the compare register to the number of external events you want to detect.

The 2-digit display:

- Updates every time the 16-bit timer/counter generates a match interrupt
- Displays the number of sets of external events detected in ascending order—1, 2, …99

Specifications:

- TM00 counts the falling edges present at TI000 and interrupts after 50 counts.
- TM50 outputs a 50- or 100-Hz square wave on the TO50 output.

3.3 Software Flow Charts

The demonstration program consists of:

- Program-initialization code, called before the main() program starts
- The main-program loop, which checks the status of switches and changes the timing of TM50
- Subroutines generated by the Applilet to handle TM00 and TM50 starting, stopping and changing interval
- Subroutines (with user code) for handling TM00 interrupts (Applilet-generated stub interrupt service routines, with user code added). Note that no interrupt is generated for TM50.
- Subroutines for pushbutton input and LED display output

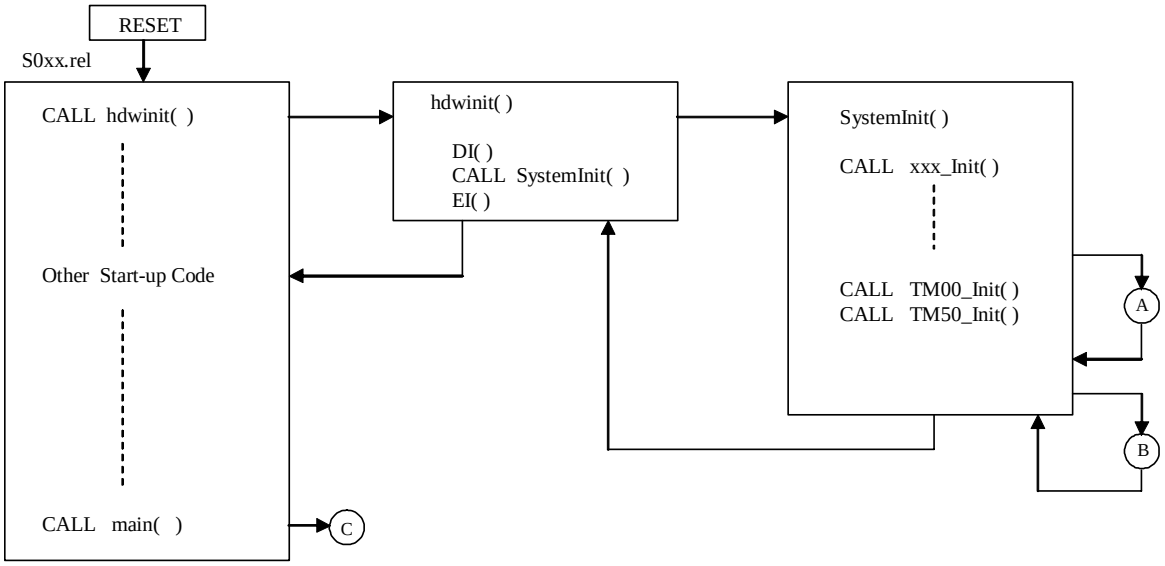
The following flowcharts describe the initialization, the main program, subroutines related to TM00 and TM50, and the interrupt-service routine for TM00.

3.3.1 Program Startup and Initialization

For 78K0 programs written in the C language, an object-code file, such as s01.rel, supplies the startup code that is linked to the user program. This startup code calls a function named hdwinit( ) and you can place hardware-specific initialization code here.

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

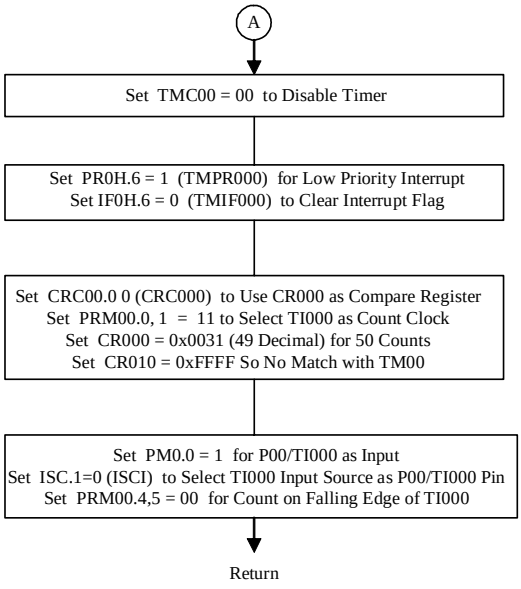
Figure 16. Program Startup and Initialization Flow



After the hdwinit( ) function finishes, the startup code calls the main( ) function of the user program. When the main( ) function starts, it does not need to call individual peripheral initialization routines.

3.3.2 TM00_Init( ) -- Timer 00 Initialization

Figure 17. Initializing Timer 00



SystemInit() calls the TM00_Init() routine, which sets the Timer 00 registers to prepare for external event-counter operation. The routine disables the timer before changing register settings.

The TM00_Init() routine selects the registers related to the INTTM000 interrupt for low priority and clears the interrupt flag, leaving the mask register that controls the enabling of the timer interrupt in the default (disabled) state.

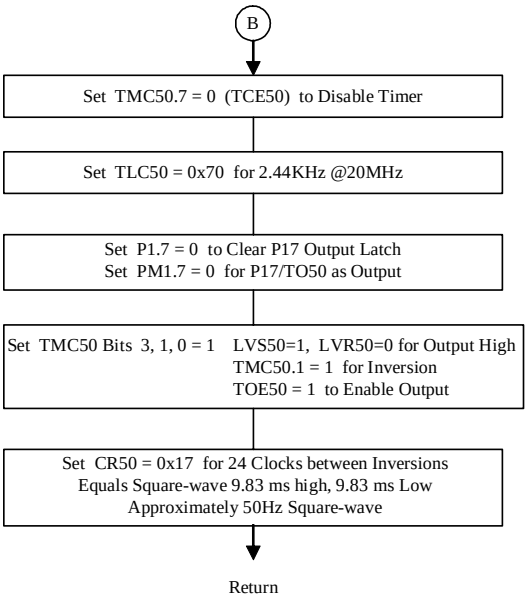
The initialization routine sets the CRC00 register to have CR000 act as a compare register. The routine also sets the prescaler-mode register (PRM00) for TI000 to act as Timer 00's count clock and to count up on the falling edge of TI000. Setting the compare register CR000 to 49 causes an interrupt every 50 counts, and setting CR010 to its maximum value prevents the register from ever matching TM00, so no interrupt occurs.

The initialization routine also sets port-mode register (PM0) bit 0 for P00 to act as an input pin, providing the P00/TI000 input. The port settings in the Port_Init() routine also reflect his setting of the port pin. Finally, TM00_Init() sets the ISC register to select the P00/TI000 pin as the input source for TI000.

The Applilet tool calculates the values and provides the code for the above settings based on the settings you provide in the Timer 00 dialog box.

3.3.3 TM50_Init( ) -- Timer 50 Initialization

Figure 18. Initializing Timer 50



The TM50_Init() routine disables the timer while writing to its registers, to configure the timer as a square-wave generator with an output frequency of approximately 50 Hz.

The initialization routine sets the TLC50 register 0x07; this value selects the clock as fprs/2**13. A main clock of 20 MHz results in a count clock of 20,000,000/8192, or 2.44 kHz. Thus, timer TM50 counts up every 409.6 microseconds.

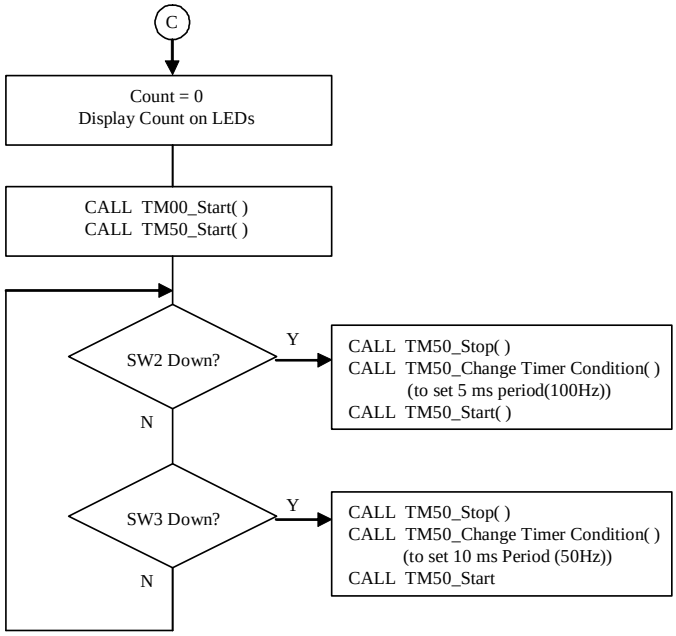
TM50 outputs its square wave on P17/TO50. The initialization routine sets P1.7 to zero to clear the port-output latch, then sets PM1.7 to zero to make the pin an output. Port_Init() also reflects these port settings.

The routine sets TMC50 at an initially high output that inverts when the values in TM50 and CR50 match. This arrangement produces a square wave, with the low and high widths determined by the interval between TM50/CR50 matches. With CR50 set to 0x17, or 23, the timer sees such a match between TM50 and CR50 every 24 count clocks. With 409.6 microseconds per count clock, a match occurs every 409.6 * 24 microseconds, or every 9.83 milliseconds. Low and high widths of approximately 10 milliseconds produces a square wave of approximately 50 Hz.

The Applilet calculates the values and provides the code for the above settings based on the settings in the Timer 50 dialog box.

3.3.4 Main( ) – Main Program — External Event Counter

Figure 19. Main Program



The main program initializes the count variable and displays it on the LEDs. The program then calls the TM00_Start() and TM50_Start() functions to start the timers.

After this setup, the main program loops endlessly—simply calling the sw_get() function in sw_0537.c to check the status of the two switches. If no switch is down, the program does nothing.

Timer 50 produces a continuous 50-Hz square wave at the P17/T050 output, which directly connects to the P00/TI000 input. On each falling edge at the TI000 input, Timer 00 counts up. After every 50 counts (about 1 second), the values in TM50 and CR000 match, causing the timer to generate the INTTM000 interrupt, which calls interrupt-service routine MD_INTTM000(). This interrupt-service routine increments the count and displays it on the LEDs.

So, if you do nothing at all, the display counts up about once per second.

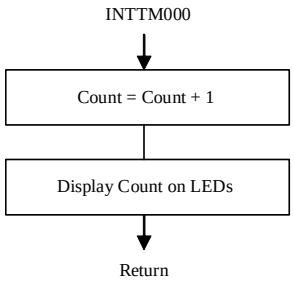
If you press SW2, the main program calls the TM50_Stop() function to halt the square-wave generator. The program then calls the TM50_ChangeTimerCondition function to changes the value of CR50 to half its previous value, resulting in a 100-Hz square wave instead of 50 Hz. The program then calls TM50_Start() to restart the timer.

At this point, the display counts up about twice per second.

If you press SW3, the program steps through the same procedures but sets CR50 back to its original value. The display resumes counting up once per second.

3.3.5 MD_INTTM000( ) — Interrupt Service Routine for INTTM000

Figure 20. Interrupt-Service Routine



The INTTM000 interrupt causes the MD_INTTM000() interrupt-service routine to execute, incrementing the global-count variable and displaying that count on the LEDs.

3.4 Applilet's Reference Driver

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

The Applilet produces the basic initialization and driver code for Timer 00 and Timer 50, as well as code that manages the I/O ports used for switch input and LED display output. After the Applilet produces the basic code, you can add additional code to customize the program.

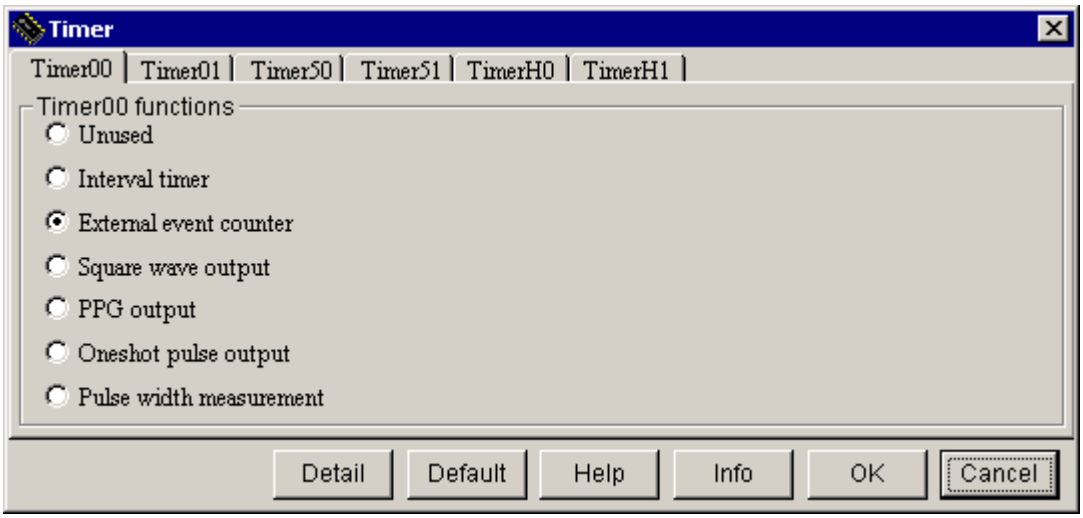
This section describes how the Applilet is set up to produce code for Timer 00 and Timer 50 and lists the routines produced.

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

3.4.1 Configuring Applilet for Timer 00

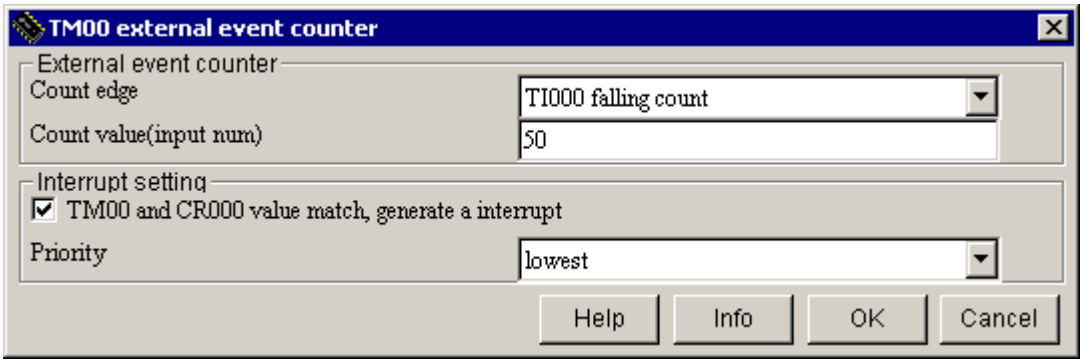
Selecting **Timer** brings up a dialog box showing the various timer blocks. Select **Timer 00** and click **External event counter**.

Figure 21. Configuring Timer 00



Once you have selected **Timer 00** and **External event counter**, clicking **Detail** brings up the detail dialog box for external-event counter settings.

Figure 22. External-Event Counter Detail Dialog Box



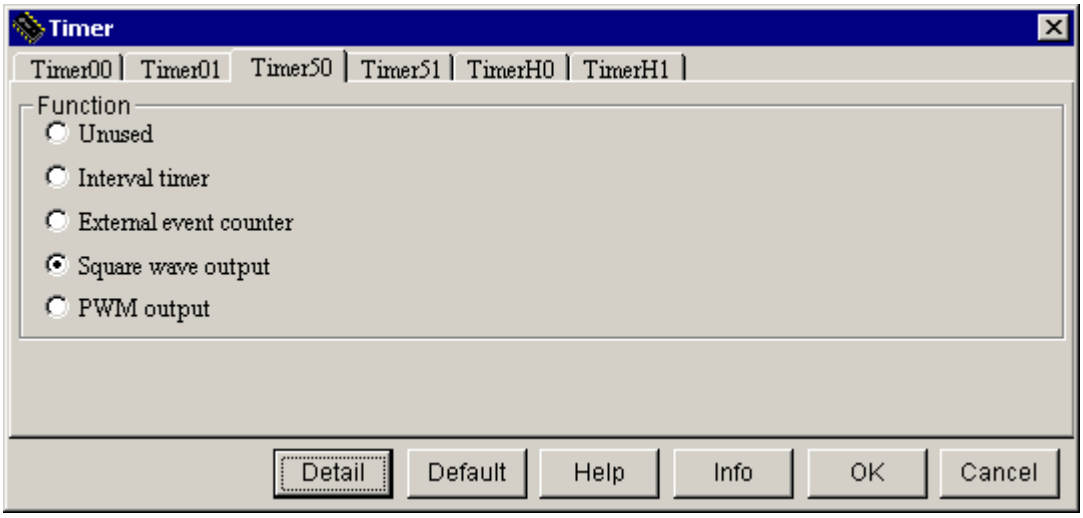
The timer should generate an interrupt after every 50 counts, so set **Count value** to 50 and **Count edge** to count on the falling edges of TI000.

To generate an interrupt when the count reaches 50, click **interrupt setting** and select the priority as lowest.

3.4.2 Configuring Applilet for Timer 50

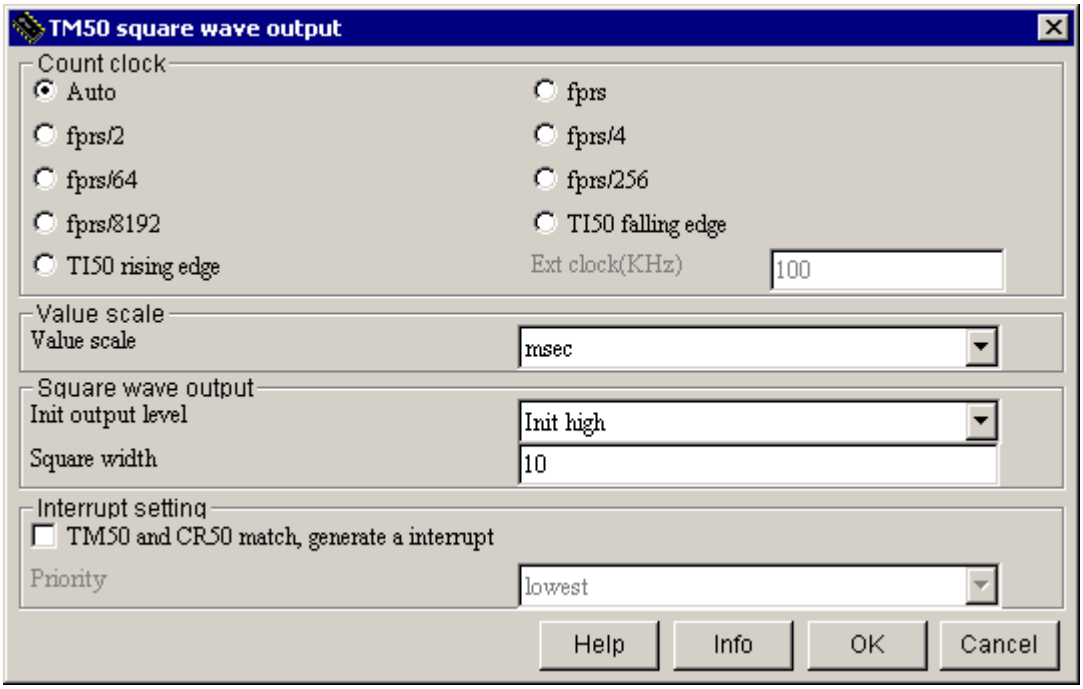
After selecting **Timer** and then **Timer 50,** click **Square-wave output**.

Figure 23. Applilet's Timer-Selection Dialog Box



Once you click **Square-wave output,** clicking **Detail** brings up the dialog box for square-wave output settings.

Figure 24. Dialog Box For Square-Wave Output Settings



The timer should generate a 50-Hz square wave that is high for about 10 milliseconds and then low for about 10 milliseconds.

Set **Value scale** to milliseconds (**msec**) and **square width** to **10**. Set the **Init output level** to **Init high**. Setting **Count clock** to **Auto** lets the Applilet generate the appropriate clock-divider registers value for the set timing.

Leave **Interrupt setting** unselected.

3.4.3 Generating Code with Applilet

Once you have set up the values in the dialog boxes, select **Generate code** from the Applilet's main dialog box. The Applilet displays the peripherals and functions and lets you select a directory for the source code. The Applilet then generates the initialization code, interrupt-service routines, and driver subroutines for Timer 00 and Timer 50.

The Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and shows the list of files created in a dialog box. To support Timer 00 and Timer 50, the Applilet generates timer.h, timer.c and timer_user.c, as well as several other files not related to the timers.

3.4.4 Applilet-Generated Timer 00 and Timer 50 Files and Functions

The files timer.h, timer.c, and timer_user.c contain the code generated for Timer 00 and Timer 50 support.

3.4.4.1 Timer.h

The header file timer.h contains declarations for the functions controlling Timer 00 and Timer 50, and definitions of values for Timer 00 and Timer 50 initialization. The header file macrodriver.h, used for all Applilet generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

3.4.4.2 Timer.c

The source file Timer.c contains:

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

void TM00_Start(void)
The TM00_Start() routine enables Timer 00 and the INTTM000 interrupt.

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

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

The array_reg parameter points to an array of values that get stored in one of the compare registers; the array_num parameter is either 1 (to select CR000) or 2 (to select both CR010 and CR000). Since the demonstration does not need to change the counter compare values for Timer 00, this routine is not used in the demonstration program.

void TM50_Init(void)
The TM50_Init() routine initializes Timer 50.

void TM50_Start(void)
The TM50_Start() routine starts Timer 50 by enabling it.

void TM50_Stop(void)
The TM50_Stop() routine stops Timer 50 by disabling it.

MD_STATUS TM50_ChangeTimerCondition(UCHAR value)
The TM50_ChangeTimerCondition() function changes the value in compare register CR50 and therefore changes the square-wave high/low width for Timer 50.

3.4.4.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. These functions are empty on code generation so that you can add application-specific code.

__interrupt void MD_INTTM000(void)
This is the interrupt-service routine for interrupt INTTM000. Once the timer starts, it generates this interrupt once every 50 counts.

The Applilet generates this interrupt service routine without code. You must add the code required to increment the count variable and display it on the LEDs.

3.4.5 Applilet-Generated Files Not Related to Timer 00 and Timer 50

The Applilet generates several other source files required for the demonstration.

Table 6. Applilet-Generated Files Not Related to Timers

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
System.h	Clock-related definitions
System.c	Clock_Init() function
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

3.4.6 Demonstration Program Files Not Generated By Applilet

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

Table 7. Additional Files Needed by Demonstration Program

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

3.5 Demonstration Platform

The demonstration platform for the external event counter is 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.

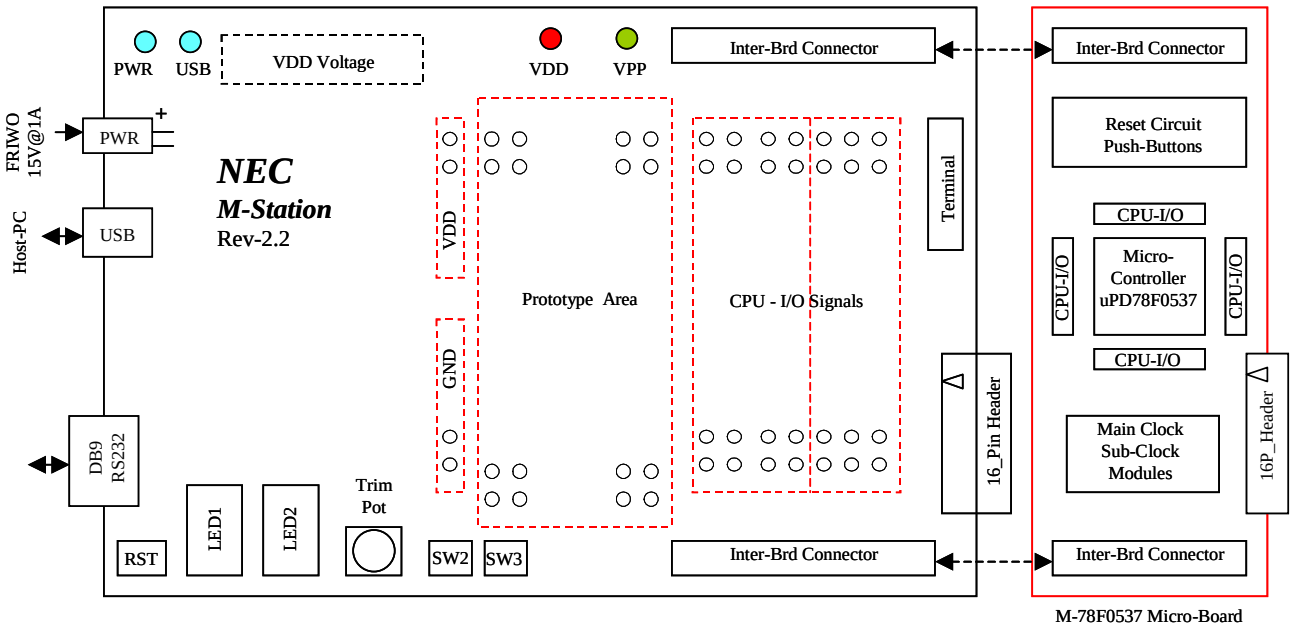
3.5.1 Resources

This program demonstration uses the following resources:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- M-Station II Evaluation System, using M-Station II resources:
  - 7-Segment LEDs LED1 and LED2
  - Pushbutton switches SW2 and SW3

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

Figure 25. Demonstration Hardware



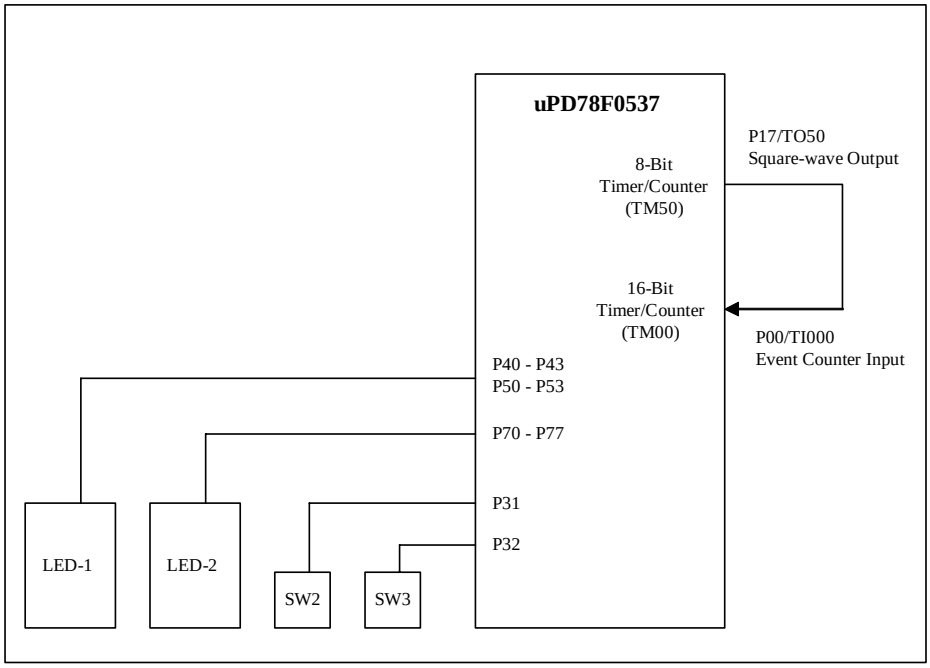
3.5.2 Demonstration of Program

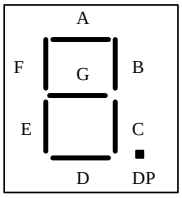
With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Press SW3 Observe speed at which display increments
- Press SW2 Observe display incrementing faster

3.6 Hardware Block Diagram

Figure 26. Demonstration Hardware Block Diagram





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

3.7 Software Modules

The demonstration program uses the following files. The table shows which files the Applilet generates completely and which needed modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 8. Software Modules Used for Demonstration

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	Yes
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Led_0537.h	LED display definitions	--	Yes
Led_0537.c	LED display functions	--	Yes
Sw_0537.h	Switch input definitions	--	Yes
Sw_0537.c	Switch input functions	--	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

4. One-Shot Pulse Generation Using 16-Bit Timer 01 (TM01)

This demonstration illustrates use of the 16-bit Timer 01 (TM01) to generate one-shot pulses. The counter outputs a one-shot pulse when triggered by software that is initiated by pressing an external switch. The pulse generated on the output of TM01 is connected to the input of 16-bit Timer/Counter 00 (TM00), which is configured as an external-event counter. TM00 counts the pulses generated by TM01 and increments a counter display after detecting a preset number of pulses.

4.1 16-Bit Interval Timer Configured For One-Shot Pulses

The timer outputs one-shot pulses when:

- Timer mode-control register bits are set to generate one-shot pulses
- The output-control register bits are set to enable one-shot operation

In the one-shot pulse mode, the timer offers:

- A variable time base, based on divisions of the peripheral clock, using the PRM00 register
- Trigger by software or external input
- Variable delay from the trigger to the pulse start using CR010 to count time-base clocks
- Variable pulse width by setting the time from trigger to end of pulse with CR000
- Optional interrupt on start or end of pulse

Figure 27. Block Diagram of One-Shot Pulse Generator

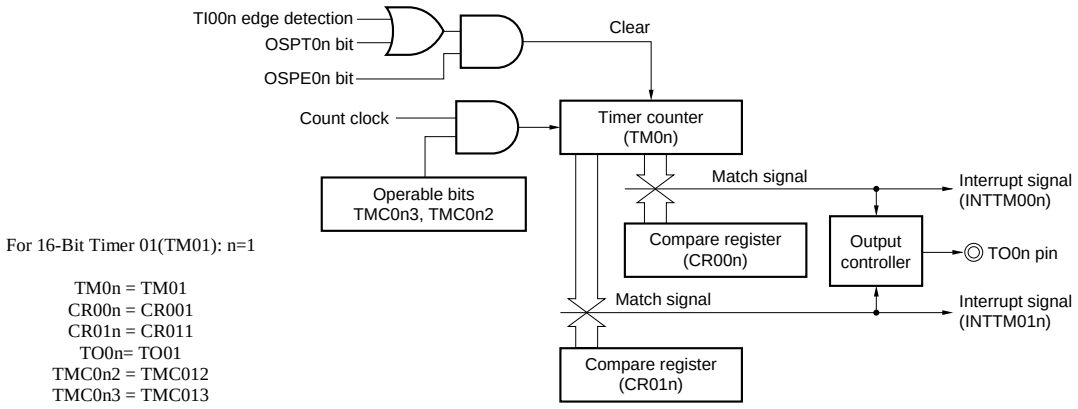
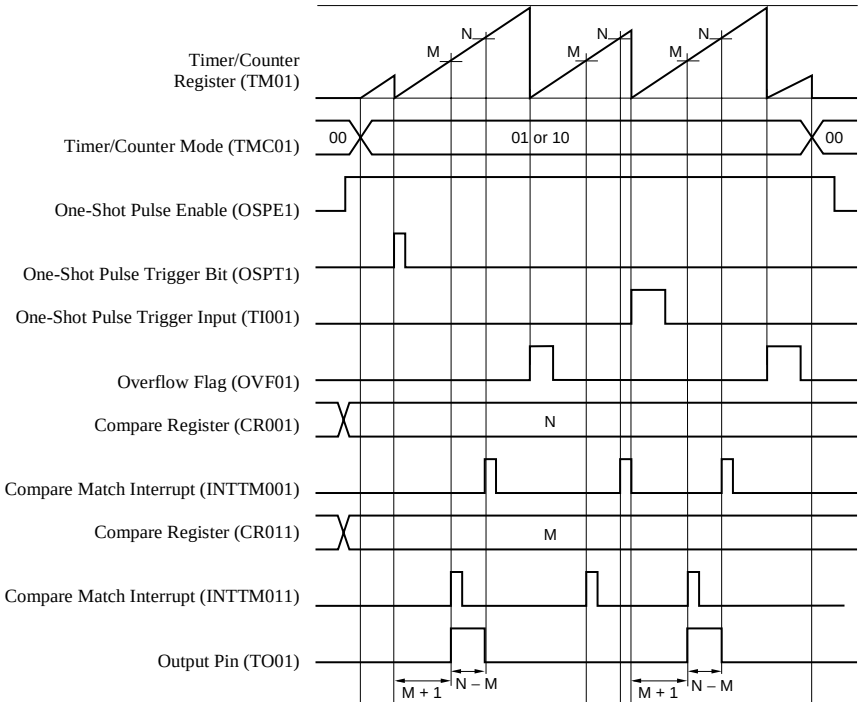


Figure 28. One-Shot Timing Diagram



One-shot pulse output active-level width = (N-M) * count clock cycle

Say, for example, that you set the timer/counter mode register for free-running mode and set the output-control register to enable one-shot and software trigger. This configurations sets a software trigger in the control register and triggers the clear-and-start of the timer/counter (TM00). The timer outputs a pulse with a width equal to the difference between two compare registers (CR000 and CR010).

To configure Timer 01 for one-shot operation with software trigger:

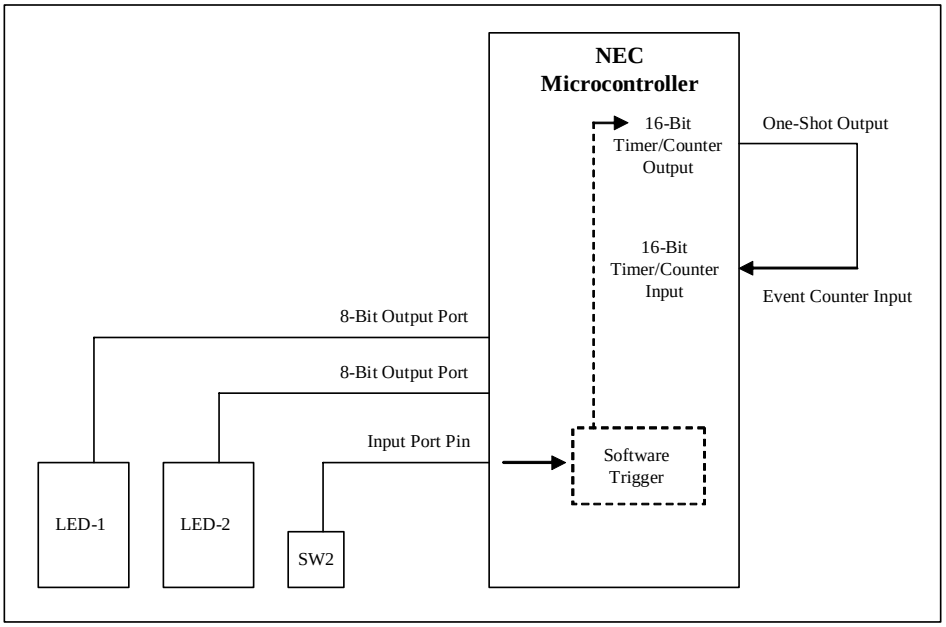
- Set the time base using PRM01; if using external trigger, set the valid edge of the input
- Set CRC01 to use CR001 and CR011 as compare registers
- Set the time to pulse start using CR010 and time to pulse end using CR001
- Set the appropriate port-mode registers and latches for the output pin used
- Set the TOC01 register to enable the output, set initial state, set inversion of output on CR001 and CR011 match of TM01, and set one-shot enable bit
- Set the priority for the interrupts, clear the interrupt flags, and unmask the interrupts if used

To start the timer for software triggering, set the TMC01 register for free-running mode. (To start the timer for external trigger, set the TMC01 register for clear-and-start mode.)

4.2 Program Description And Specification

The demonstration program uses an external switch to initiate software triggering for TM01. Each time you press the switch it generates one pulse. The one-shot output is input to TM00 in external-event counter mode. TM01 counts the number of pulses input (number of external events) and interrupts after a set number of pulses. The LEDs display the number of interrupts.

Figure 29. Demonstration Hardware Block Diagram



Specifications:

- Pressing SW2 causes a software trigger of TM01.
- TM01 produces a one-shot pulse, going high 50 microseconds after triggering and going low 100 microseconds after triggering.
- TM00 counts falling edges of TI000 and generates an interrupt after five counts.

4.3 Software Flow Charts

The demonstration program consists of the following major sections:

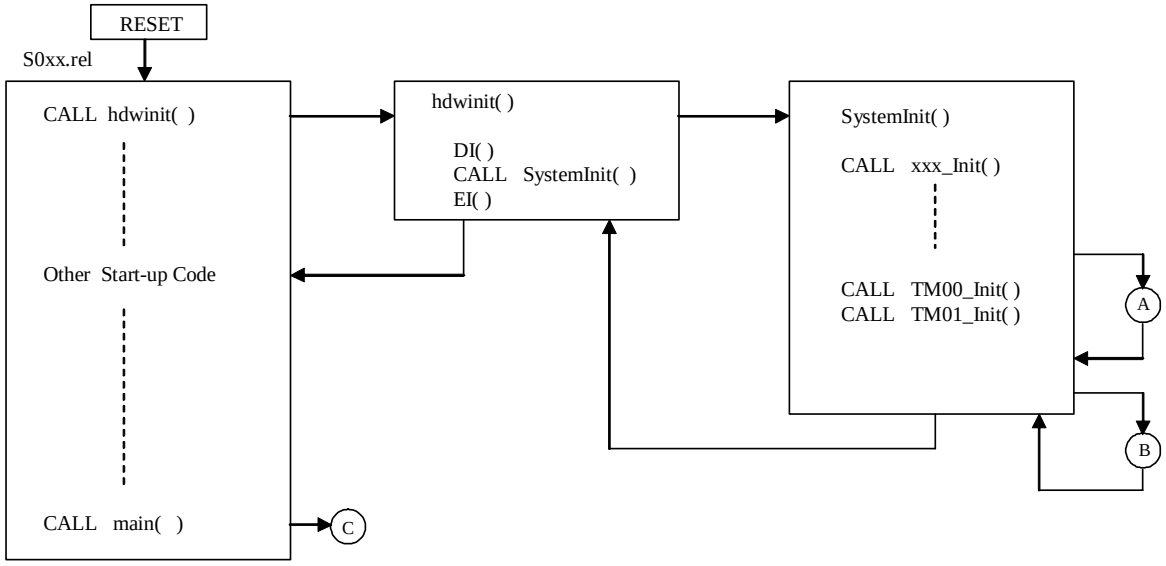
- Initialization code for the program, called before the main() program starts
- The main program loop, which checks the status of switches and triggers TM01

- Subroutines generated by the Applilet to handle TM00 and TM01 starting, stopping, and changing interval
- Subroutines with user code for handling TM00 interrupts (Applilet-generated stub interrupt service routines, with user code added); note that no interrupt is generated for TM01
- Subroutines for pushbutton input and LED display output

The following flowcharts describe the initialization, the main program, subroutines related to TM00 and TM01, and the interrupt-service routine for TM00.

4.3.1 Program Startup And Initialization

Figure 30. Program Startup Flow



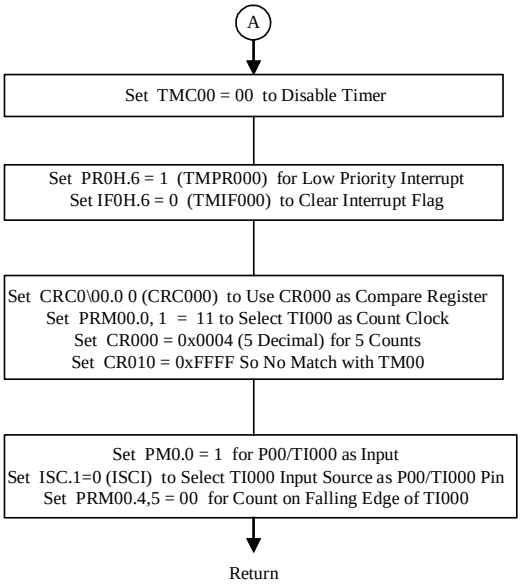
For 78K0 programs written in the C language, an object-code file such as s01.rel provides the startup code. This file is linked into the user program and calls a function named hdwinit( ). You can place hardware initialization code here.

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

When hdwinit( ) completes, the startup code calls the main( ) function of the user program. So at the start of the main( ) function, all peripherals have been initialized, and the main( ) function does not need to call individual initialization routines.

4.3.2 TM00_Init( ) – Timer 00 Initialization

Figure 31. Flowchart for Initializing Timer



SystemInit() calls the TM00_Init() routine, which sets the timer registers for external event-counter operation. TM00_Init() disables the timer while changing the register settings.

The initialization routine sets the registers related to the INTTM000 interrupt for low priority and clears the interrupt flag. The mask register controlling the enabling of the timer interrupt is left in the default disabled state.

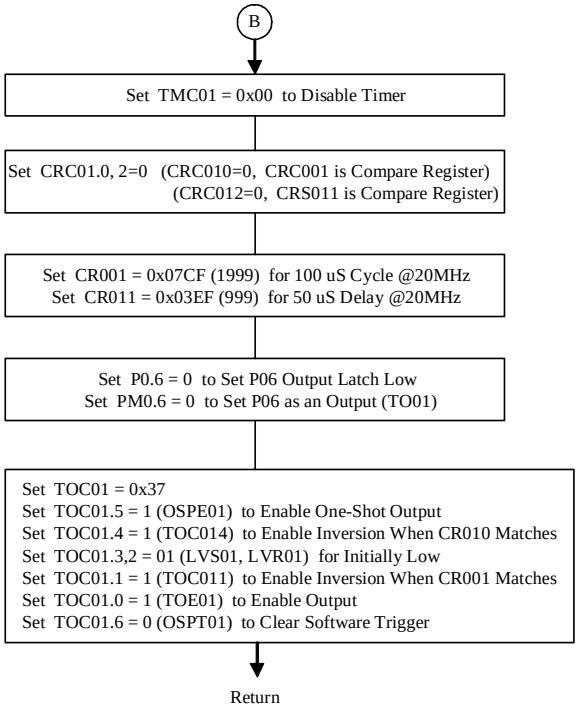
The initialization routine sets the CRC00 register for CR000 to act as a compare register. The routine sets the prescaler-mode register PRM00 to have TI000 act as the count clock and to count up on the falling edge of TI000. The compare register CR000 is set to 4, which causes an interrupt every 5 counts. CR010 is set to the maximum value, so no interrupt occurs when CR010 matches TM00.

The initialization routine sets the port-mode register PM0 bit 0 for P00 to act as an input pin, providing the P00/TI000 input. This selection is also reflected in the port settings in the Port_Init() routine. The ISC register is also set to select the P00/TI000 pin as the input source for TI000.

The Applilet calculates the values and provides the code for the above settings based on the settings in the Timer 00 dialog box (more on this later).

4.3.3 TM01_Init( ) -- Timer 01 Initialization

Figure 32. Initializing Timer 01



In addition to calling the TM00_Init() routine, SystemInit() calls the TM01_Init() routine to initialize this timer for one-shot pulse generation. TM01_Init() disables the timer while writing to the control registers.

Since Timer 01 does not generate interrupts, there is no modification of interrupt-control registers, and the Timer 01 interrupt INTTM001 is left disabled.

The prescaler-mode register (PRM01) controls the clock input, and TM01_Init() sets this register to 00 to specify the count clock as fprs, which is the 20-MHz system clock. This setting results in TM01 counting up every 0.05 microseconds.

TM01_Init() sets CR001 to 0x7CF (1999). This value is compared with TM01 after 2000 counts, or 2000 * 0.05 = 100 microseconds. This value sets the one-shot cycle time, at which the pulse goes low again. CR011 is set to 0x3EF (999); a comparison with TM01 occurs after 1000 counts, or after 1000 * 0.05 microseconds = 50 microseconds. This setting controls the delay for the one-shot; the signal goes high 50 microseconds after the trigger.

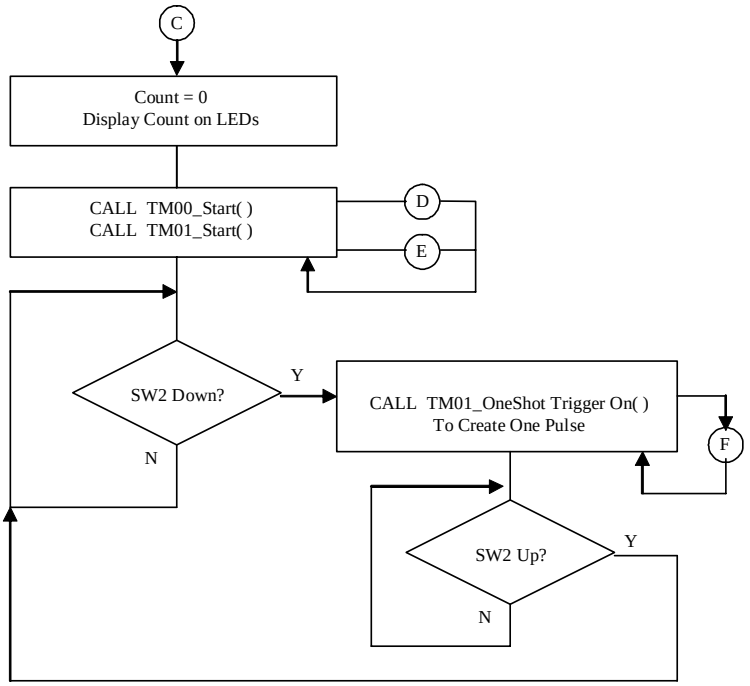
CRC01 is set to use CR001 and CR011 as compare registers.

To send the one-shot output to P06/TO01, the routine sets the P0.6 output latch and the port-mode register (PM0) bit 6 to zero.

The TOC01 output control register for Timer 01 is set to enable one-shot output, which will be inverted when both CR001 and CR011 match TM01. The TM01_Init() routine sets the output as initially low and clears the software trigger.

4.3.4 Main( ) – Main Program — One-Shot Pulse Generation

Figure 33. Main Program Flow



The main program sets the count variable to zero initially and displays the count on the LEDs. The program then calls TM00_Start() and TM01_Start() to start the timers.

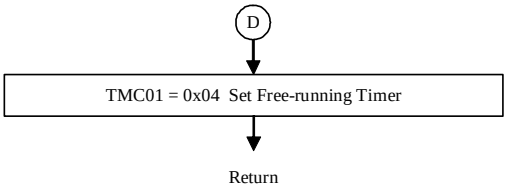
The program checks the state of SW2; if the switch is up, no action is taken.

If SW2 is down, the program calls TM01_OneShotTriggerOn() to trigger the one-shot. This routine causes one pulse to be output on P06/TO01. Because this pin is connected to P00/TI000, the pulse causes one event count in the Timer 00 external-event counter. The program then waits for SW2 to be released.

The result of this program is that for every five times you press SW2, it generates five counts for Timer 00 and, when the event counter matches the set count, it generates interrupt INTTM000. This interrupt invokes the interrupt-service routine MD_INTTM000, which counts up and displays the count variable.

4.3.5 TM01_Start( ) – Start Timer 01 Operation

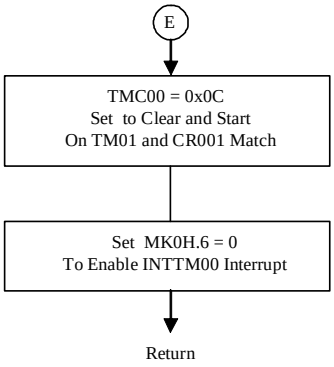
Figure 34. Flowchart for Starting Timer 01



The TM01_Start() function sets the TMC01 register to enable the timer's one-shot pulse-generation mode. A software trigger can then generate the one-shot pulse.

4.3.6 TM00_Start( ) – Start Timer 00 Operation

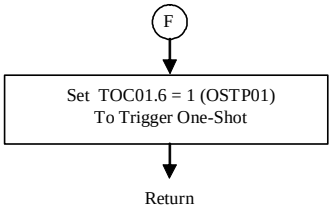
Figure 35. Flowchart for Enabling Timer's One-Shot Mode.



TM00_Start() sets TMC00 to clear-and-start the external-event counter in Timer 00, and enables the INTTM000 interrupt on count compare.

4.3.7 TM01_OneShotTriggerOn( ) – Trigger One-Shot Pulse on Timer 01

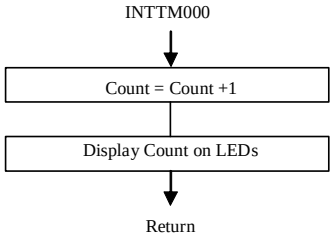
Figure 36. Flowchart for Configuring Software Trigger



The TM01_OneShotTriggerOn() function sets the OSTP01 (one-shot software trigger bit) in register TOC01, which clears TM01 and starts one-shot operation. Timer 01 generates one pulse of the one-shot and does not create another pulse until this routine is called again.

4.3.8 MD_INTTM000( ) – Interrupt-Service Routine

Figure 37. Flowchart for Interrupt-Service Routine



When INTTM000 occurs, it invokes the MD_INTTM000() interrupt-service routine to increment the global count variable and display the count on the LEDs.

4.4 Applilet's Reference Driver

NEC Electronics' Applilet program generator automatically generates C or assembly language source code to manage peripherals for the 78K0 family of devices. Please see the Appendix for the version of the Applilet used.

The Applilet produces the basic initialization and driver code for the timers, as well as code to manage the I/O ports used for switch input and LED display output. After the Applilet produces the basic code, you can add additional code to customize the program.

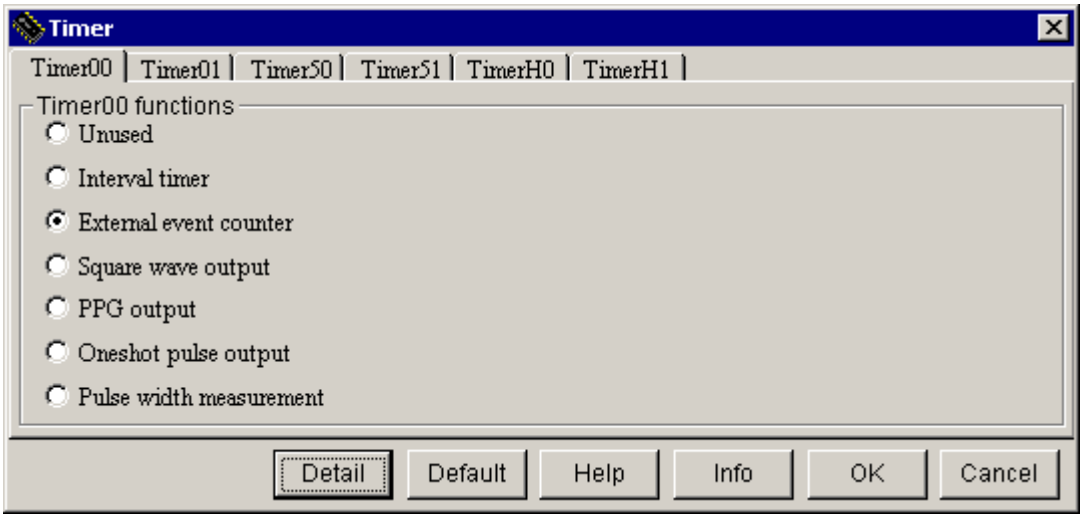
The following section describes how the Applilet is set up to produce code for Timer 00 and Timer 01, and lists the routines produced.

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

4.4.1 Configuring Applilet for Timer 00

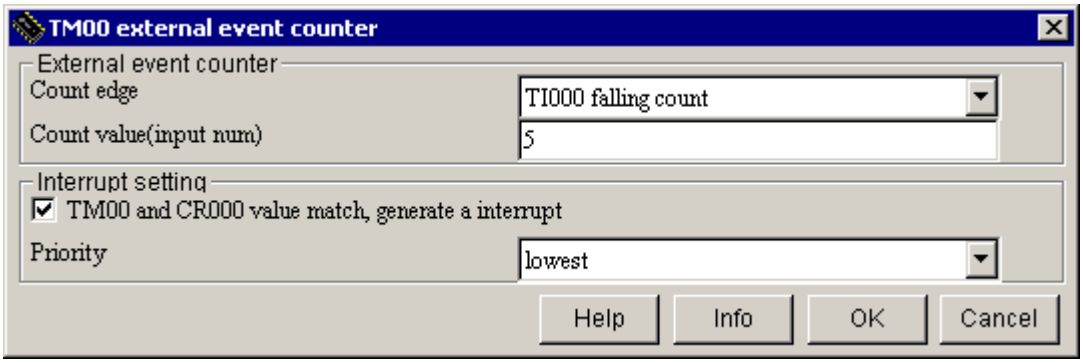
Selecting **timer** brings up a dialog box showing the various timer blocks. For this demonstration, select **Timer 00** and click on **External event counter**.

Figure 38. Timer Selection Dialog



Once you have selected Timer 00 as an external-event counter, clicking **Detail** brings up the **TM00 external event counter** dialog box.

Figure 39. Applilet Detail Dialog Box For External-Event Settings



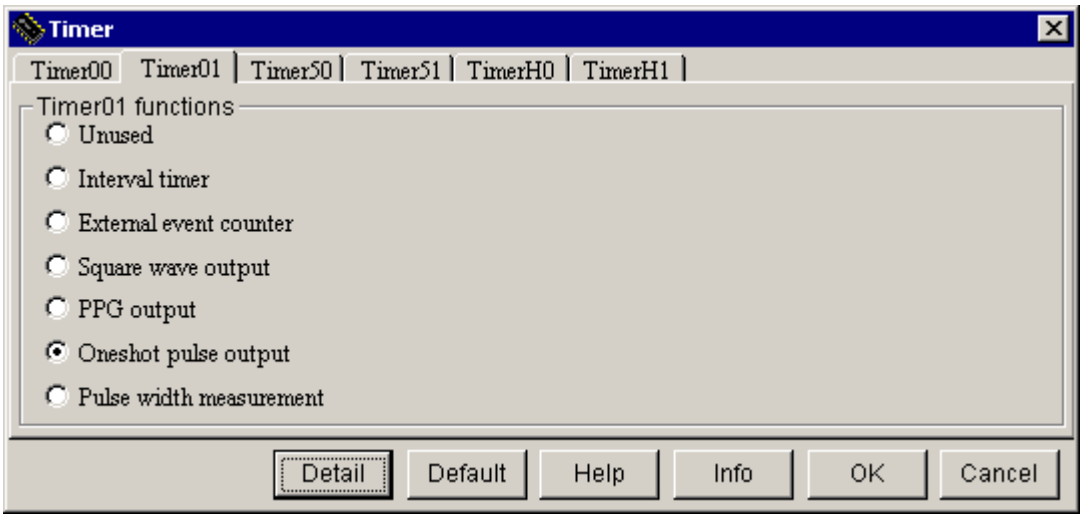
To configure the timer to generate an interrupt on every 5 counts of the input TI000, set **Count value** to **5**, and **Count edge** to count on **falling** edges of TI000.

Since the program must generate an interrupt when the count reaches 5, click **Interrupt setting** to generate an interrupt when TM00 (the Timer 00 timer count register) and CR000 (the Timer 00 compare register) match. Set **Priority** for this interrupt to **lowest**.

4.4.2 Configuring Applilet for Timer 01

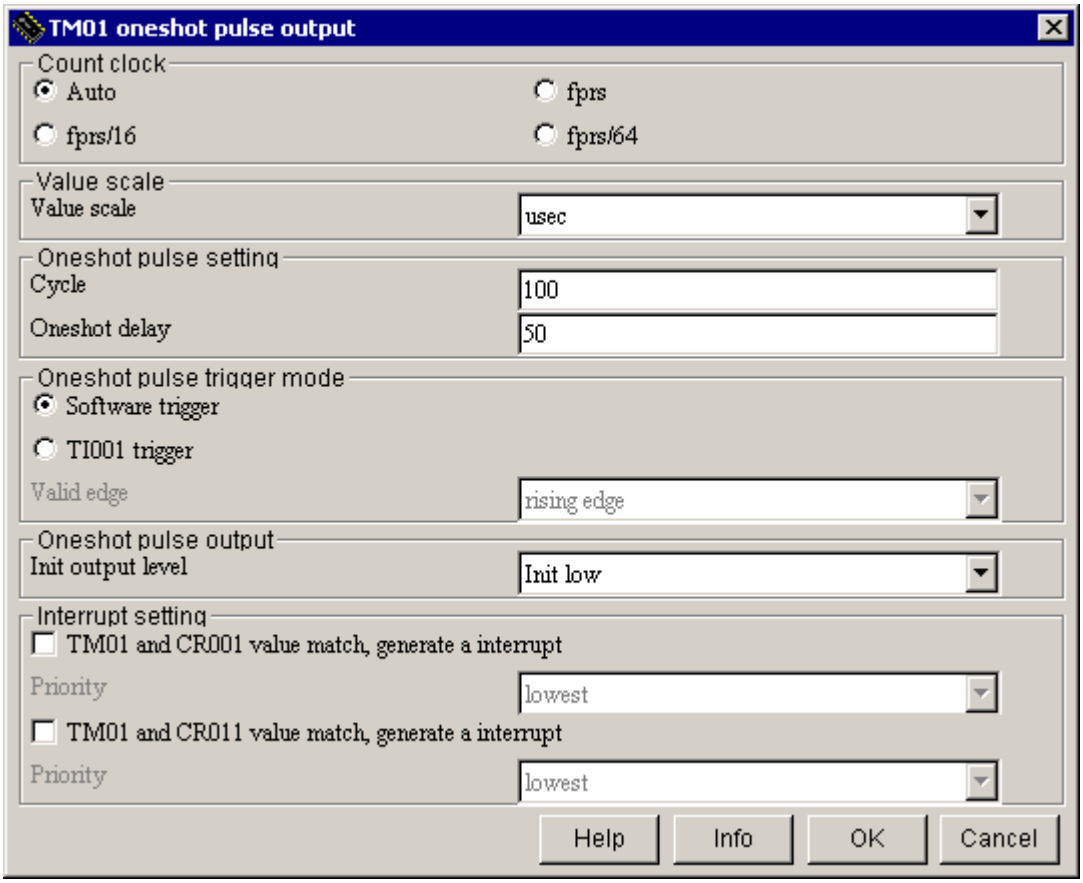
Selecting **Timer** and then **Timer 01** shows the available settings. Select **One-shot pulse output**.

Figure 40. Dialog Box for Timer 01 Functions



Once you have selected the function for **Timer 01**, click on **Detail** to bring up the dialog box for **One-shot pulse output** settings.

Figure 41. One-Shot Output Detail Dialog Box



For this demonstration, you want the timer to generate a one-shot pulse, triggered by software, with the pulse 50 microseconds wide and delayed for 50 microseconds.

Set **Value scale** to microseconds. Set **Cycle** (delay plus On time) to 100 to get 100 microseconds cycle time, and **One-shot delay** to 50 microseconds. Setting **Count clock** to AUTO allows the Applilet to generate the appropriate value for the clock-divider registers.

Click **Software trigger.** The demonstration program triggers the one-shot when you press switch SW2. Set the **Initial output level** to **Init low** (initially low).

Leave the **Interrupt setting** boxes unchecked. Since there is no interrupt generated, the priority setting for the interrupt is grayed out.

4.4.3 Generating Code with Applilet

Once you have set up the timer dialog boxes, select **Generate code** from the Applilet's main dialog box. The Applilet displays the peripherals and functions, and allows you to select a directory for the source code. The Applilet then generates the initialization code, interrupt-service routines, and driver subroutines for Timer 00 and Timer 01.

When you click **Generate**, the Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and shows the list of files created in a dialog box. To support Timer 00 and Timer 01, the Applilet generates timer.h, timer.c and timer_user.c, as well as several other files not related to the timers.

4.4.4 Applilet-Generated Timer 00 and Timer 01 Files And Functions

The Applilet stores the code generated for Timer 00 and Timer 01 in the files timer.h, timer.c and timer_user.c.

4.4.4.1 Timer.h

The header file timer.h contains declarations for the functions controlling Timer 00 and Timer 01, and definitions of values for Timer 00 and Timer 01 initialization. The header file macrodriver.h, used for all the Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

4.4.4.2 Timer.c

The source file Timer.c contains the following functions:

void TM00_Init(void)
The TM00_Init() routine initializes the timer.

void TM00_Start(void)
The TM00_Start() routine enables Timer 00 and the INTTM000 interrupt.

void TM00_Stop(void)
The TM00_Stop() routine disables the timer and the interrupt. The demonstration program does not use this routine.

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

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

void TM01_Init(void)
The TM01_Init() routine initializes Timer 01.

void TM01_Start(void)
The TM01_Start() routine enables Timer 01.

void TM01_Stop(void)
The TM01_Stop() routine disables the timer.

MD_STATUS TM01_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
The TM01_ChangeTimerCondition() function changes the value in the Timer 01 compare registers, CR001 and CR011, and therefore changes the one-shot cycle and delay values for the Timer 01 one-shot.

The array_reg parameter is a pointer to an array of values to be put in one or the other of the compare registers; the array_num parameter is either 1 (to select CR001) or 2 (to select both CR011 and CR001). The demonstration program does not use this routine.

void TM01_OneshotTriggerOn()
The TM01_OneshotTriggerOn() routine sets the software trigger for the Timer 01 one-shot pulse generation.

4.4.4.3 Timer_user.c

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

__interrupt void MD_INTTM000(void)
This is the interrupt-service routine for the Timer 00 interrupt INTTM000, generated when TM00 and CR000 values match. Once the timer is started, it generates this interrupt once every 5 counts.

On generation by the Applilet, this interrupt-service routine is blank. To use the external-event counter to update the display, code is added to increment the count variable and display the count on the LEDs.

4.4.5 Applilet-Generated Files Not Related to Timer 00 and Timer 01

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

Table 9. Applilet-Generated Files Not Related to Timers

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
System.h	Clock-related definitions
System.c	Clock_Init() function
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

4.4.6 Demonstration-Program Files Not Generated by Applilet

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

Table 10. Files Not Generated by Applilet

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

4.5 Demonstration Platform

The demonstration platform for the one-shot pulse generator is 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.

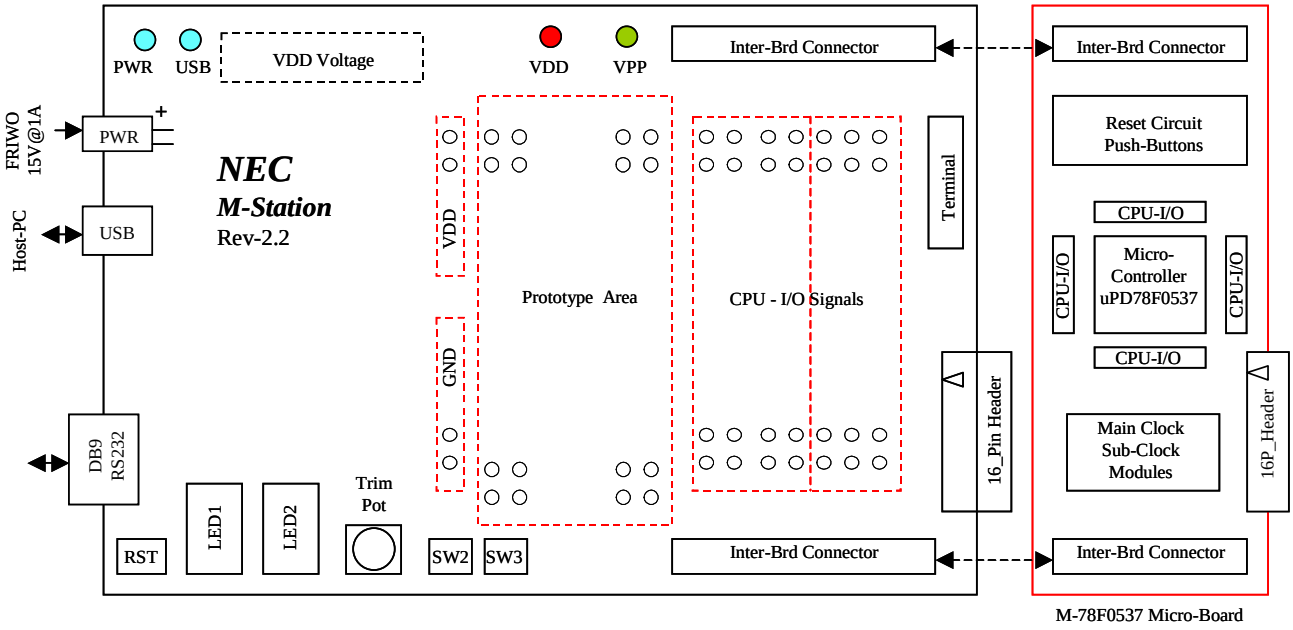
4.5.1 Resources

To demonstrate the program, the following resources are used:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- M-Station II Evaluation System, using M-Station II resources:
  - 7-Segment LEDs LED1 and LED2
  - Pushbutton switch SW2

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

Figure 42. Demonstration Hardware Platform



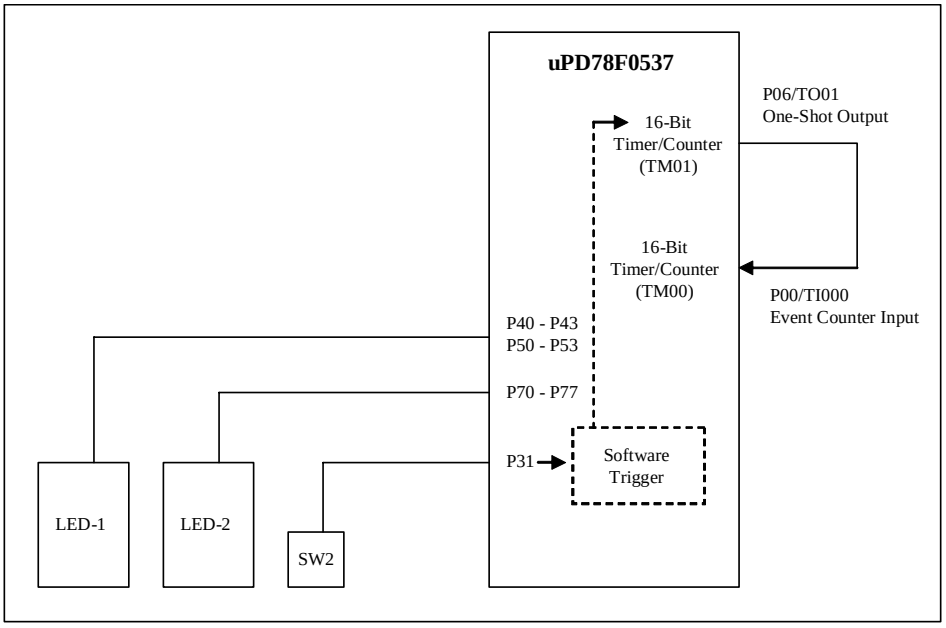
4.5.2 Demonstration of Program

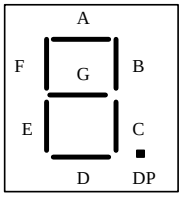
With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Press SW2 Observe the number of interrupts displayed on the LEDs.
- The event count should increment once for every five presses of the switch.

4.6 Hardware Block Diagram

Figure 43. Hardware Block Diagram





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

4.7 Software Modules

The following files make up the software modules for the demonstration program. The table shows which files are generated by the Applilet and which of those need modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 11. Complete List of Software Modules for Demonstration Program

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	Yes
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Led_0537.h	LED display definitions	--	Yes
Led_0537.c	LED display functions	--	Yes
Sw_0537.h	Switch input definitions	--	Yes
Sw_0537.c	Switch input functions	--	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

5. PWM Output for D/A Conversion Using 8-bit Timer51 (TM51)

You can use an 8-bit timer/counter to generate a fixed-frequency, variable-duty-cycle pulse-width modulation (PWM) output for a variety of applications. The following example demonstrates a low-cost, low-resolution solution to D/A-conversion PWM output.

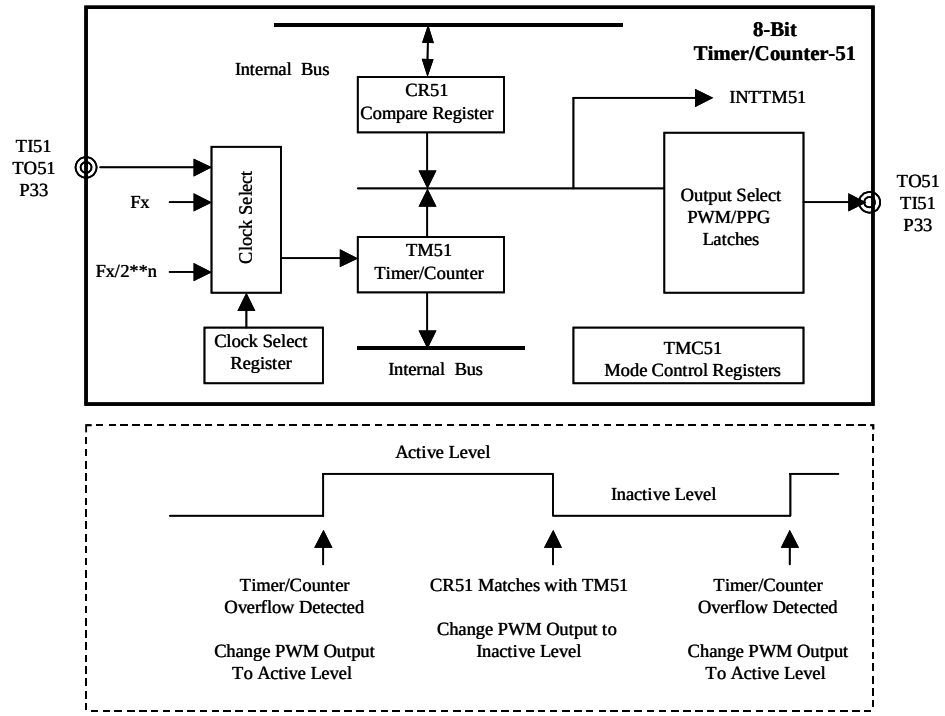
5.1 PWM Features

In PWM mode, the 8-bit timer offers the following features:

- Selection of the PWM count clock by dividing the main peripheral clock
- An output pulse active from 0/256 to 255/256 clocks
- Output active either high or low
- Optional interrupt at the start of the PWM cycle

When the timer/counter mode register is set to generate a PWM output, the timer/counter produces a fixed-frequency, variable-duty-cycle PWM signal.

Figure 44. Block Diagram for Timer/Counter Configured for PWM Output

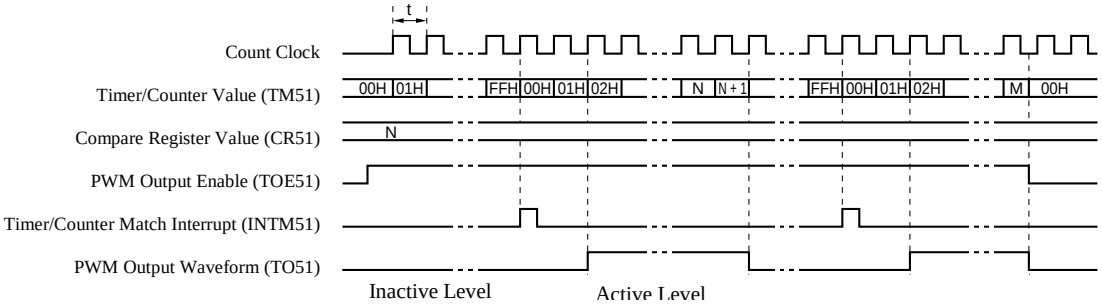


- Timer/counter mode control register — Set to produce PWM output
- Timer/counter counting clock — Count clock can be selected by clock selection register
- Compare register — Contains a value that determines the duty cycle

When the timer/counter overflows, it activates the PWM output. When a timer/counter and compare register match occurs (CR51 = TM51), the event deactivates the PWM output until the next overflow when the PWM output is again activated. The PWM output is:

- Cycle 2**8 x t
- Active Period N x t where N = 00h to FFh
- Duty cycle N/2**8

Figure 45. PWM Timing Waveform



To configure timer TM51 for PWM operations:

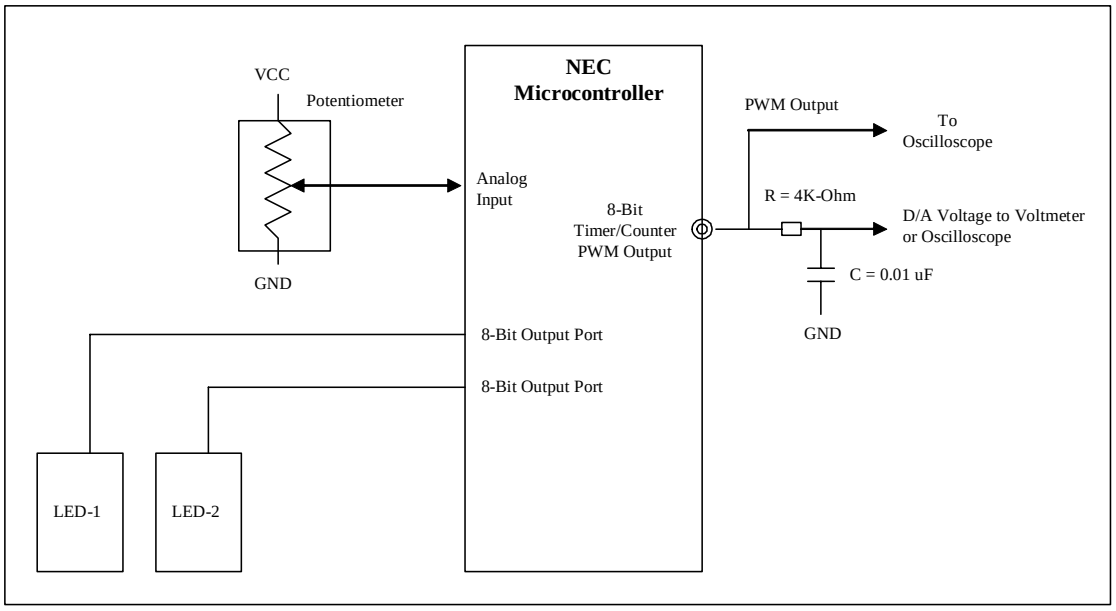
- Select the count clock using the TCL51 register.
- Set the output-port pin to output.
- Set the priority for the interrupt, clear the interrupt flag, and unmask the interrupt, if used.
- Set the PWM active time using the CR51 register.
- Set the PWM mode, active high or low, initial output state, and output enable using TMC51.
- Enable the timer with TMC51.7 (TCE51 bit).

5.2 Program Description and Specification

The program demonstrates digital-to-analog conversion using a PWM output. The PWM output passes through a low-pass filter to generate a time-averaged analog signal. The analog-signal amplitude is directly proportional to the PWM pulse width.

The demonstration program uses an analog signal input to an NEC Electronics microcontroller through a potentiometer. The A/D converter then converts the DC voltage at the potentiometer to a digital value. This A/D conversion value also displays on the LEDs.

Figure 46. Hardware Block Diagram for A/D Conversion



The A/D conversion value loads into a compare register to provide data for PWM generation. This approach produces about the same level of analog signal on the PWM output at the low-pass filter (the RC network shown). To test this operation, examine the analog signal into the low-pass filter with an oscilloscope and measure the analog voltage out of the low-pass filter with a voltmeter.

General design guidelines suggest that the PWM frequency should be 5 to 10 times that of the low-pass filter cut-off frequency to minimize ripple at the low-pass filter output. Thus, choose resistor and capacitor values for the low-pass filter to cut off at approximately 4 kHz when the PWM frequency is at its maximum value of approximately 78 kHz, using the 20-MHz main clock oscillator for the uPD78F0537.

Specifications:

- TM51 is set for PWM mode, active high, initial output low, no interrupt.
- TM51 PWM clock is set to 20 MHz; PWM cycle time is 20,000,000/256 = 78.125 kHz.
- AD0/P20 is set for A/D input in continuous-conversion mode.

5.3 Software Flow Charts

The demonstration program consists of the following major sections:

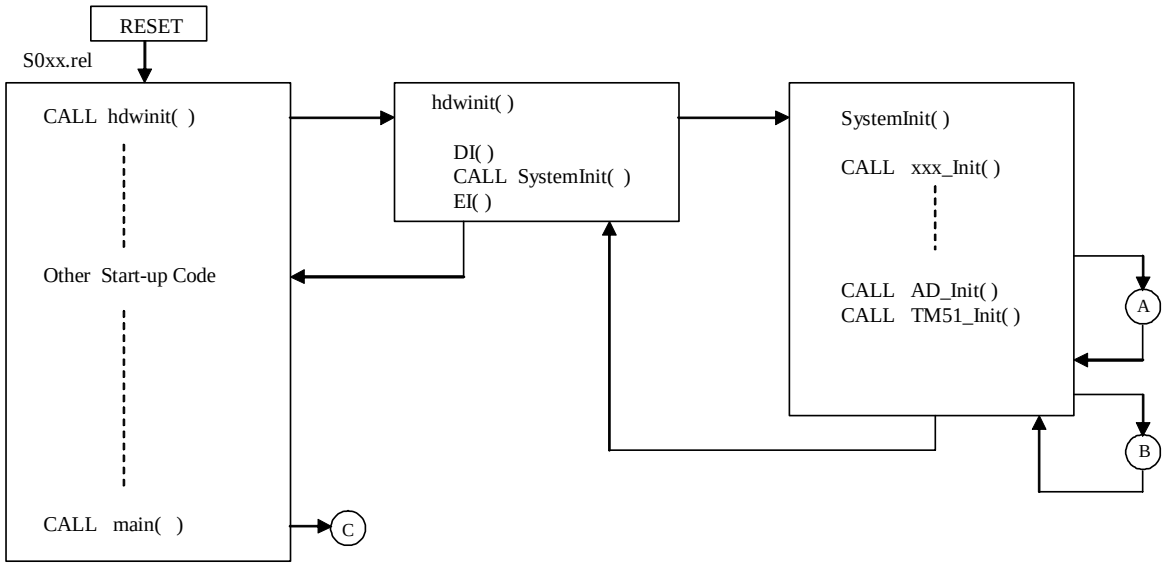
- Initialization code for the program, called before the main() program starts
- The main program loop, which checks the A/D input and changes TM51 PWM active time accordingly

- Subroutines generated by the Applilet to handle A/D, and TM51 starting, stopping and changing interval
- Subroutines for LED-display output

The flowcharts that follow describe initialization, the main program, and subroutines related to TM51.

5.3.1 Program Startup and Initialization

Figure 47. Flowchart for Program Startup and Initialization



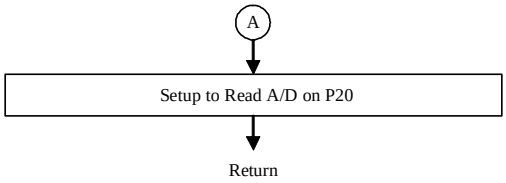
For 78K0 programs written in the C language, an object code file such as s01.rel links into the user program to provide the startup code. This startup code calls a function named hdwinit( ); you can place any hardware-specific initialization code here.

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

When hdwinit( ) finishes, the startup code calls the main( ) function of the user program. So at the start of the main( ) function, all peripherals have been initialized, and the main( ) function does not need to call individual initialization routines.

5.3.2 AD_Init( ) — Initialize A/D Converter

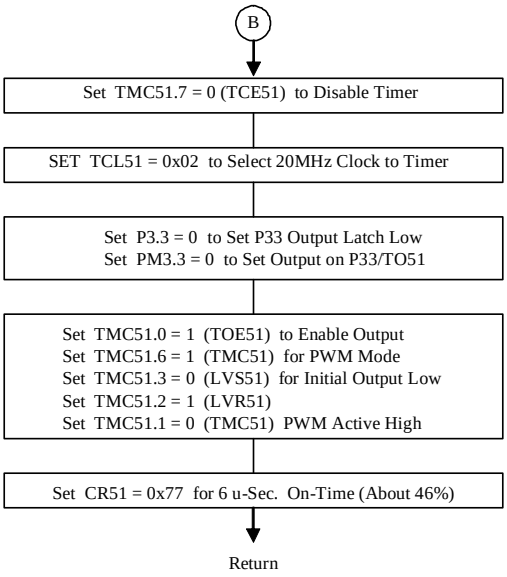
Figure 48. Initializing A/D Converter



SystemInit() calls the AD_Init() function to initialize the A/D converter and sets the converter up to read the analog-input pin ANI0/P20 .

5.3.3 TM51_Init( ) —Initialize TM51 Timer/Counter for PWM Generation

Figure 49. Flowchart for Initializing PWM Output with TM51



SystemInit() also calls the TM51_Init() function, which sets up Timer51 for PWM output. The routine sets the timer enable bit to zero to disable the timer while writing to the control registers.

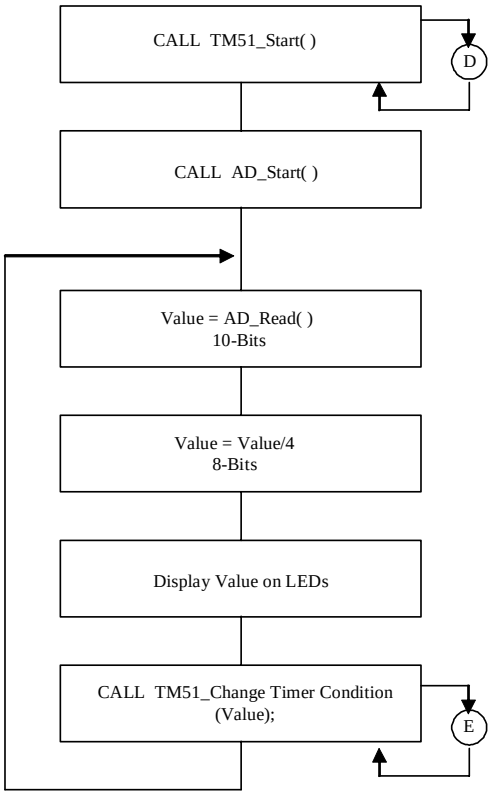
The initialization routine sets TCL51 to 0x02 – selecting fprs, the main 20-MHz clock, as the clock input. TM51 counts up every 0.05 microseconds, counting through all values every 256 counts, resulting in one cycle every 256 * 0.05 microseconds, or every 12.8 microseconds. This arrangement produces a PWM cycle frequency of 78.125 kHz.

The PWM waveform outputs on pin P33/TO51. The program sets the P3.3 output latch and the port-mode register PM3 bit 3 to zero to establish P33/TO51 as an output pin.

TM51_Init() enables the TO51 output for default output low and the PWM output in an active-high mode. The routine also sets CR51 to 0x77, or 119, which causes the output to be high for 118 pulses out of 256, or about 46%.

5.3.4 Main( ) – Main Program — D/A Conversion Using PWM

Figure 50. Main Program Flow



At the start of the program, the TM51_Start() routine initializes PWM output to the D/A converter, using an initial value in CR51 of 0x77; this value results in an initial voltage of 118/256 * 5V, or about 2.30V. The routine then starts the A/D converter by calling AD_Start().

The program repeatedly reads the A/D converter input until a 10-bit A/D value becomes available from the potentiometer. The program shortens this value to 8 bits and displays the value on the LEDs.

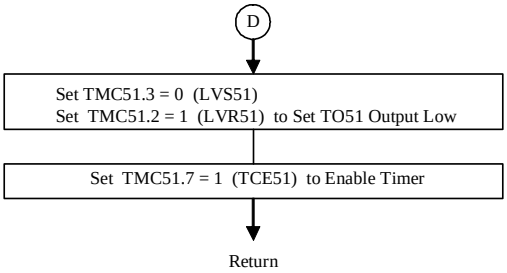
Calling the TM51_ChangeTimerCondition(value) routine sets the CR51 register to a digital value corresponding to the analog input. If this value differs from the value set before, the routine alters the duty cycle of the PWM output, with the high portion set to the value read from the A/D converter.

The A/D converter provides inputs of 00 to FF. When set in the CR51 register, these inputs result in a PWM duty cycle ranging from 0/256 to 255/256, or from 0 to 4.98V DC.

The D/A converter voltage should follow the A/D input voltage set by the potentiometer.

5.3.5 TM51_Start( ) – Start Timer51 Operation

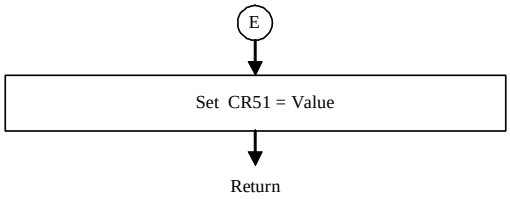
Figure 51. Flowchart for Starting Timer



The TM51_Start() routine sets the TMC51 register, which causes Timer51 to set the output low and then enables the timer.

5.3.6 TM51_ChangeTimerCondition(value) – Set CR51 Value

Figure 52. Flowchart for Changing CR51 Value



The TM51_ChangeTimerCondition(value) routine writes the passed value into the CR51 compare register, changing the duty cycle of the PWM. This operation does not require stopping the timer.

5.4 Applilet's Reference Driver

NEC Electronics' Applilet program generator automatically generates C or assembly language source code to manage peripherals for the 78K0 family of microcontrollers. Please see the Appendix for the version of the Applilet used.

The Applilet produces the basic initialization code and driver code for timer Timer51, as well as code to read the A/D converter and manage the I/O ports used for LED display. Once the Applilet produces the basic code, you can add additional code to customize the program.

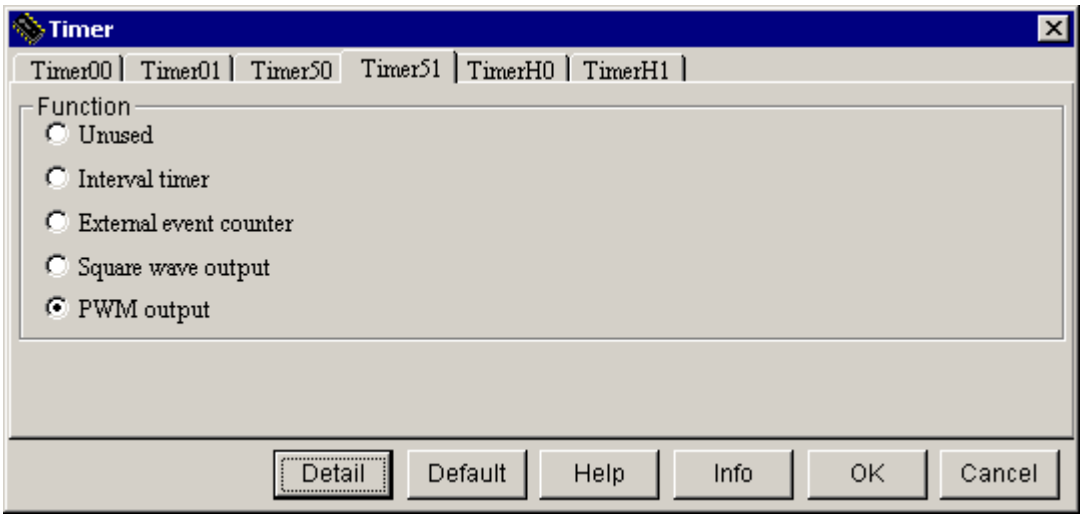
This section describes how the Applilet is set up to produce code for Timer51 and lists the routines produced.

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

5.4.1 Configuring Applilet for Timer51

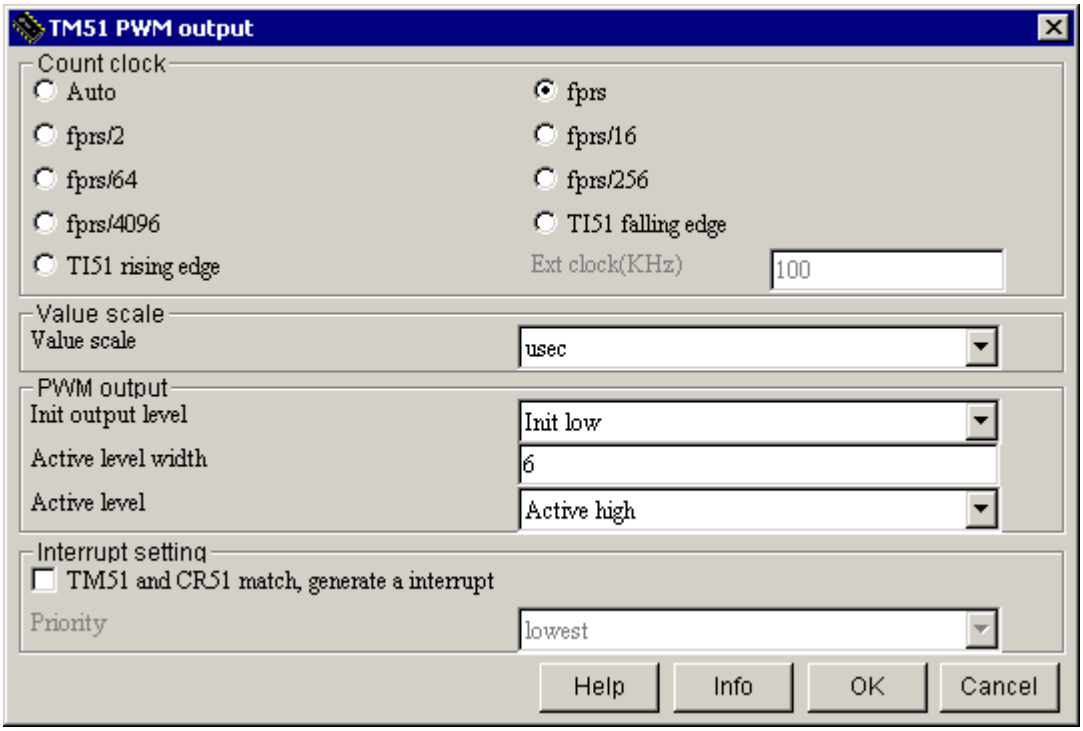
Selecting **Timer** brings up a dialog box showing the various timer blocks. Select **Timer51** and click **PWM output**.

Figure 53. Applilet Peripheral-Selection Dialog Box



Once you have selected **Timer51** for PWM output, click **Detail** to bring up the PWM output dialog box.

Figure 54. Applilet PWM Output Dialog Box



This dialog box lets you select operation details for **Timer51**. Under **Count clock**, select the fprs clock input. This choice provides the highest possible PWM frequency. An fprs of 20 MHz results in a 78.125-kHz PWM frequency, with each cycle 12.8 microseconds long.

Set **PWM output** to **Init low** (initially low), and set **Active level width** to **6** microseconds to provide a 6/12.8 or about 46% high time for the initial setting. Since the PWM width changes to match the analog input, the initial value does not really matter.

Do not click the **Interrupt setting**.

5.4.2 Generating Code with Applilet

Once you have set up the timer dialog box, select **Generate code** from the Applilet's main dialog box. The Applilet displays the peripherals and functions and lets you select a directory for the source code. Then, when you click **Generate**, the Applilet creates the initialization code, interrupt service routines, and driver subroutines.

The Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and displays a dialog box with the list of files created, including timer.h, timer.c and timer_user.c, as well as several others not related to the timer.

5.4.3 Applilet-Generated Timer51 Files and Functions

The code for Timer51 support is in the files timer.h, timer.c and timer_user.c.

5.4.3.1 Timer.h

The header file timer.h contains declarations for the functions controlling Timer51 and definitions of values for Timer51 initialization. The header file macrodriver.h, used for all Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

5.4.3.2 Timer.c

The source file Timer.c contains the following functions:

void TM51_Init(void)
The TM51_Init() routine initializes the Timer51 peripheral.

void TM51_Start(void)
The TM51_Start() routine enables the timer.

void TM51_Stop(void)
The TM51_Stop() routine stops timer operation by disabling the timer. The demonstration program does not use this routine.

MD_STATUS TM51_ChangeTimerCondition(UCHAR value)
The TM51_ChangeTimerCondition() function changes the value in compare register CR51, thereby changing the PWM high time and duty cycle.

5.4.3.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. These functions are empty on code generation, so you can add application-specific code. The demonstration program does not use this routine, so timer_user.c contains no code.

5.4.4 Applilet-Generated Files Not Related to Timer51

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

Table 12. Files Not Related to Timer51

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
Ad.h	Defines for the A/D converter
Ad.c	A/D converter routines
Ad_user.c	User A/D converter routines
System.h	Clock-related definitions
System.c	Clock_Init() function
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

5.4.5 Demonstration Program Files Not Generated by Applilet

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

Table 13. Files Used by Demonstration Not Produced by Applilet

File	Function
Led_0537.h	Header file for seven-segment LED patterns and functions
Led_0537.c	Code to display data in seven-segment LEDs

5.5 Demonstration Platform

The demonstration platform for the D/A-conversion PWM output is 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.

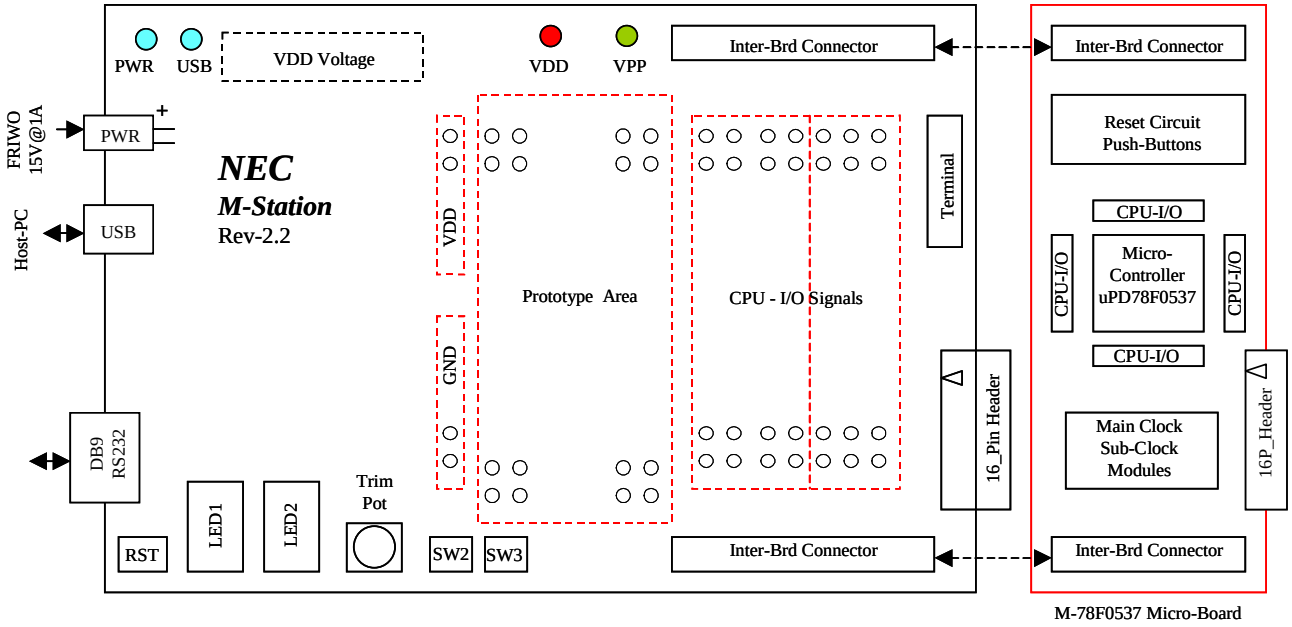
5.5.1 Resources

The demonstration uses:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- M-Station II Evaluation System, using M-Station II resources:
  - 7-Segment LEDs LED1 and LED2
  - Potentiometer to provide variable analog input voltage to P20/ANI0
  - Custom hardware added to implement low-pass filter

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

Figure 55. Demonstration Hardware Platform



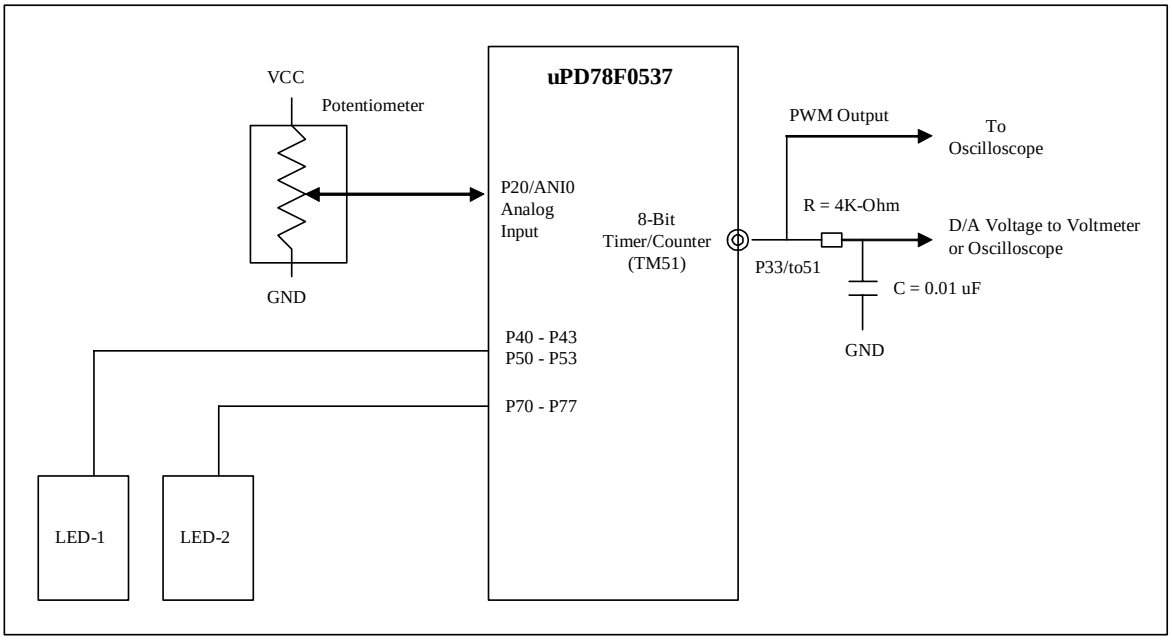
5.5.2 Demonstration of Program

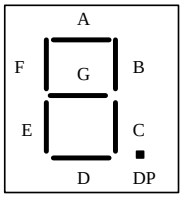
With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Turn the potentiometer and observe the input level displayed on the LEDs.
- Observe the PWM waveform on an oscilloscope.
- Observe D/A voltage using voltmeter or oscilloscope.

5.6 Hardware Block Diagram

Figure 56. Demonstration Hardware Block Diagram





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

Selecting the timer/counter clock determines the fixed frequency of the PWM signal. The value stored in the compare register varies the PWM pulse width from 0% to 100%. The PWM pulse width is directly proportional to the amplitude of the analog signal. For example, a PWM signal with a 50% duty cycle is proportional to ½ VDD at the low-pass filter output.

The response time of the low-pass filter depends on the RC time constant. The RC time constant = 1/(2*Pi*Frequency). The resistor and capacitor values for the demonstration should give approximately 4-kHz bandwidth. The PWM frequency can be as high as 78 kHz, well over the low-pass filter cut-off frequency.

5.7 Software Modules

The following files make up the software modules for the demonstration program. The table below shows which files are generated by the Applilet and which of those files need modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 14. Files That Make Up Demonstration Program

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	No
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Led_0537.h	LED display definitions	--	Yes
Led_0537.c	LED display functions	--	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

6. Carrier Generation for Remote Control Using TM51 and TMH1

You can implement an infrared remote control by using microcontroller timers to generate the carrier signal. Typically, a pair of 8-bit timer/counters generates the infrared remote-control time count signal. Specifically, 8-bit timer/counter-H1 (TMH1) generates the carrier clock, which is output during the cycle set by 8-bit timer/counter-51 (TM51). The carrier clock generator (TMH1) compares the timer/counter value with the value in two compare registers to set the high- and low-level carrier-pulse waveform.

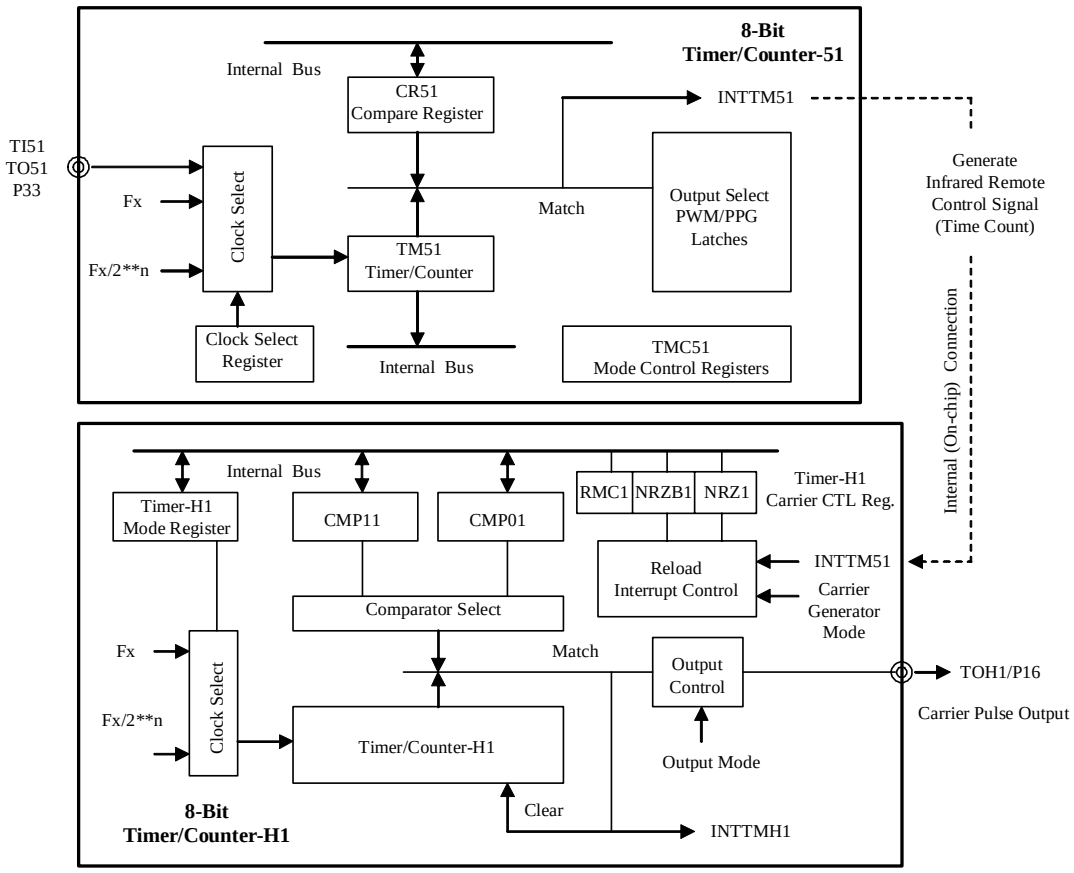
Several NEC Electronics' 78K0 microcontrollers support carrier-output generation as part of their 8-bit timer/counter functions. In addition to serving in TV or VCR remote controls, the carrier generator also serves controlling systems at remote locations.

6.1 Features of Carrier-Output Generator

The controller uses a pair of 8-bit timer/counters – timer/counter-51 and timer/counter-H1. Timer/counter-51 generates infrared remote control signals (time count), while timer/counter-H1 generates the carrier-pulse output. When used in carrier generation mode, Timers H1 and 51 offer:

- Variable carrier frequency
- Variable carrier duty cycle
- Independently variable carrier on and carrier off times
- Automatic change of carrier on/carrier off state after the time period elapses
- Automatic gating of changes to on/off state, for regular carrier pulses
- Optional interrupt on the edges of carrier pulses
- Interrupt on end of carrier on/carrier off times

Figure 57. Timer/Counter Block Diagrams



The demonstration uses these compare registers:

- CMP01 Sets low-level width of the carrier pulse waveform
- CMP11 Sets high-level width of the carrier-pulse waveform

The carrier pulse output is controlled by INTTM51, the interrupt-request signal from timer/counter-51. The output is also controlled by the RMC1, NRZB1 and NRZ1 bits of the carrier-control register:

- NRZ1 = 0 Carrier-output disable
- = 1 Carrier-output enable, when RMC1 = 1
- RMC1 = 0 NRZB1 = 0 Low-level output
- RMC1 = 0 NRZB1 = 1 High-level output
- RMC1 = 1 NRZB1 = 0 Low-level output
- RMC1 = 1 NRZB1 = 1 Carrier-pulse output

The carrier pulse generation steps include:

- Set operation registers.
  - Set CMP01 with the compare value.
  - Set CMP11 with the compare value.
- Set control register:
  - RMC1 = 1 Remote-control output enable
  - NRZB1 = 0/1 Carrier-output enable
- Set Timer/Counter-H1 and Timer/Counter-51 mode registers: timer starts counting.
- CMP01 = Timer/Counter-H1 Match interrupt INTTMH1
  - Timer/counter-H1 is cleared.
  - Switch compare register to CMP11 for next compare.
- CMP11 = Timer/counter-H1 Match Interrupt INTTMH1
  - Timer/counter-H1 is cleared.
  - Switch compare register to CMP01 for next compare.
- Repeat compare-and-clear and switch-compare-register steps to generate carrier clock.

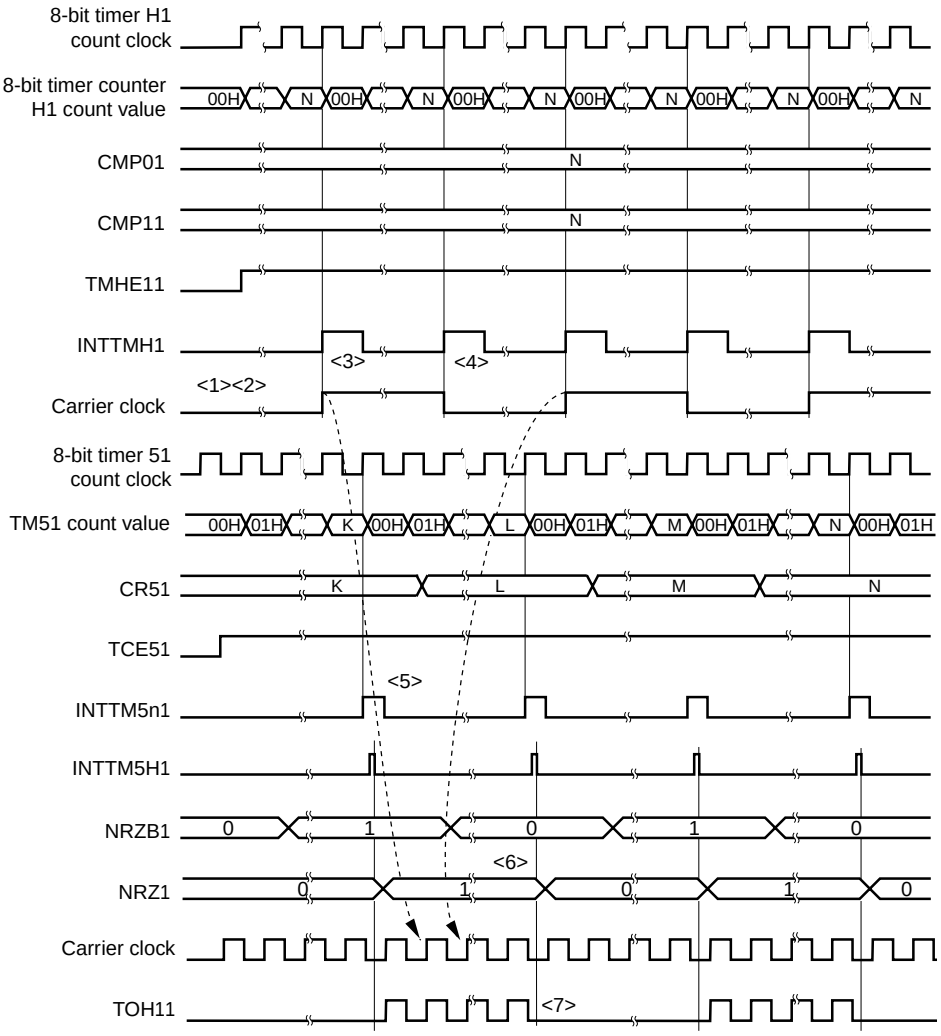
These steps put the output carrier clock on the TOH1 output:

- INTTM51 interrupt signal synchronizes with the INTTMH1 interrupt signal.
  - When INTTMH1 Transfer NRZB1 to NRZ1
  - Write next value for NRZB1
- If NRZ1 = 1, TOH1 output = carrier clock
- If NRZ1 = 0, TOH1 output = 0

The carrier clock output cycle works this way:

- If
  - CMP01 = N
  - CMP11 = M
  - Count clock frequency = Fx
- Then
  - Carrier-clock output cycle = (N + M + 2)/Fx
  - Carrier-clock duty cycle = (high-level width)/(carrier clock width) = (M + 1)/(N + M + 2)

Figure 58. Timing Diagram for Carrier Generation



To configure TimerH1 and Timer51 for carrier generation:

- Set the carrier clock, carrier mode, and output enable using TMHMD1.
- Set the priority for the carrier interrupt, clear the interrupt flag, and unmask the interrupt, if used.
- Set the output pin and latch low for carrier output.
- Set the initial on/off state and enable the carrier using TMCYC1.
- Set the CMP01 and CMP11 registers for carrier duty cycle.
- Set the interval-timer mode for TM51; set clock to TM51 using TCL51.
- Set the priority for the TM51 interrupt, clear the interrupt flag, and unmask the interrupt.
- Set the initial time to first bit using CR51.

At this point the carrier generator is ready. To start carrier generation:

- Start TimerH1 to generate the internal carrier waveform
- Start Timer51 to count down to first transition.

After this point, interrupts generated by TM51 signal the start of the next half-bit period and update the preset carrier on/off state. In the interrupt-service routine, timing and on/off state are set for the next half-bit period.

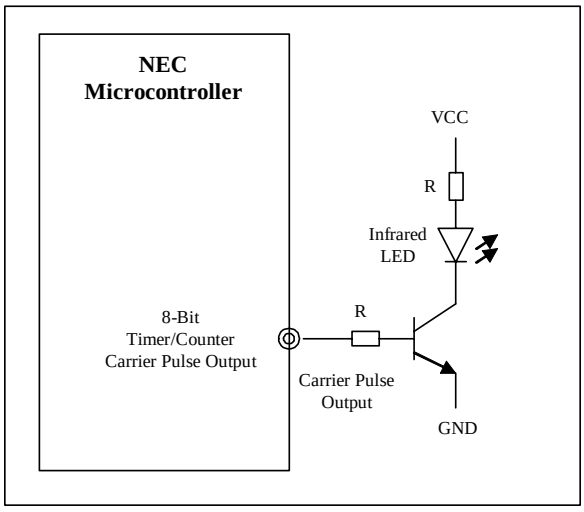
6.2 Program Description and Specification

The demonstration program generates a carrier-clock waveform by setting values in compare registers CMP01 and CMP11. Timer51 times the intervals of carrier pulses on and off. The carrier-clock output is connected to a light emitting diode for transmitting the infrared signal. You can observe the carrier-clock output, TOH1, with an oscilloscope.

In remote-control applications you would typically have a keypad attached to a 78K0-family microcontroller. The remote control unit should generate specific carrier-clock waveforms controlled by keypad entry. For example, pressing 5 on the keypad could generate a waveform equivalent to selecting channel 5 on the television set. The carrier-clock waveform must be appropriate for the selected TV set.

Although implementing a complete remote control unit is beyond the scope of this application note, the demonstration does show all of the necessary register settings and carrier-clock generation steps needed to form the basis of an actual remote control.

Figure 59. Remote Control Hardware



Specifications:

- TMH1 is set to produce carrier pulses of 8.5 microseconds high, 18 microseconds low.
- TM51 produces one and zero bits, with the following characteristics:
  - One bit = carrier pulses on for 565 microseconds, off for 565 microseconds
  - Zero bit = carrier pulses on for 565 microseconds, off for 1.685 microseconds

6.3 Software Flow Charts

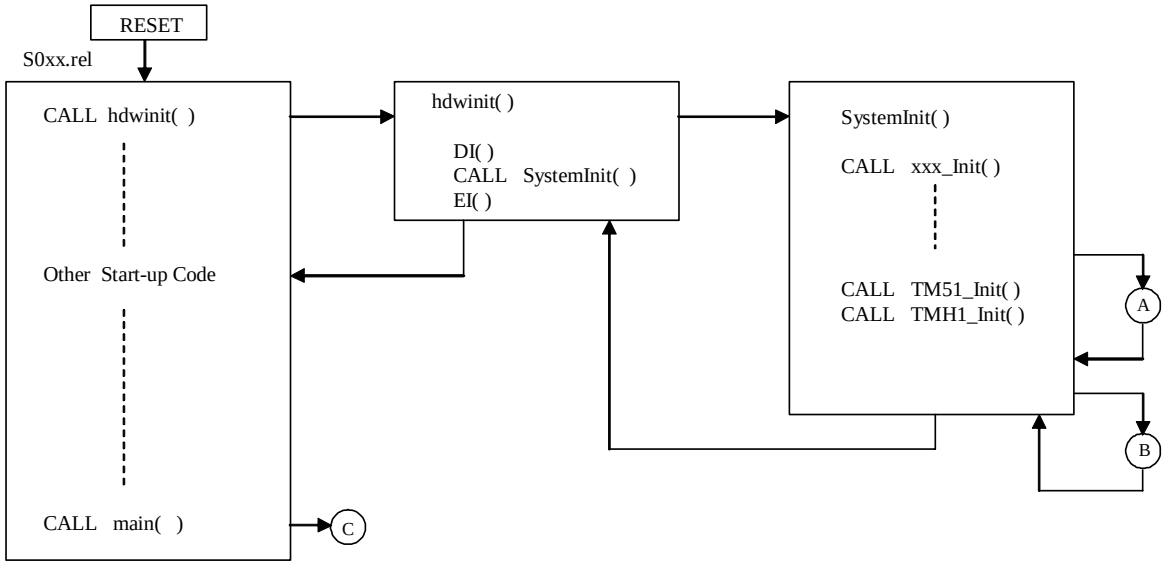
The demonstration program consists of the following major sections:

- Program initialization code — called before the main() program starts
- The main program loop, which sets up data transmission and starts sending
- Subroutines generated by the Applilet to handle TMH1 and TM51 starting, stopping and changing interval
- Subroutines with user code for handling TM51 interrupts (Applilet-generated stub interrupt-service routines, with user code added)

The following flowcharts describe the initialization, the main program, subroutines related to TMH1 and TM51, and the interrupt-service routine for TM51.

6.3.1 Program Startup and Initialization

Figure 60. Flowchart for Program Startup and Initialization



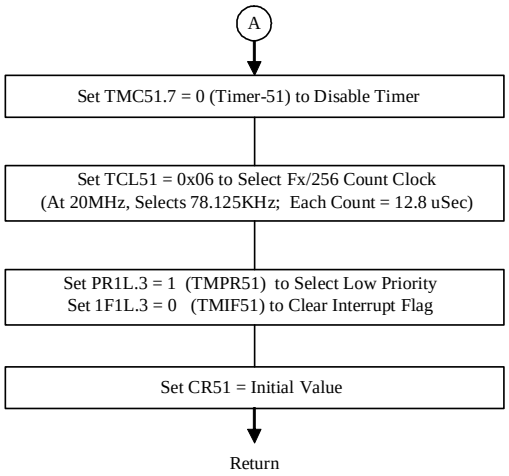
For 78K0 programs written in the C language, an object code file such as s01.rel links into the user program to provide the startup code. This startup code calls a function named hdwinit( ); you can place any hardware-specific initialization code here.

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

When hdwinit( ) finishes, the startup code calls the main( ) function of the user program. So at the start of the main( ) function, all peripherals have been initialized, and the main( ) function does not need to call individual initialization routines.

6.3.2 TM51_Init( ) – Timer51 Initialization

Figure 61. Initialization Flow Chart for Timer51



SystemInit() calls the TM51_Init() function, which initializes Timer51 for interval-timer operation. Timer51 times the high (carrier pulses on) and low (carrier pulses off) times of bits, which determine whether a bit is transmitted as a one or a zero.

TMC51 bit 7 associated with TME51 is set to zero to disable the timer while writing control registers.

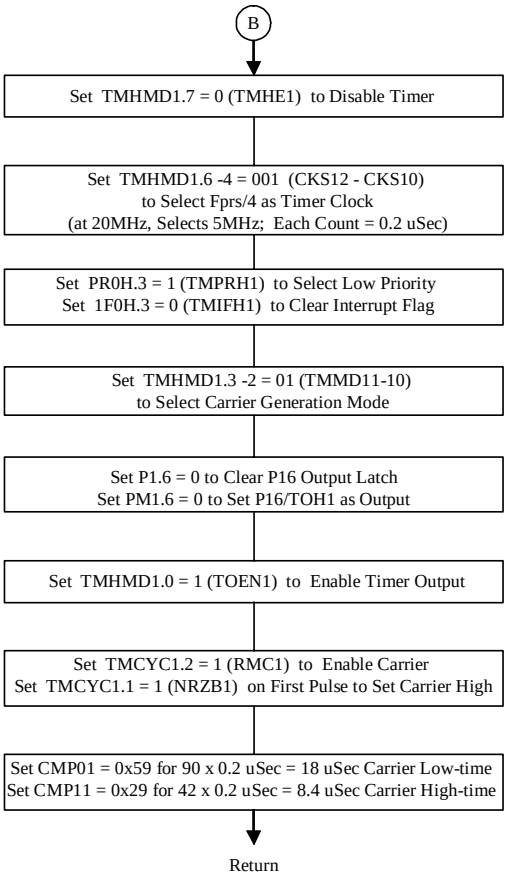
TCL51 is set to 0x06 to select fprs/256 as the TM51 count clock. At the main system clock frequency of 20 MHz, this selection provides in 20,000,000/256 or 78.126 kHz as the TM51 count clock. Each count to TM51 is 12.8 microseconds.

The interrupt-control registers for Timer51 are set for low priority, and the interrupt flag is cleared. The interrupt mask is left in the default disabled state.

Although CR51 is set for an initial value, the value is overwritten before Timer51 is used.

6.3.3 TMH1_Init( ) – TimerH1 Initialization

Figure 62. Initialization Flow Chart for TimerH1



SystemInit() calls TMH1_Init() to initialize TimerH1 for carrier generation. The routine disables the timer before writing to the control registers.

TMHMD1 is set to select fprs/4 as the timer clock. At 20 MHz, this selection results in a 5 MHz clock; TMH1 thus counts up every 0.2 microseconds.

The interrupt register controlling TimerH1 is set for low priority, and the interrupt flag is cleared. An interrupt is not needed here because this timer simply creates the carrier-pulse waveform.

The demonstration uses P16/TOH1 as the carrier output pin. Setting the P1.6 output latch and the port-mode register PM1 (bit 6) to zero configures P16/TOH1 as an output. Setting TMHMD1 bit zero (TOEN1) to 1 enables the timer output.

Setting TMCYC1 bit 2 (RMC1) to 1 enables carrier generation, and setting the NRZB1 bit to 1 turns the carrier pulses on for the first pulse.

Setting the CMP01 register to 0x59 produces a 90 * 0.2 microseconds carrier low time, or 18 microseconds. CMP11 is set to 0x29 for 42 * 0.2, or 8.4 microseconds carrier high time. The resulting carrier waveform has a duty cycle consisting of 1/3 high and 2/3 low.

6.3.4 Main( ) – Main Program — Carrier Pulse Generation

Figure 63. Main Program Flow

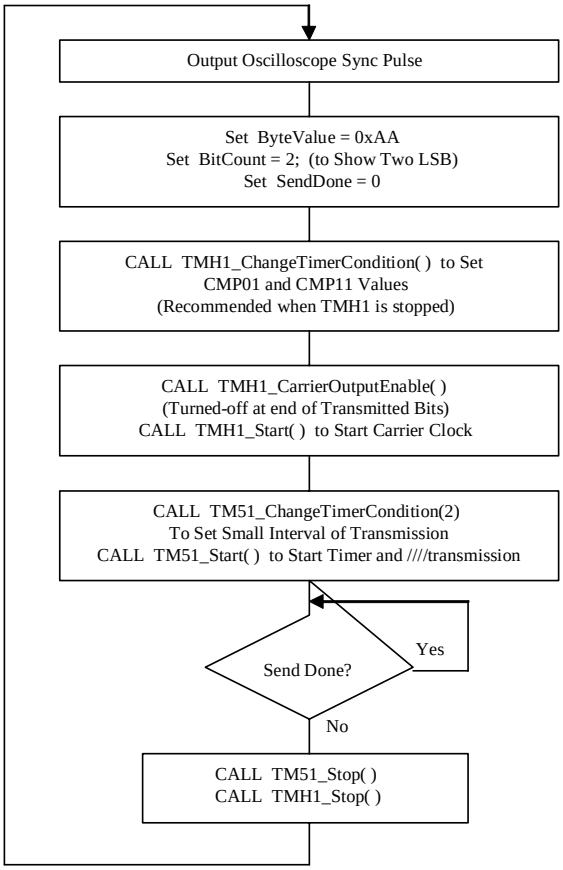
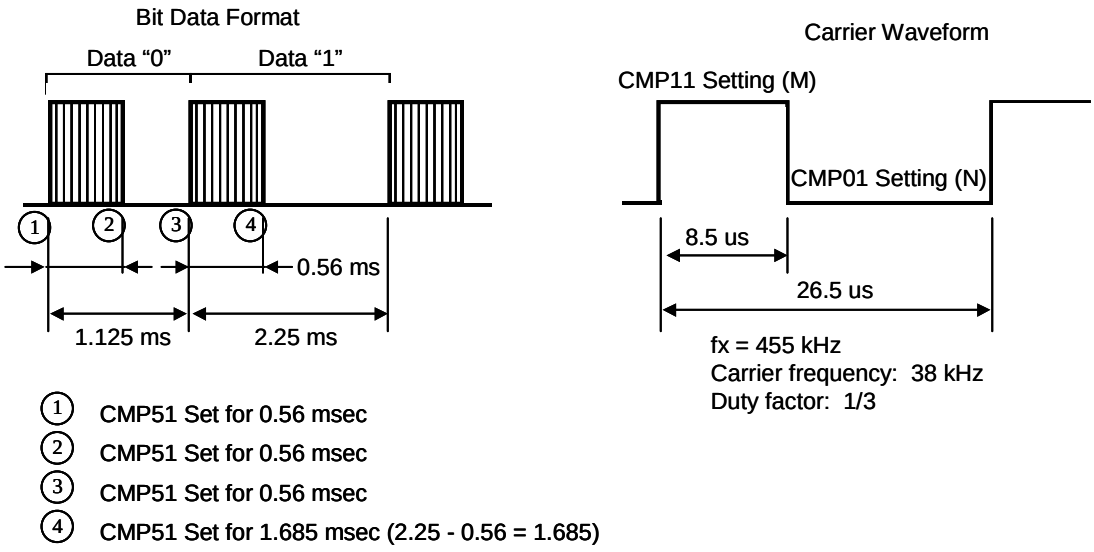


Figure 64. Carrier Waveform Generated by Demonstration Program



① CMP51 Set for 0.56 msec
② CMP51 Set for 0.56 msec
③ CMP51 Set for 0.56 msec
④ CMP51 Set for 1.685 msec (2.25 - 0.56 = 1.685)

The main program generates a waveform corresponding to two bits of the data 0xAA (10101010) sent out LSB first. To enable observation on an oscilloscope, the program generates a sync-bit pulse on port P77. The program sets the data to be sent as 0xAA, the bit count to 2, and the SendDone flag to zero.

The routine TMH1_ChangeTimerCondition(*array, 2) writes the values into CMP01 and CMP11 to determine the carrier-waveform timing. These values do not change, but you should rewrite them if TimerH1 is stopped.

TMH1_CarrierOutputEnable() sets the carrier output on, and then TMH1_Start() is called to start the TimerH1 timer. At this point, TimerH1 is generating an internal waveform, but since the NRZ1 bit is still zero, no carrier pulses are output.

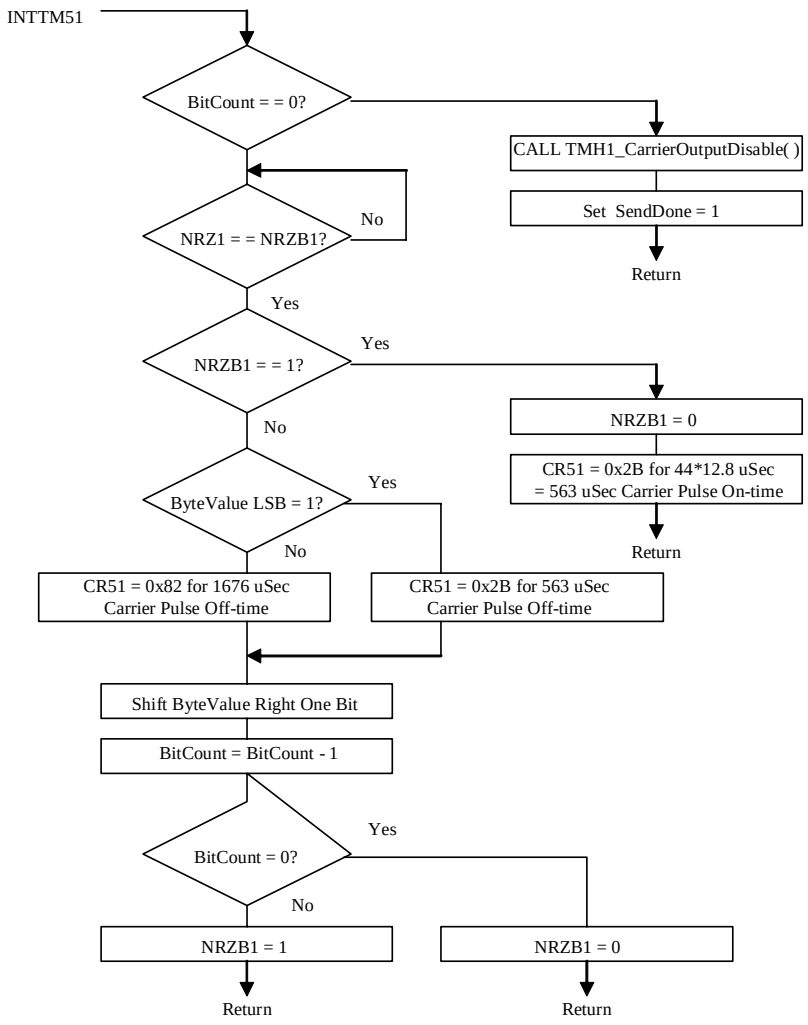
TM51_ChangeTimerCondition(2) puts the value 2 in CR51, which results in a short time (3 * 12.8, or 38.4 microseconds) before the first Timer51 interrupt INTTM51 occurs. TM51_Start() starts Timer51.

The main program waits until it sees that the SendDone flag is no longer zero. The program then stops the timers by calling TM51_Stop() and TMH1_Stop(). At this point, the two bits have been sent.

All of the actual work in sending the bits is done in the interrupt service routine MD_INTTM51, which is invoked when the INTTM51 interrupt occurs.

6.3.5 MD_INTTM51( ) – Interrupt-Service Routine for INTTM51

Figure 65. Interrupt-Service Flow



INTTM51 is normally generated when TM51 and CR51 match. Because TimerH1 is in the carrier generation mode, however, the INTTM51 interrupt changes to the INTTM5H1 interrupt. The latter interrupt is held off until the next rising edge of the TimerH1 count clock to synchronize the interrupt with the clocks. This approach avoids carrier-pulse truncation.

When INTTM5H1 occurs, the value in the NRZB1 bit (initially 1) transfers to the NRZ1 bit, and carrier pulses appear on P16/TOH1.

After the last bit-low portion of the carrier has been sent, the interrupt-service routine checks the BitCount variable. If this variable is zero, sending is complete, so the interrupt-service routine calls the TMH1_CarrierOutputDisable() function and sets the SendDone variable to 1 to signal the main program.

If the bits have not all been sent, however, the interrupt-service routine waits until the value of NRZ1 equals NRZB1. The routine then decides how to set the NRZB1 bit for the next transition, depending on the current state.

If NRZB1 is high, the transmission is starting on the high portion of a bit, which produces carrier pulses for 563 microseconds. NRZB1 is set to zero to change NRZ1 to zero on the next transition, and CR51 is set to 0x2B for 563 microseconds delay until the next interrupt. At this point the interrupt-service routine returns to the main program.

If NRZB1 is low, the transmission is starting on the low portion of a bit. If the current LSB is 1, the low time is short (563 microseconds). If the current LSB is 0, the low time is long (1676 microsecond). The routine sets CR51 with appropriate values.

The current byte value shifts down one bit, to prepare for sending the next LSB, and the BitCount variable counts down. If this is not the last bit (BitCount does not equal zero), then NRZB1 is set to 1 so that NRZ1 will go to 1 on the next interrupt and begin carrier pulses for the high portion of the next bit.

If this is the last bit, NRZB1 is left as zero. On the next interrupt, NRZ1 does not go to 1, and no carrier pulses are output. When this next interrupt sees BitCount as zero, the service routine disables the carrier transmission and sets SendData to 1.

6.4 Applilet's Reference Drivers

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

The Applilet produces the basic initialization code and driver code for TimerH1 and Timer51. After the Applilet produces the basic code, you can add additional code to customize the program.

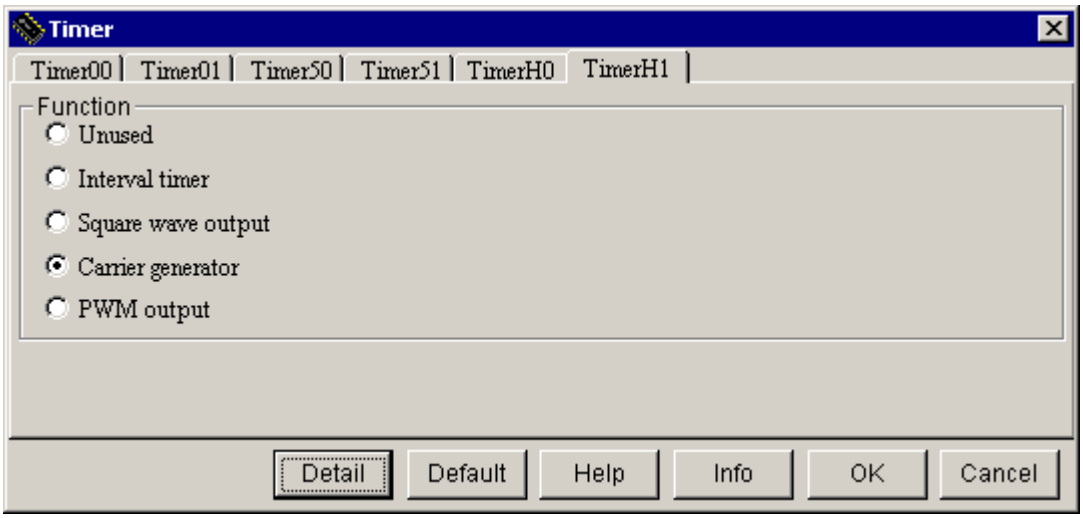
This section describes how to set up the Applilet to produce the code for configuring TimerH1 and Timer51 for carrier generation and lists the routines produced.

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

6.4.1 Configuring Applilet for TimerH1 Carrier Generation

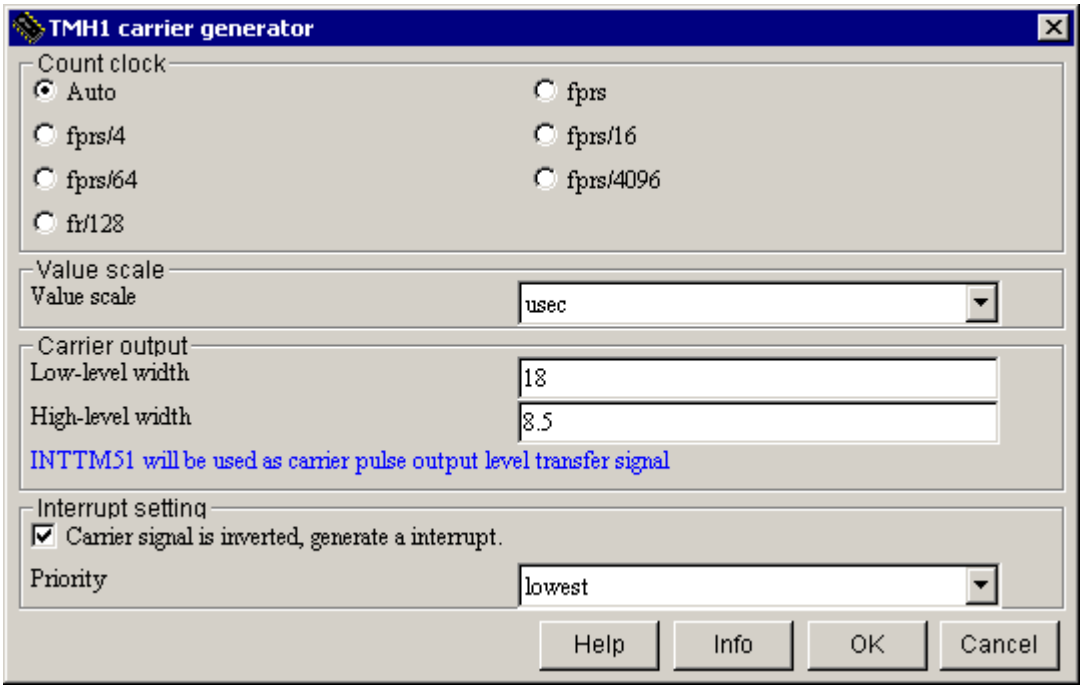
Selecting **Timer** brings up a dialog box showing the various timer blocks. Select **TimerH1** and click **Carrier generator**.

Figure 66. Applilet Timer Setup Dialog Box



Once you select **TimerH1** the **Detail** button is available for settings in the selected mode. Clicking **Detail** brings up the **TimerH1 carrier generator** dialog box.

Figure 67. TMH1 Carrier-Generator Dialog Box



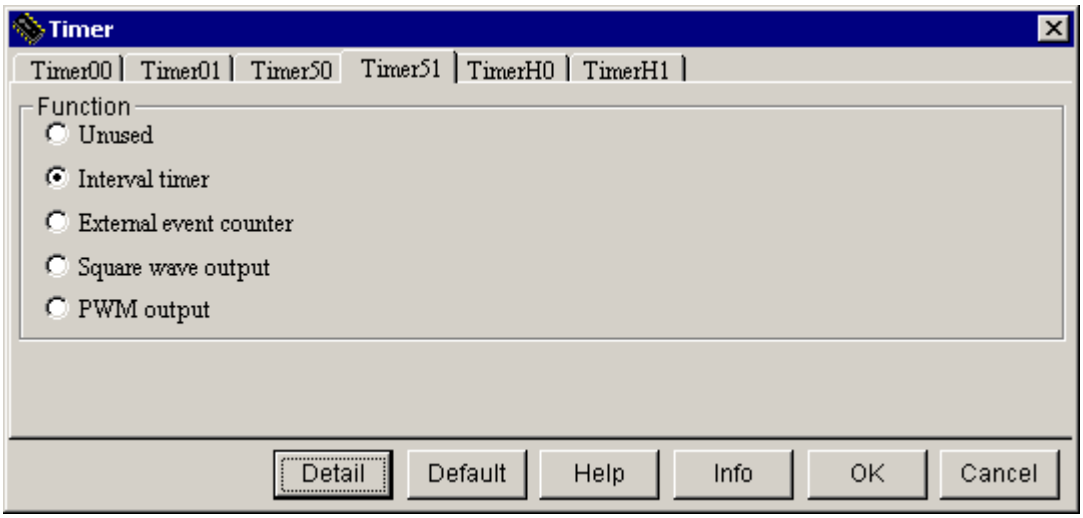
This dialog allows you to select operation details for TimerH1. The timer should generate a carrier waveform with an 8.5-microsecond high portion and an 18-microsecond low portion. Set **Value scale** to microseconds, and enter the appropriate values in the **Low-level width** and **High-level width** boxes. Set **Count clock** to **Auto** so that the Applilet can select clock divider values to produce the desired widths.

You do not need to generate an interrupt on carrier inversion, but click **Interrupt setting** anyway. The interrupt-service routine could be used to invert the output bit so that you can observe the internal carrier waveform.

6.4.2 Configuring Applilet for Timer51

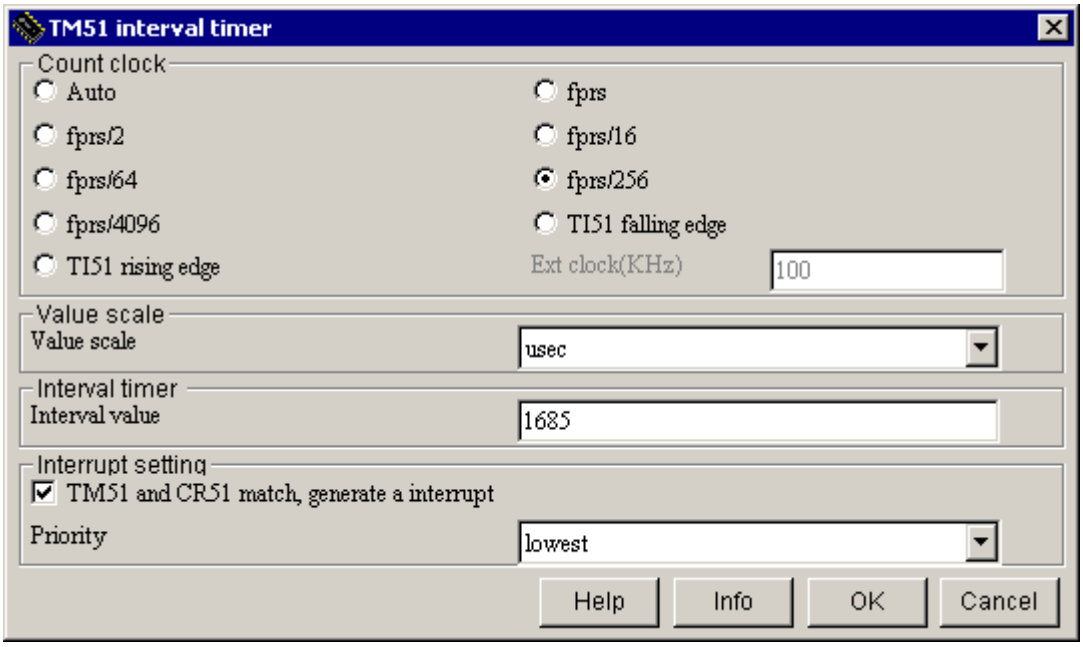
Selecting **Timer** and then the **Timer51** tab shows you this timer's available settings. Select **Interval timer**.

Figure 68. Configuring Timer51 with Applilet



Once you have selected **Timer51** the **Detail** button is available for settings in the selected mode. Clicking **Detail** brings up the **Timer51 interval-timer** dialog box.

Figure 69. Applilet Interval-Timer Dialog Box



The timer should generate intervals ranging from 563 to 1685 microseconds. Set the larger value in **Interval value**, and set **Count clock** to fprs/256. This clock selection produces a count clock of 20,000,000/256, or 78125 Hz — one clock every 12.8 microseconds.

Click **Interrupt setting** to generate an interrupt when TM51 (the Timer51 timer-count register) and CR51 (the compare register) match. Set the interrupt **Priority** to **lowest**.

6.4.3 Generating Code with Applilet

Once you complete the timer dialog boxes, select **Generate code** from the Applilet's main dialog box. The Applilet displays the peripherals and functions and lets you select a directory for the source code. Clicking **Generate** causes the Applilet to generate the initialization code, interrupt service routines, and driver subroutines.

Clicking **Generate** creates code in several C-language source files (extension .c) and header files (extension .h), and then displays a dialog box with the list of files created. The Applilet generates timer.h, timer.c and timer_user.c, as well as several other subroutines not related to the timers.

6.4.4 Applilet-Generated TimerH1 and Timer51 Files and Functions

The code for TimerH1 and Timer51 support is in the files timer.h, timer.c and timer_user.c.

6.4.4.1 Timer.h

The header file timer.h contains declarations for the functions controlling timers H1 and 51, and definitions of values for their initialization. All of the Applilet-generated code uses the header file macrodriver.h, which also defines some data types and values, such as the MD_STATUS values returned by some functions.

6.4.4.2 Timer.c

The source file Timer.c contains the following functions:

void TMH1_Init(void)
The TMH1_Init() routine initializes TimerH1 as specified in the Applilet TimerH1 detail dialog.

void TMH1_Start(void)
The TMH1_Start() routine starts TimerH1 by enabling the timer and the INTTMH1 interrupt.

void TMH1_Stop(void)
The TMH1_Stop() routine stops TimerH1 by disabling the timer and the interrupt.

MD_STATUS TMH1_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
The TMH1_ChangeTimerCondition() function changes the value in compare registers CMP01 and CMP11. This value change alters the TimerH1 carrier-generation waveform values.

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

void TMH1_CarrierOutputEnable()
The TMH1_CarrierOutputEnable() routine sets the NRZB1 bit high, which enables carrier-pulse output, depending on the state of the NRZ1 and RMC1 bits.

void TMH1_CarrierOutputDisable();
The TMH1_CarrierOutputEnable() routine sets the NRZB1 bit low, which forces the carrier output low.

void TM51_Init(void)
The TM51_Init() routine Initializes Timer51 as specified in the Applilet detail dialog.

void TM51_Start(void)
The TM51_Start() routine starts Timer51 by enabling the timer and the timer interrupt.

void TM51_Stop(void)

The TM51_Stop() routine stops Timer51 by disabling the timer and the interrupt.

MD_STATUS TM51_ChangeTimerCondition(UCHAR value)

The TM51_ChangeTimerCondition() function changes the value in compare register CR51, thereby altering the interval time.

6.4.4.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. These functions are empty on code generation so you can add application-specific code.

__interrupt void MD_INTTM51( )

This is the interrupt service routine for Timer51 interrupt INTTM51, which is generated when the values in TM51 and CR51 match. Once the timer starts, the interval specified by CR51 determines when the interrupt occurs.

The Applilet leaves this interrupt-service routine blank. You must add code to handle the carrier-generation algorithm.

__interrupt void MD_INTTMH1( )

This is the interrupt service routine for interrupt INTTMH1, which is generated when the carrier waveform is inverted. The Applilet leaves this routine blank; it is not necessary to add any code.

6.4.5 Applilet-Generated Files Not Related to TimerH1 and Timer51

For the demonstration program, the Applilet generates several additional source files, as listed in the table.

Table 15. Files Not Related to TimerH1 and Timer51

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

6.4.6 Demonstration Program Files Not Generated by Applilet

For this demonstration, you need no files other than those generated by the Applilet.

6.5 Demonstration Platform

The demonstration platform for the remote control carrier generator is 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.

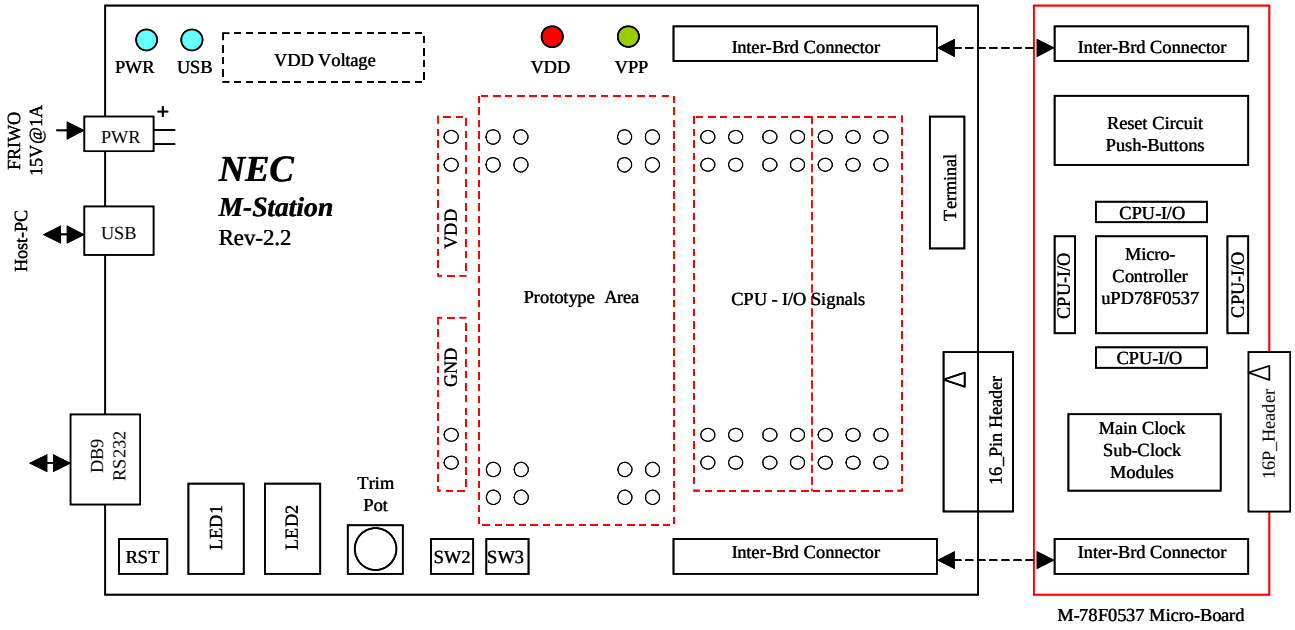
6.5.1 Resources

The demonstration program requires:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- M-Station II Evaluation System, using M-Station II resources:
  - Custom hardware added for infrared light-emitting diode

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

Figure 70. Demonstration Hardware



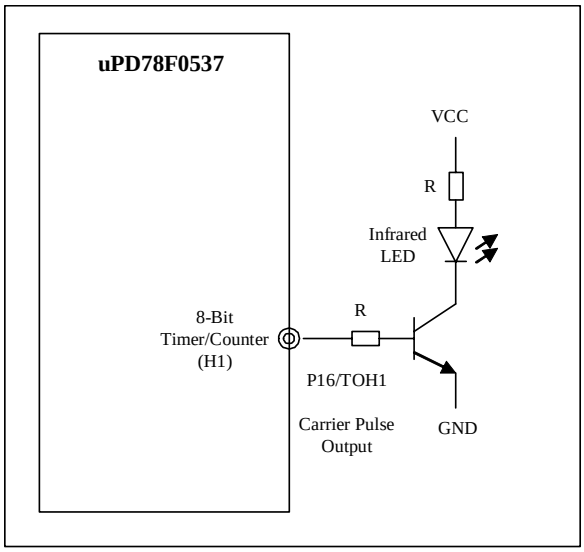
6.5.2 Demonstration of Program

With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Observe the carrier waveform on an oscilloscope
- You can also observe the lighting of the infrared LED

6.6 Hardware Block Diagram

Figure 71. Demonstration Hardware Block Diagram



For most common infrared remote control devices, you can transmit data with an infrared LED connected to the carrier-pulse output of a 78K0-family microcontroller.

This demonstration uses the infrared LED to show the transmitted carrier-clock pulse. You can see a more accurate waveform with an oscilloscope. The transmitted infrared signal can control devices such as TVs, VCRs, and CD players using each device's specific encoding formats, protocol and data. The demonstration program generates an arbitrary waveform.

6.7 Software Modules

The following files make up the software modules for the demonstration program. The table below shows which files are generated by the Applilet and which of those files need modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 16. Demonstration Program Software Modules

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

7. PPG Output-Tone Generation Using 16-Bit Timer 00 (TM00)

The programmable pulse generation (PPG) output is available for most 8- and 16-bit timer/counters in the 78K0 family of microcontrollers. This output can provide a precise clock for peripherals. The PPG output shares resources with other timer functions such as PWM, one-shot output and square-wave output.

The program described here demonstrates tone generation with the PPG 16-bit timer/counter output driving a speaker. However, because the PPG function shares an output with the PWM, one-shot and square-wave functions, you can use the same circuit to create other tones simply by using these other timer functions.

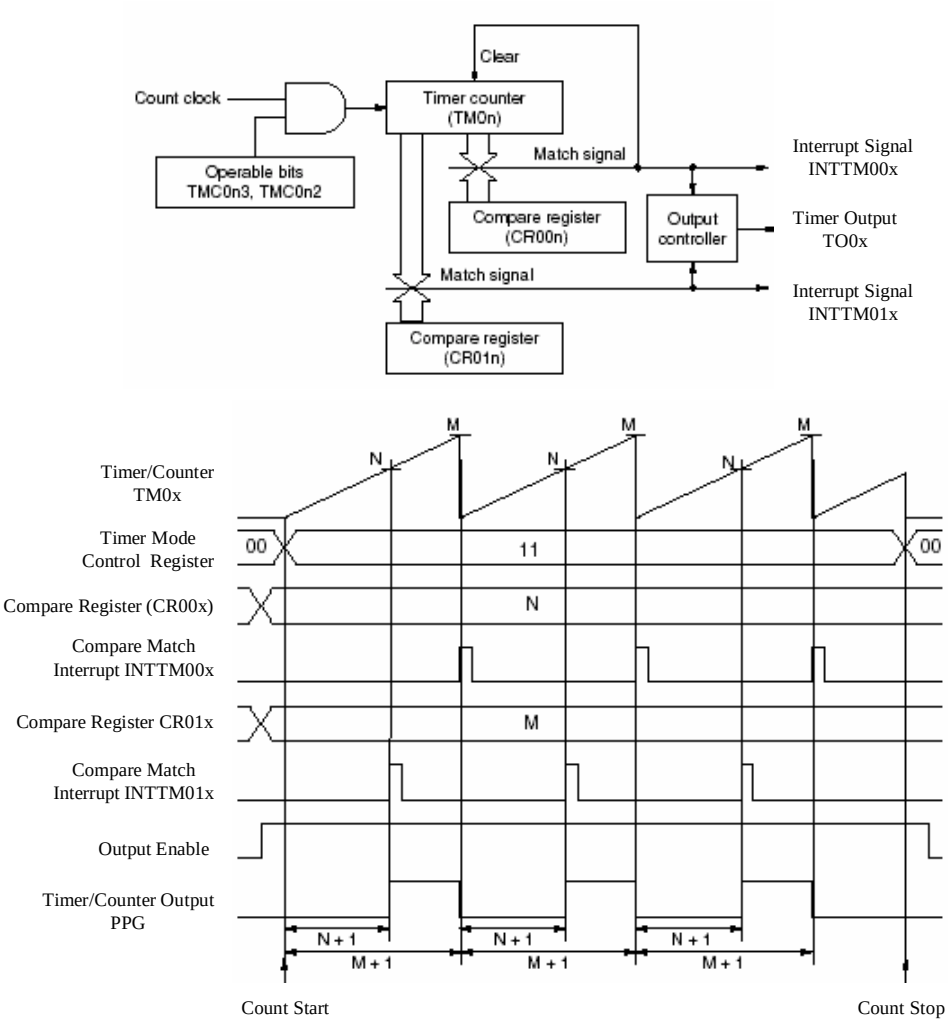
7.1 Features of PPG Output

In PPG mode, the 16-bit timer offers the following features:

- Variable time base, based on divisions of the peripheral clock, using the PRM00 register
- Variable PPG cycle time, using the CR010 register
- Variable PPG duty cycle, using the CR000 register
- Settable initial output level, active-high or active-low output
- Optional interrupts on cycle end or active time end

The circuit outputs a waveform from the timer output pin (TO00) with the pulse width set by a compare register (CR010). When you set the timer-mode control register to its clear-and-start mode, the pulse width value loads into another compare register (CR000)

Figure 72. Using PPG Output



PPG Pulse Cycle = (M + 1) * Count Clock Cycle

PPG Duty Cycle = (N + 1)/(M = 1)

To configure the timer for PPG operations, follow these steps:

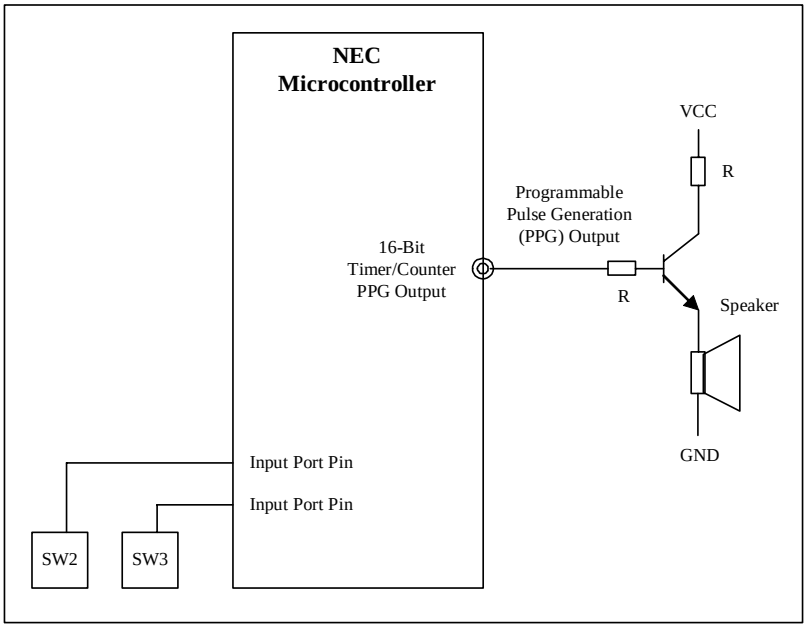
- Set the time base using PRM00.
- Set CR000 and CR010 as compare registers using CRC00.
- Set the cycle time in CR010.
- Set the active time in CR000.
- Set the output port latch and mode register bit low for PPG output.
- Set the priority for the interrupts, clear the interrupt flags, and unmask the interrupts, if used.
- Set the TOC00 register for initial output level, enable output, and invert on CR000/CR010 match.
- Enable the timer by setting the timer mode register TMC00 for clear-and-start mode on compare of TM00 and CR000.

7.2 Program Description and Specification

With the 16-bit timer/counter configured to generate PPG output and a speaker attached to the output, the demonstration program generates a sound with overtones.

Initially, the PPG output results in a 586-Hz tone with overtones of 440 and 880 Hz. Pressing and releasing switch SW2 generates a 528-Hz tone with overtones of 440 and 660 Hz. Pressing and releasing switch SW3 generates a 754-Hz tone with overtones of 660 and 880 Hz.

Figure 73. Microcontroller Drives Speaker from PPG Output



Specifications:

- TM00 is set for PPG mode, active high, initial output low, and no interrupt.
- TM00 time base clock is set to 5 MHz.
- Tones generated are:
  - 1704 µs (586 Hz), 66.7/33.3% duty cycle (440 and 880 Hz)
  - 1893.4 µs (528 Hz), 60/40% duty cycle (440 and 660 Hz)
  - 1325.4 µs (754 Hz), 42.8/57.2% duty cycle (880 and 660 Hz)

7.3 Software Flow Charts

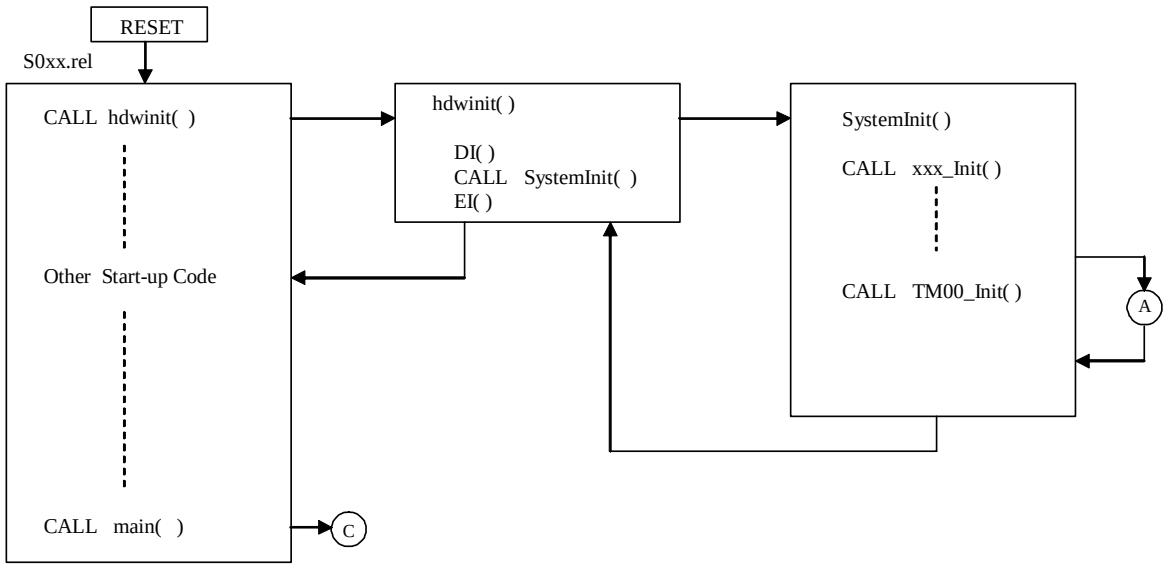
The demonstration program consists of the following major sections:

- Program initialization code, which is called before the main() program starts
- The main program loop, which checks the status of switches and changes the timing of TM00
- Subroutines generated by the Applilet to handle TM00 starting, stopping, and changing interval
- Subroutines for pushbutton input and LED display output

The following flowcharts describe the initialization, the main program, and the subroutines related to TM00.

7.3.1 Program Startup and Initialization

Figure 74. Program Startup and Initialization Flow



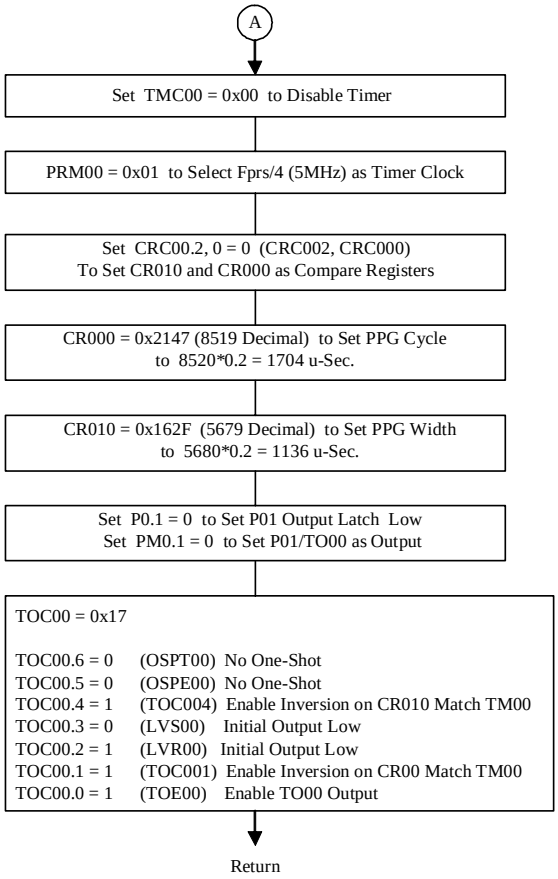
For 78K0 programs written in the C language, the initialization code is in an object code file such as s01.rel, which you link to the user program. This code calls a function named hdwinit( ) where you can place any hardware initialization code that you need.

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

When the hdwinit( ) function is done, the initialization code then calls the main( ) function of the user program. This way, by the time the main( ) function is called, all peripheral initialization is complete, and the main( ) function does not need to call any individual peripheral initialization routines.

7.3.2 TM00_Init( ) – Timer 00 Initialization for PPG Output

Figure 75. Initializing Timer for PPG Output



SystemInit() calls the TM00_Init() routine, which initializes Timer 00 for PPG output. The routine disables the timer while writing to the control registers.

The prescaler mode register PRM00 is set to 0x01 to select fprs/4 as the Timer 00 clock. With a main system clock of 20 MHz, this setting results in a 5-MHz timer clock; each count takes 0.2 microseconds.

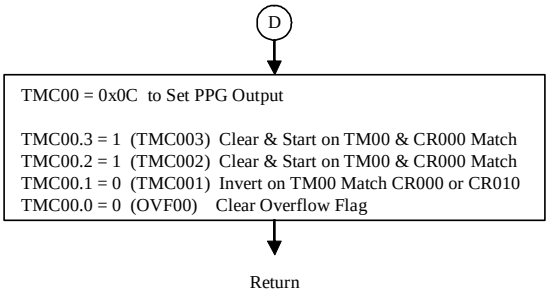
The program sets CRC00 to use CR010 and CR000 as compare registers. The program sets CR000 to 0x2147 (8519) to provide a PPG cycle time of 8520 * 0x2 = 1704 microseconds for 586 Hz. CR010 is set to 0x162F (5679) to set the PPG width to 5680 * 0.2 = 1136 microseconds. These values result in a PPG waveform that is low for 1136 microseconds, then high for (1704 – 1136) 568 microseconds. These alternating values result in a perceived tone of 586 Hz, with overtone frequencies related to the two periods of 568 microseconds and 1136 microseconds, or about 880 and 440 Hz, respectively.

Because the demonstration program uses Timer 00, output TO00 is used on pin P01/TO00. To use this output, the program sets both the P0.1 output latch and the port mode register PM0 bit 1 to zero.

The TOC00 register is set to have the initial timer output low, so the output can be inverted when both CR000 and CR010 match TM00, thus enabling the TO00 output.

7.3.3 TM00_Start( ) – Start Timer 00 Operation

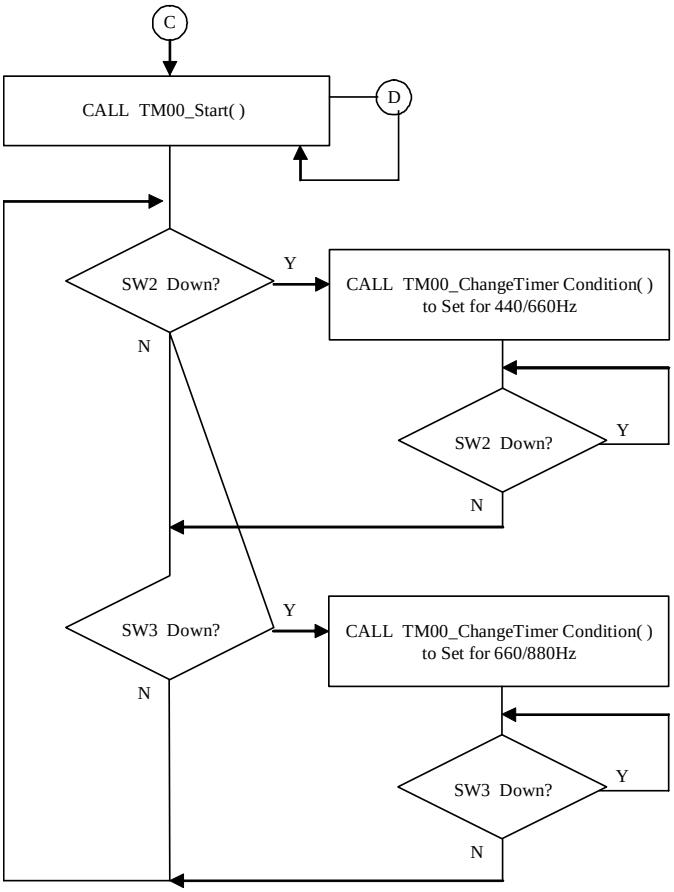
Figure 76. Starting Timer



The TM00_Start() routine sets TMC00 for PPG output, and clears and starts the timer.

7.3.4 Main( ) –Main Program – Generating Two Tones

Figure 77. Main Program Using PPG to Generate Two Tones



The main program calls TM00_Start() to enable the timer and start PPG tone generation using the default values stored in CR000 and CR010.

The program then checks to see if either switch SW2 or SW3 is down (closed). If SW2 is closed, the program calls TM00_ChangeTimerCondition() to change the CR000 and CR010 registers, which in turn change the PPG timing. This change produces a 528-Hz output, with two overtones: 440 and 660 Hz. The program then waits for SW2 to open before proceeding.

If SW3 is down, the program calls TM00_ChangeTimerCondition() to set CR000 and CR010 for tone generation of 754 Hz with overtones of 660 and 880 Hz. The program then waits for SW3 to be up. This approach allows the program to switch between the two sets of two tones. You can alter the program, putting different values in CR000 and CR010 for other tone combinations.

7.4 Applilet's Reference Driver

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

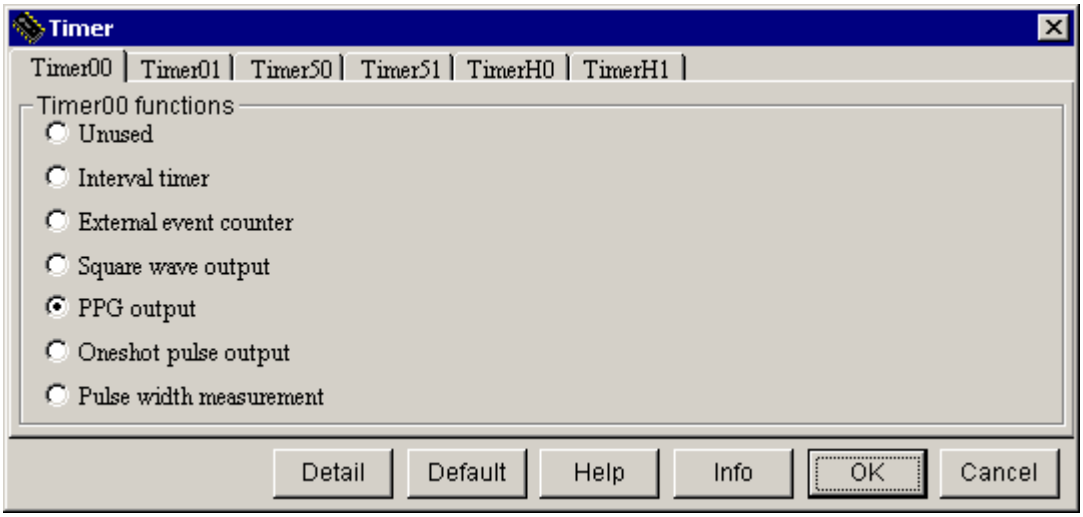
The Applilet produces the basic initialization code and driver code for Timer 00 and Timer 50, as well as the code to manage the I/O ports used for switch input and LED display output. After the Applilet produces the basic code, you can add additional code to customize the program's functions.

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

7.4.1 Configuring Applilet for Timer 00

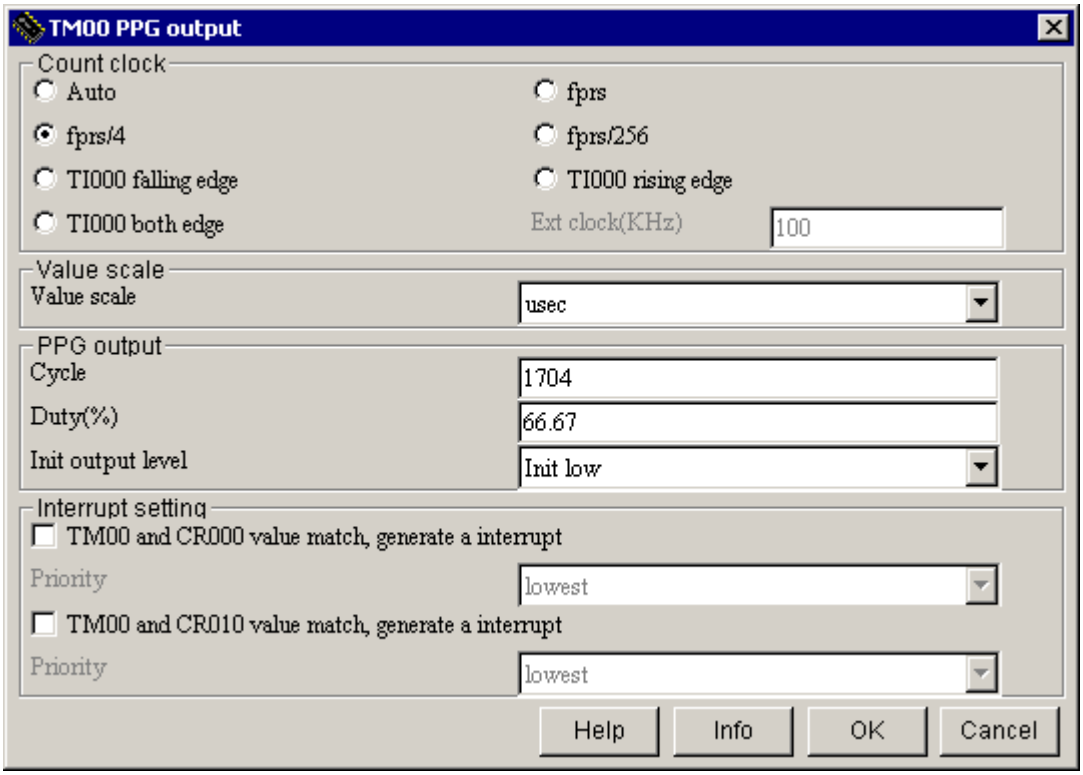
Selecting **Timer** brings up a dialog box showing the various timer blocks. Select **Timer 00** and click **PPG output**.

Figure 78. Applilet Dialog Box For Configuring Timers



Once you have selected **Timer 00** and **PPG** output, click **Detail** to bring up the **TM00 PPG output** dialog box.

Figure 79. Applilet PPG Output Dialog Box



Use this dialog box to select **TIMER 00** operation details. In this example, you want the timer to initially generate a PPG waveform of 1704 microseconds, divided into 1/3 and 2/3 duty cycle.

Select the **count clock** as **fprs/4** to give a count frequency of 5 MHz. This setting provides a fine resolution for the counter. Set **Value scale** to microseconds and **PPG output cycle** to 1704 microseconds. Setting the **Duty** cycle to **66.67**% generates the desired 1/3 and 2/3 duty cycle. Set **Init output level** for initially low output.

To avoid generating an interrupt when TM00 matches the compare registers, leave both **interrupt setting** check boxes unchecked.

7.4.2 Generating Code with Applilet

Once you have completed the timer dialog box, select **Generate code** from the Applilet's main dialog box. The Applilet shows the peripherals and functions to be generated and allows you to select a directory for the source code. Then, when you click **Generate**, the Applilet produces the initialization code, interrupt service routines, and driver subroutines for Timer 00.

The Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and displays the list of files created in a dialog box. To support Timer 00, the Applilet generates timer.h, timer.c and timer_user.c, as well as several subroutines not related to the timers.

7.4.3 Applilet-Generated Timer 00 Files and Functions

The code generated for Timer 00 support is in the files timer.h, timer.c and timer_user.c.

7.4.3.1 Timer.h

The header file timer.h contains declarations for the functions controlling Timer 00 and definitions of values for timer initialization. The header file macrodriver.h, used for all Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

7.4.3.2 Timer.c

The source file Timer.c contains the following functions:

void TM00_Init(void)
The TM00_Init() routine initializes the timer.

void TM00_Start(void)
The TM00_Start() routine starts the timer.

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

MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
The TM00_ChangeTimerCondition() function changes the value in the compare registers, CR000 and CR010, and therefore changes the PPG waveform values.

The array_reg parameter is a pointer to an array of values to be put in one or both of the compare registers; the array_num parameter is either 1 (to select CR000) or 2 (to select both CR010 and CR000).

7.4.3.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. These functions are empty on code generation, to allow you to add application-specific code. Since the demonstration does not use interrupt generation for Timer 00, there is no code in this file.

7.4.4 Applilet-Generated Files Not Related to Timer 00

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

Table 17. Files Not Related to Timer 00

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
System.h	Clock-related definitions
System.c	Clock_Init() function
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

7.4.5 Demonstration Program Files Not Generated by Applilet

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

Table 18. Files Not Generated by Applilet

File	Function
Sw_0537.h	Header file for pushbutton switch input
Sw_0537.c	Code to read and debounce pushbutton switches

7.5 Demonstration Platform

The demonstration platform for the PPG tone generator is 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.

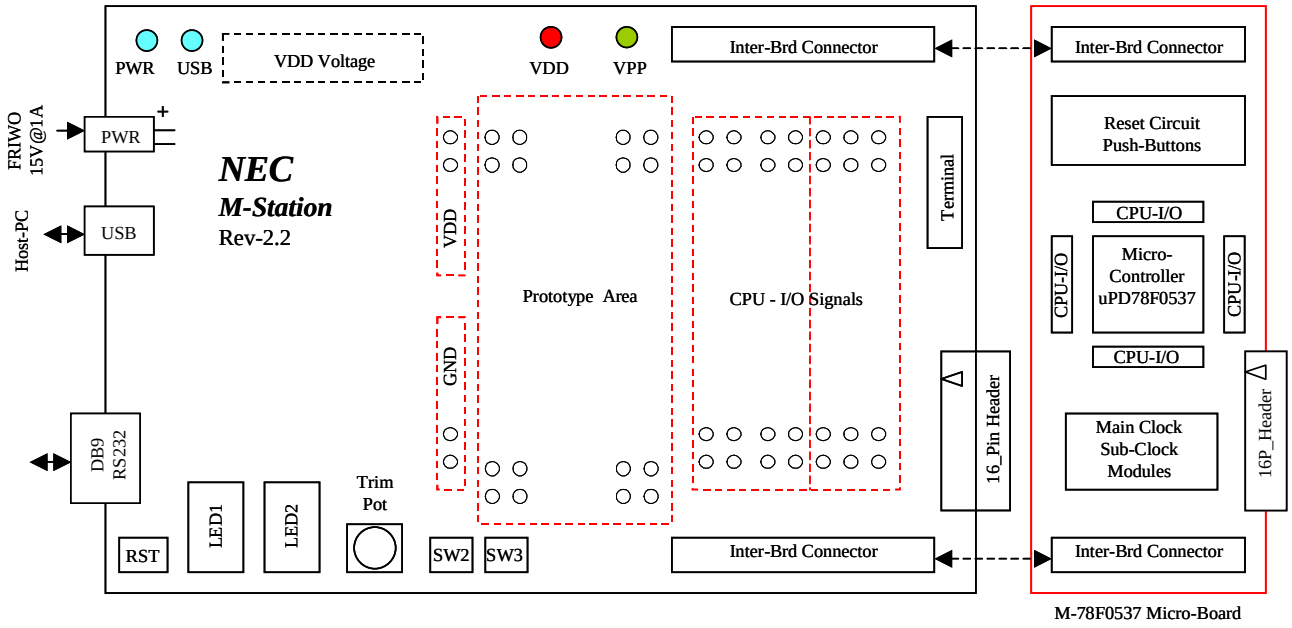
7.5.1 Resources

To demonstrate the program, use the following items:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- M-Station II Evaluation System, using M-Station II resources:
  - Pushbutton switches SW2 and SW3
  - Custom hardware added to connect a speaker to the timer output

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

Figure 80. Demonstration Hardware

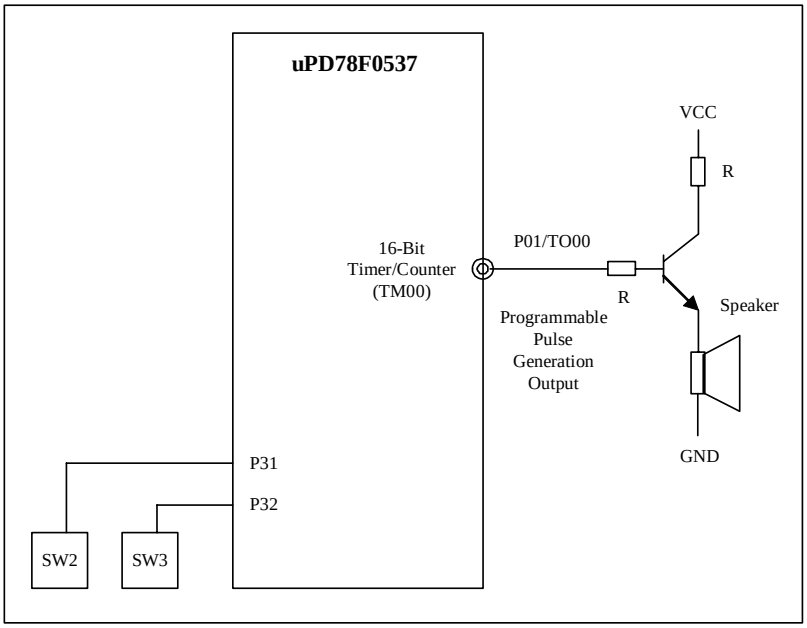


7.5.2 Demonstration of Program

With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Press SW2 440/660 Hz Two-tone sound
- Press SW3 660/880 Hz Two-tone sound

Figure 81. Hardware Block Diagram



7.6 Software Modules

The following files make up the software modules for the demonstration program. The table below shows which files are generated by the Applilet and which of those files need modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 19. Software Modules for Demonstration Program

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	No
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Sw_0537.h	Switch input definitions	--	Yes
Sw_0537.c	Switch input functions	--	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

8. Square-Wave Tone Generation Using 16-Bit Timer 00 (TM00)

Most 8- and 16-bit timer/counters for the 78K0-family of microcontrollers provide a square-wave output. You can use the 16-bit output to drive a speaker and generate tones.

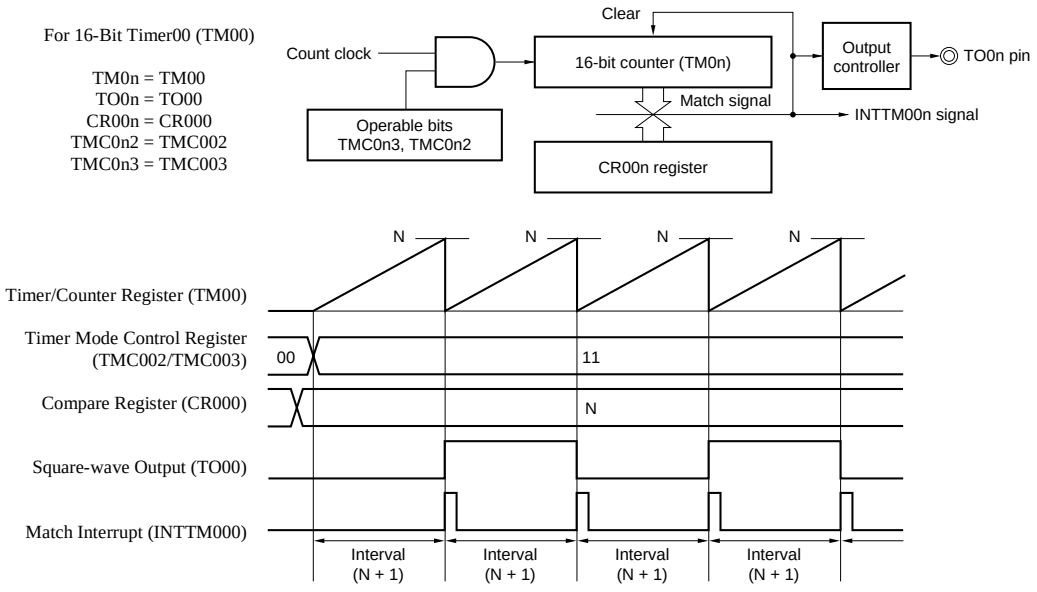
8.1 Features of Square-Wave Generation

In its square-wave generation mode, the 16-bit Timer 00 offers:

- Variable time base, based on divisions of the peripheral clock, using the PRM00 register
- Variable square-wave half-cycle time of 1 to 65536 clocks, using the CR000 register
- Precise 50% duty cycle, using CR000 to count both high and low periods
- Optional interrupts at the end of each half cycle

The timer outputs a square wave from output pin TO00. Compare register CR000 establishes the square-wave pulse width, and the output begins when the timer mode-control register is set to clear-and-start mode.

Figure 82. Square-Wave Tone Generation



Square-wave frequency = 1/[2x(N+1)x(Count Clock Cycle)]

To configure the timer for square-wave operations:

- Set the time base using PRM00.
- Set CR000 as a compare register using CRC00.

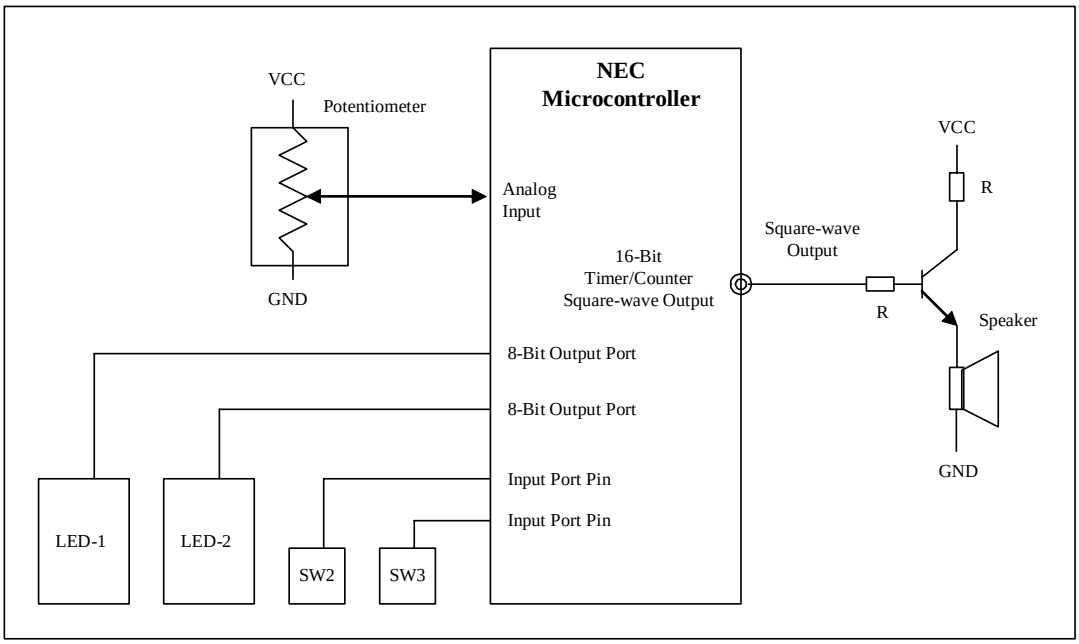
- Set the half-cycle time in CR000.
- Set the output-port latch and mode-register bit low (for square-wave output).
- Set the priority for the interrupts, clear the interrupt flags, and unmask the interrupts, if used.
- Set the TOC00 register for enable output, and invert on CR000 match.
- Enable the timer by setting the timer-mode register TMC00 for clear-and-start mode on compare of TM00 and CR000.

8.2 Program Description and Specification

For this demonstration program you need to configure two switches and a potentiometer as shown. Initially, the square wave is set at approximately 440 Hz, which produces a single tone from the speaker.

If you press and hold switch SW2 or SW3, you can change the pitch of the tone by adjusting the setting of the potentiometer. When you press SW2, the tone changes over a wider range as the A/D conversion value from the potentiometer is loaded to the upper byte position of the compare register. If you press and hold switch SW3 while adjusting the potentiometer, the tone varies over a narrower range as this value is loaded into the lower byte of the compare register.

Figure 83. Hardware Configuration



Specifications:

- TM00 is set for square-wave mode, no interrupt.
- TM00 time base clock is set to 5 MHz.
- The analog input ANI0/P20 accepts DC voltages ranging from 0 to 5.0V.
- SW2 adjusts the output frequency from 00xx to FFxx counts.
- SW3 adjusts the output frequency from xx00 to xxFF counts.

8.3 Software Flow Charts

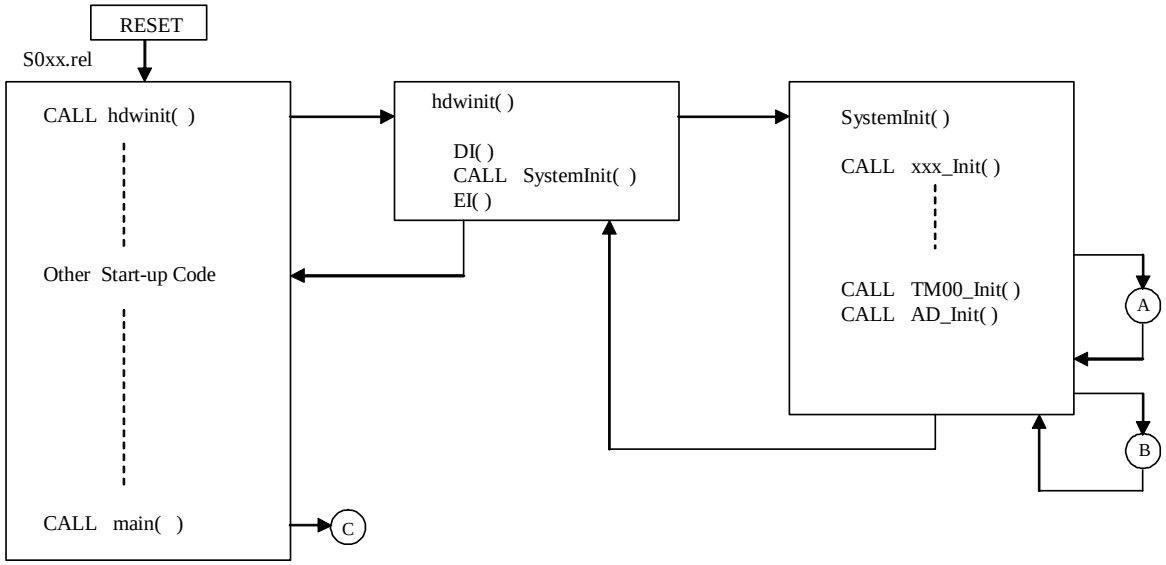
The demonstration program consists of four major sections:

- The initialization code for the program, which is called before the main() program starts
- The main program loop, which checks the status of switches, reads the A/D converter, and changes the timing of TM00
- Subroutines generated by the Applilet to handle starting and stopping the timer, changing it's interval, and handling A/D input
- Subroutines for pushbutton input and LED display output

The following flowcharts describe initialization, the main program, subroutines related to TM00.

8.3.1 Program Startup and Initialization

Figure 84. Program Startup and Initialization Flow



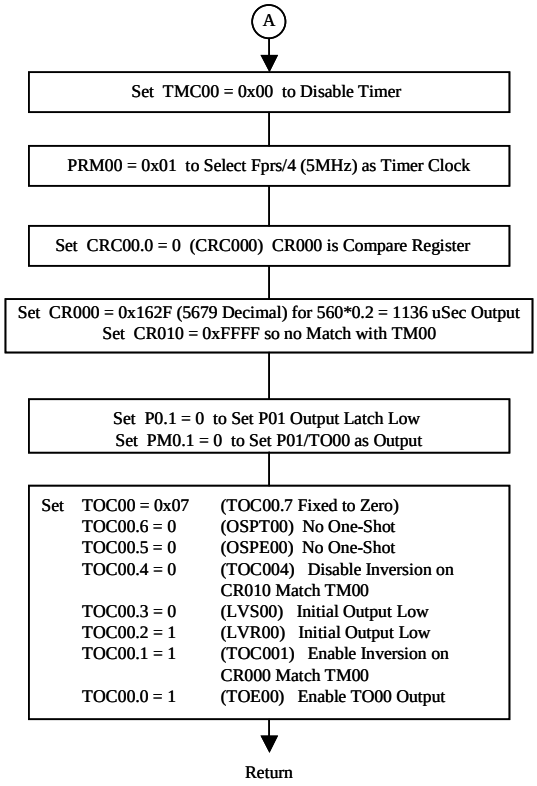
For 78K0 programs written in the C language, the startup code for the C program is supplied by an object code file such as s01.rel, which links into the user program. This startup code calls a function named hdwinit( ); you can place any specialized hardware-initialization code here.

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

After the hdwinit( ) function finishes, the startup code calls the main( ) function in the user program. So at the start of the main( ) function, all peripherals have been initialized, and the main( ) function does not need to call individual initialization routines.

8.3.2 TM00_Init( ) -- Timer 00 Initialization for Square-Wave Output

Figure 85. Initializing Timer for Square-Wave Output



SystemInit() calls the TM00_Init() routine to initialize the timer for square-wave output. The routine disables the timer while writing to the control register.

The routine sets prescaler-mode register PRM00 to 0x01 to select fprs/4 as Timer 00's count clock. Then, with the main system clock running at 20 MHz, the count clock is 5 MHz with each count lasting 0.2 microseconds.

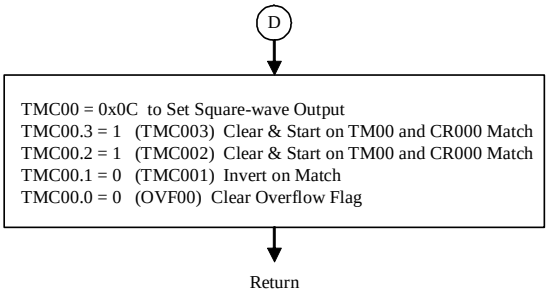
CRC00 uses CR000 as a compare register. CR000 is set to 0x162F (5679), for a square wave cycle time of 5680 * 0.2 = 1136 microseconds. This setting results in a square wave with 1136 microseconds high and 1136 microseconds low, for a total frequency of 1/(1136 + 1136) = about 440 Hz.

Timer output TO00 is used on pin P01/TO00. Setting the P0.1 output latch and the port mode register PM0 bit 1 to zero forces the output to pin P01.

The TOC00 control register is set to make the initial output low, invert the timer output on TM00 and CR000 match, and to enable the TO00 output.

8.3.3 TM00_Start( ) – Start Timer 00 Operation

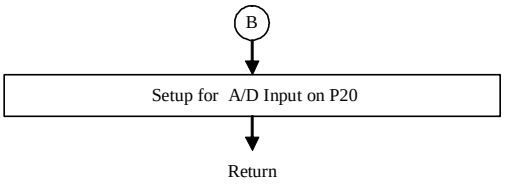
Figure 86. Starting Timer



The TM00_Start() routine configures TMC00 for square-wave output and enables the timer.

8.3.4 AD_Init( ) – Initialize A/D Converter

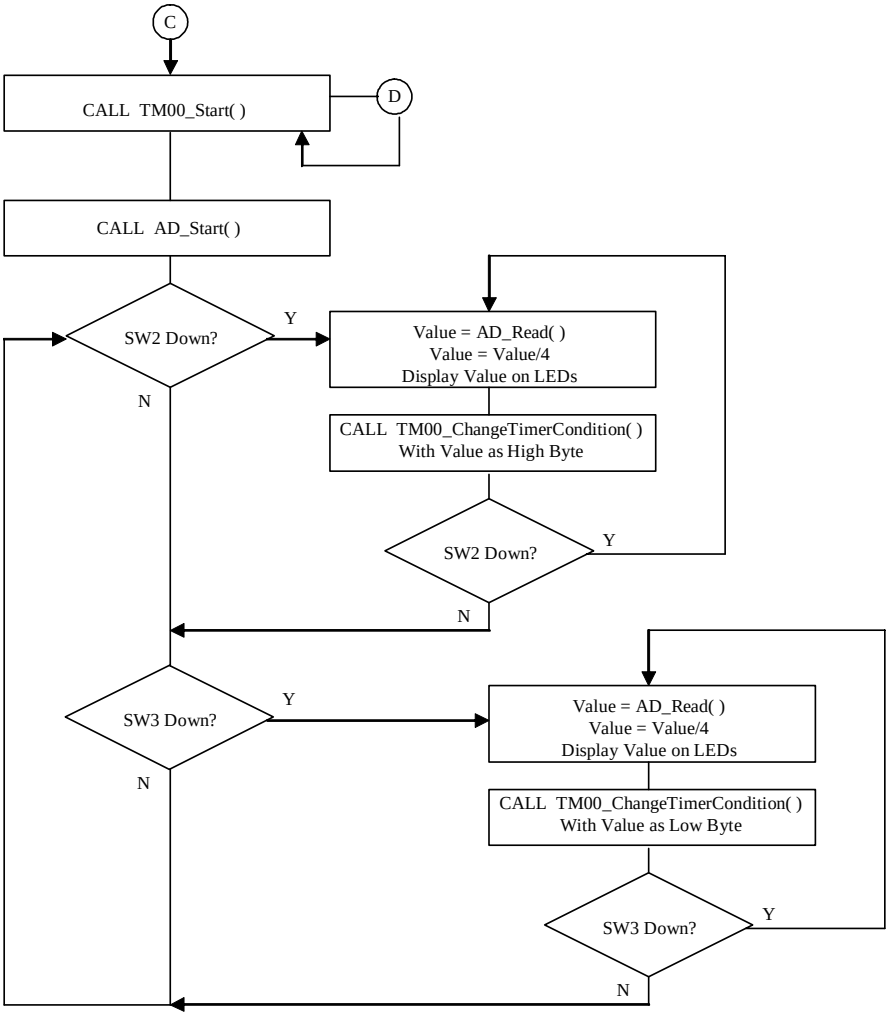
Figure 87. Initializing Timer



The AD_Init() function initializes the A/D converter to read the analog input from the potentiometer (ANI0), which is available on the ANI0/P20 pin.

8.3.5 Main( ) – Main Program — Tone Generation Using Square Wave

Figure 88. Flowchart for Generating Tones with Square Wave



The main program calls TM00_Start to start the Timer 00 timer output and calls AD_Start() to set up the A/D input. At this point, Timer 00 produces a square-wave output of about 440 Hz.

The main program checks to determine if either SW2 or SW3 is down. If neither switch is down, no action is taken, and the tone is stable. If SW2 is down, the program reads the analog input from the potentiometer, sets the high byte of CR000 to the value read, and displays that value on the LEDs. This high-byte value changes the square-wave frequency to a new frequency, depending on the value of the voltage from the potentiometer. As long as SW2 is down, you can continue to adjust the potentiometer, varying the frequency over a broad range.

If SW3 is down, the program again reads the analog input from the potentiometer, storing the value in CR000's low byte and displaying the value on the LEDs. This value change also changes the frequency of

the tone, but because the value is loaded into the low byte, the change is smaller than before, with the frequency varying over a narrower range. Thus, SW3 fine tunes the frequency of the tone, while SW2 provide a coarse adjustment.

8.4 Applilet's Reference Driver

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

The Applilet produces the basic initialization and driver code for Timer 00 and Timer 50, as well as code to manage the I/O ports used for switch input and the LED display output. Once the Applilet produces the basic code, you can add additional code to customize the program.

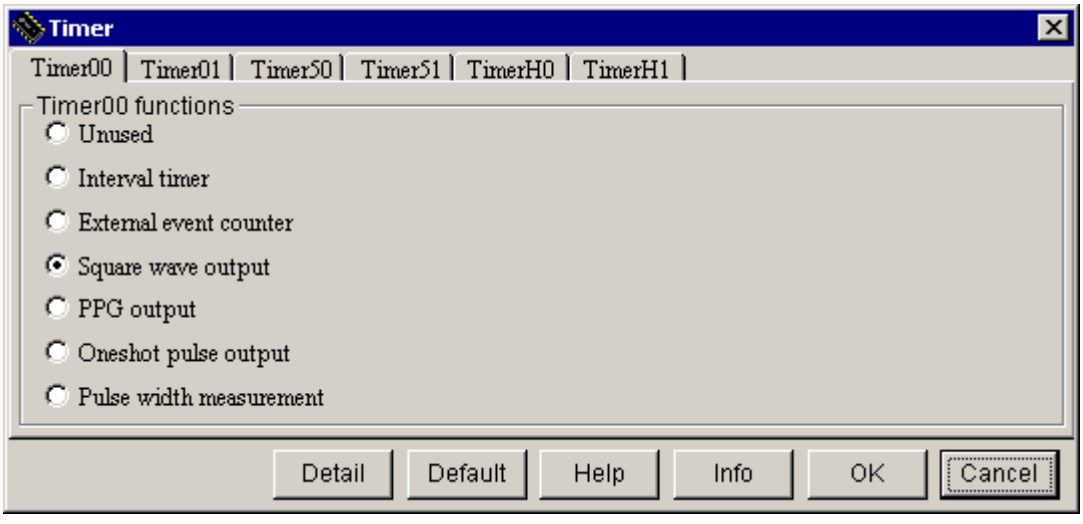
This section describes how to configure the Applilet to produce code for the timers and lists the routines produced.

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

8.4.1 Configuring Applilet for Timer 00

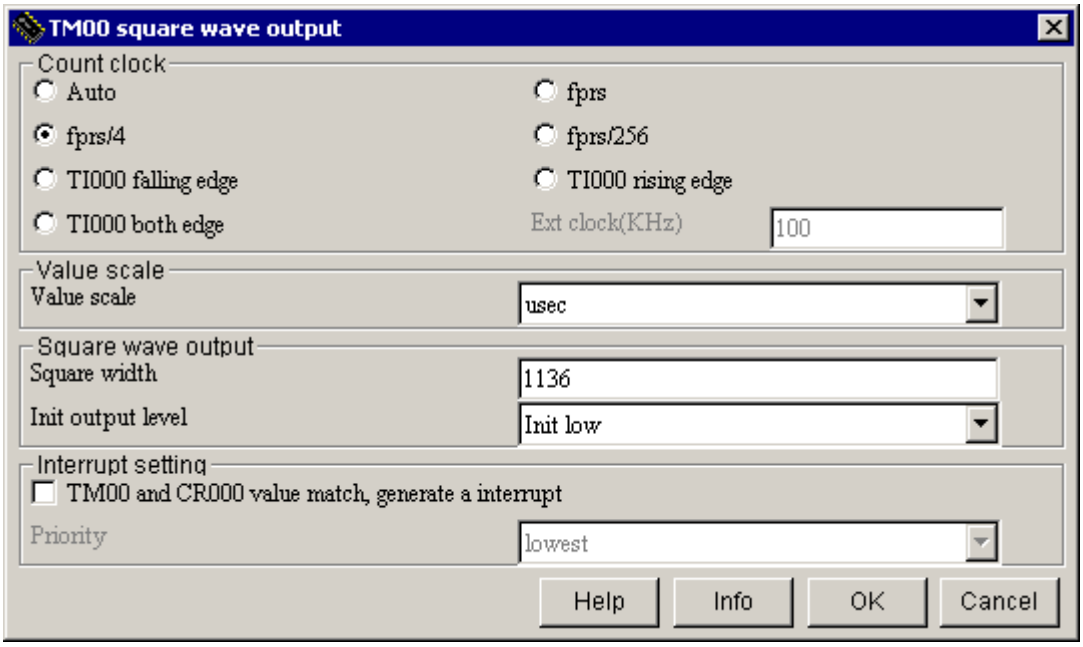
Selecting **Timer** brings up a dialog box showing the various timer blocks. Select **Timer 00**, and click **Square wave output**.

Figure 89. Applilet Timer-Block Selection Dialog Box



Once you have selected **Timer 00**, clicking **Detail** brings up the **TM00 square-wave output** dialog box.

Figure 90. Applilet Detail Dialog Box



Here you can select the appropriate timer-operation details. For this demonstration, you want the timer to generate a square wave of about 440 Hz. This output requires a high and low time of 1136 microseconds.

Set **Count clock** to **fprs/4** to get 5 MHz, with 0.2 microseconds per clock. Set **Value scale** to microseconds and **Square width** to 1136. Note that this value is the width of one half of the square wave. Set **Init output level** to be low.

Since you do not want to generate an interrupt when TM00 matches CR000, leave the **Interrupt setting** box unchecked.

8.4.2 Generating Code with Applilet

Once you have set up the timer dialog box, select **Generate code** from the Applilet's main dialog box. The Applilet displays the peripherals and functions, and allows you to select a directory for the source code. When you click **Generate code**, the Applilet produces the initialization code, interrupt service routines, and driver subroutines for the timer.

The Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and shows the list of files created in a dialog box. To support Timer 00, the Applilet generates timer.h, timer.c and timer_user.c subroutines, as well as several routines not related to the timer.

8.4.3 Applilet-Generated Timer 00 Files and Functions

The Applilet stores the code generated for Timer 00 support in the files timer.h, timer.c and timer_user.c.

8.4.3.1 Timer.h

The header file timer.h contains declarations for the functions controlling Timer 00 and definitions of values for Timer 00 initialization. The header file macrodriver.h, used for all of the Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some functions.

8.4.3.2 Timer.c

The source file Timer.c contains the following functions:

void TM00_Init(void)
The TM00_Init() routine initializes the timer.

void TM00_Start(void)
The TM00_Start() routine enables the timer.

void TM00_Stop(void)
The TM00_Stop() routine disables the timer. The demonstration program does not use this routine.

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

The array_reg parameter points to an array of values to be put in one or both of the compare registers; the array_num parameter is either 1 (to select CR000) or 2 (to select both CR010 and CR000).

8.4.3.3 Timer_user.c

The source file timer_user.c contains stub functions for user code. These functions are empty on code generation to allow you to add application-specific code. Since the timer interrupt is not used in the demonstration, this file does not contain any code.

8.4.4 Applilet-Generated Files Not Related to Timer 00

The Applilet also generates several other source files for the demonstration program.

Table 20. Files Not Related to Timer 00

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
System.h	Clock-related definitions
System.c	Clock_Init() function
Ad.h	Defines for the A/D converter
Ad.c, Ad_user.c	A/D functions
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

8.4.5 Demonstration Program Files Not Generated by Applilet

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

Table 21. Files Not Generated by Applilet

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

8.5 Demonstration Platform

The demonstration platform for the square-wave tone generator is 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.

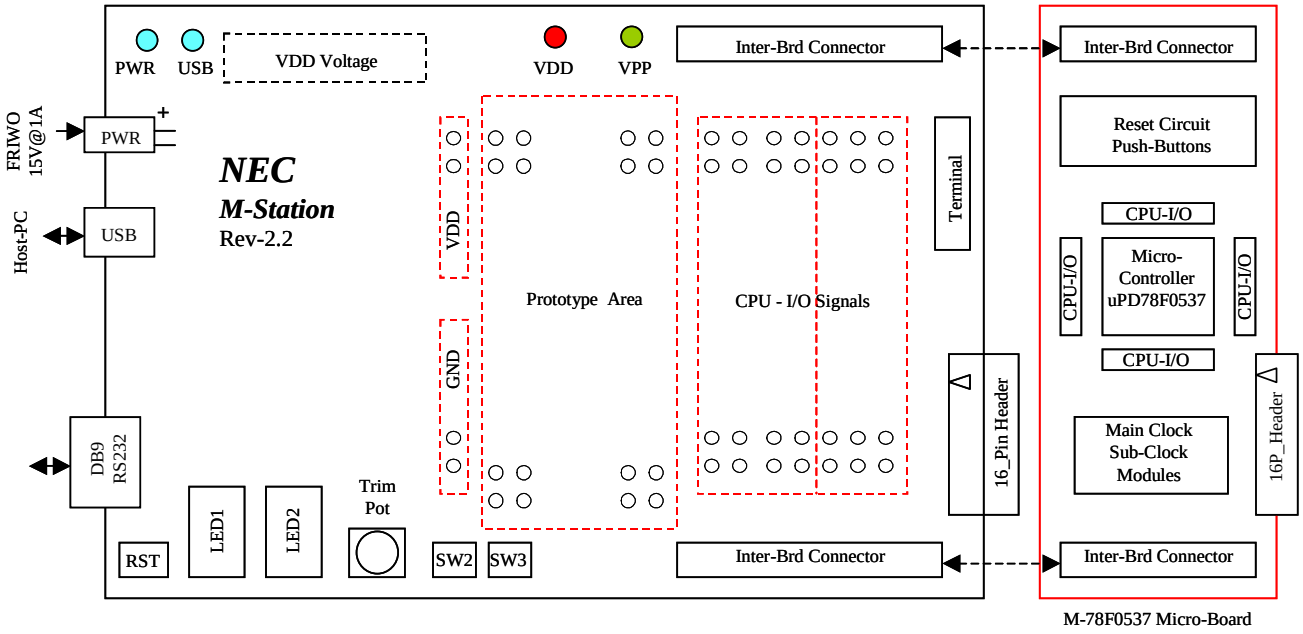
8.5.1 Resources

Demonstrating the program requires:

- M-78F0537 Micro-Board, with the uPD78F0537 8-bit microcontroller
- M-Station II Evaluation System, using M-Station II resources:
  - Pushbutton switches SW2 and SW3
  - 7-segment LEDs LED1 and LED2
  - Potentiometer providing variable analog input voltage
  - Custom hardware to connect a speaker to the timer output

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

Figure 91. Demonstration Hardware



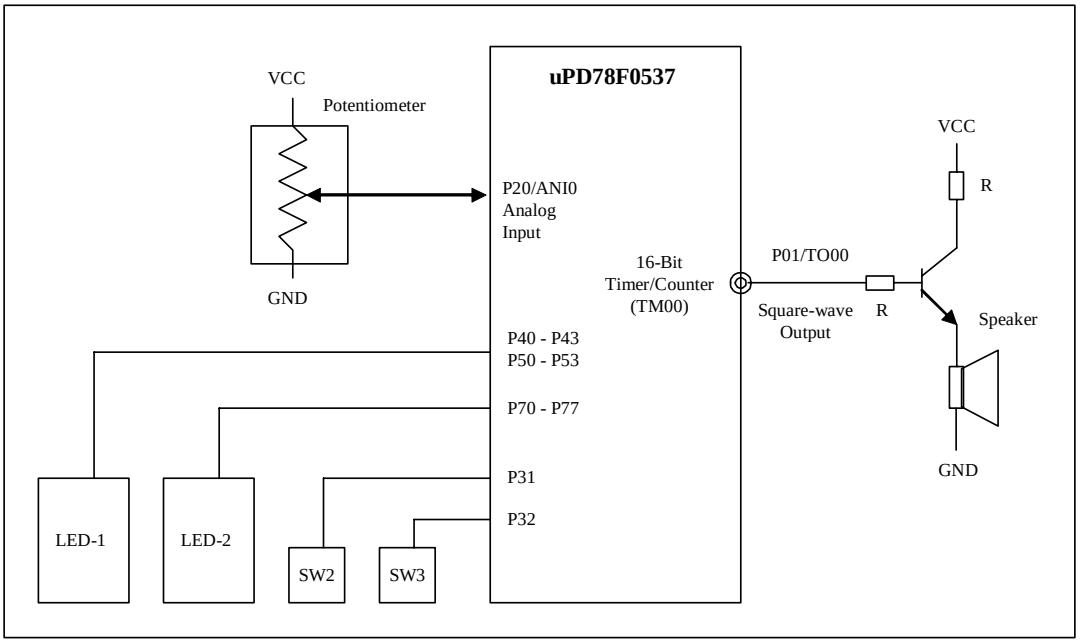
8.5.2 Demonstration of Program

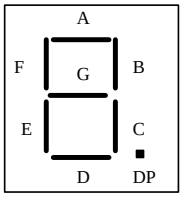
With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Press and hold SW2 Change potentiometer setting Wide-range tone change
- Press and hold SW3 Change potentiometer setting Narrow-range tone change

8.6 Hardware Block Diagram

Figure 92. Hardware Block Diagram





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

8.7 Software Modules

The following files make up the software modules for the demonstration program. The table below shows which files are generated by the Applilet and which of those files need modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 22. Demonstration Program Software Modules

File	Purpose	Generated By Applilet	Modified By User
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
TIMER_user.c	User code for timer interrupt handling	Yes	No
Ad.h	A/D converter related definitions	Yes	No
Ad.c	A/D converter functions	Yes	No
Ad_user.c	User code for A/D converter operation	Yes	No
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Led_0537.h	LED display definitions	--	Yes
Led_0537.c	LED display functions	--	Yes
Sw_0537.h	Switch input definitions	--	Yes
Sw_0537.c	Switch input functions	--	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

9. Timekeeping with Watch Timer (WTM)

A watch timer is available in most NEC Electronics' 78K0-family of microcontrollers. The watch timer provides time-keeping functions, generating periodic interrupts based on a precise clock.

The timekeeping demonstration described here uses a watch timer based on a 32.768-kHz crystal. This example also demonstrates the use of interval timing to debounce input switches.

9.1 Watch Timer Features

The watch timer offers the following features:

- Serves as either a watch timer or interval timer (or both simultaneously)
- Variable time base, based on either peripheral clock division or an external subclock
- Variable watch-timer interrupt at one of four different prescaled intervals
- Variable interval-timer interrupt at one of eight different prescaled intervals

Figure 93. Watch Timer Block Diagram

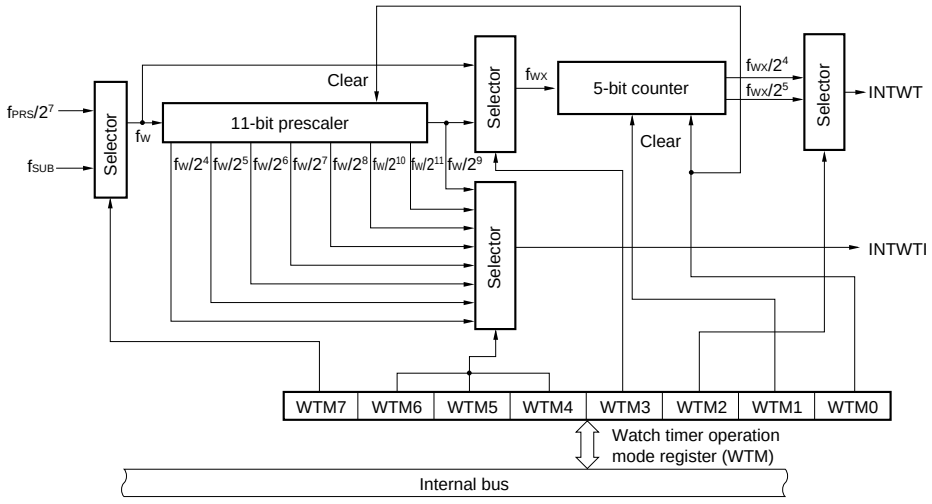
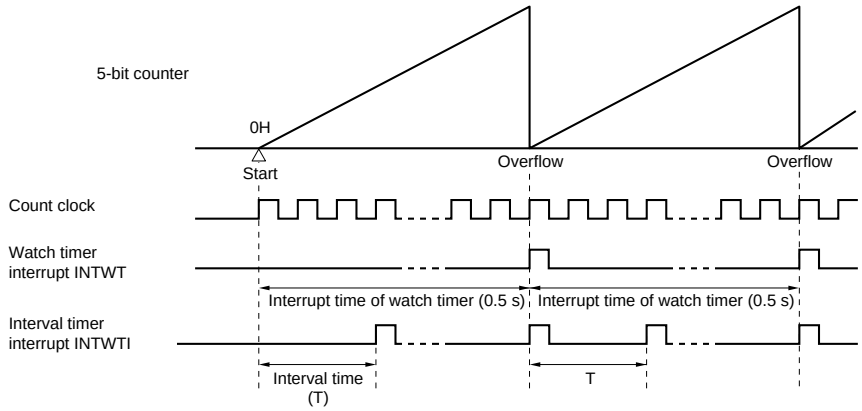


Figure 94. Timing of Watch Timer Cycle



The watch timer implements the following hardware for most NEC microcontrollers.

Table 23. Watch Timer Hardware Configuration

Item	Configuration
Counter	5 bits × 1
Prescaler	11 bits × 1
Control register	Watch timer operation mode register (WTM)

The watch timer uses peripheral hardware or a subsystem clock to generate an interrupt request (INTWT) at a specific time interval. When operating as an interval timer, the watch timer generates an interrupt signal (INTWTI) at an interval based on a preset count value.

Use this procedure to configure the watch timer for either watch timer or interval timer operation:

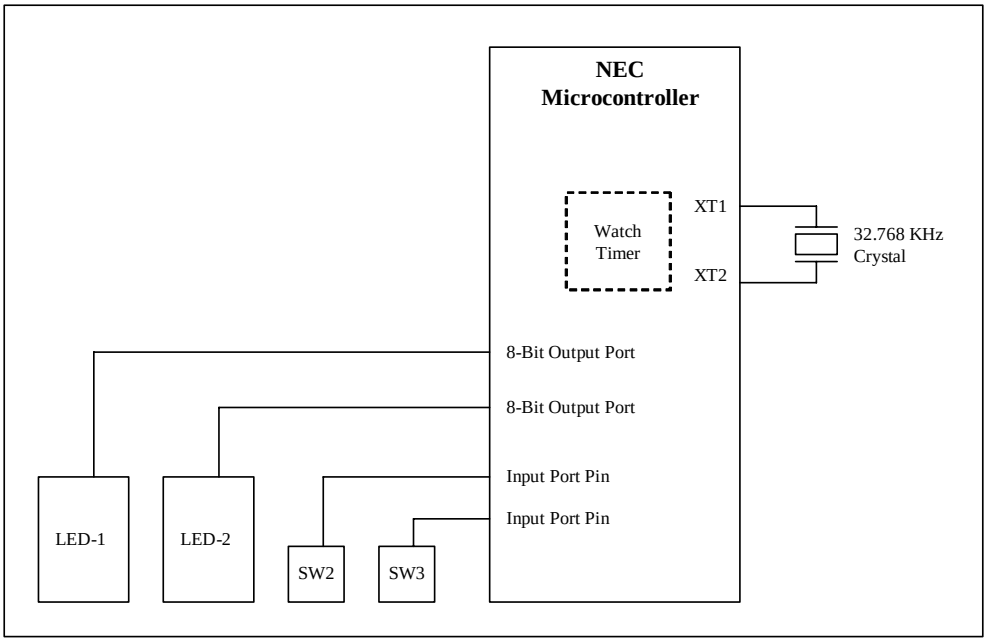
- Use WTM.7 to set the input clock as fprs/128 or subclock.
- Use WTM3 - WTM2 to set the watch timer divider.
- Use WTM6 - WTM4 to set the interval timer divider.
- Set the priority for the watch-timer interrupt, clear the interrupt flag, and unmask the interrupt, if used.
- Set the priority for the interval-timer interrupt, clear the interrupt flag, and unmask the interrupt, if used.
- Set WTM1 and WTM0 to 1 to enable the timer.

9.2 Program Description and Specification

The demonstration uses the watch timer to display seconds on two 7-segment LEDs. This timing is kept by the watch-timer function. The program also uses the watch timer interval function to debounce inputs from switches SW2 and SW3.

Pressing SW2 starts or stops the seconds timing. Pressing SW3 resets the seconds counter timing to 00.

Figure 95. Demonstration Hardware



Specifications:

- The watch-timer input is set for a 32.768 kHz subclock.
- The periodic watch-timer interrupt is set to occur every 0.5 seconds.
- The periodic interval-timer interrupt is set for 1 millisecond.
- Switches are debounced when stable for 10 milliseconds.

9.3 Software Flow Charts

The demonstration program consists of the following major sections:

- Initialization code for the program, called before the main() program starts
- The main program loop, which checks the status of switches, and starts, stops, or clears the seconds timer

- Subroutines generated by the Applilet to handle WTM starting, stopping and changing of intervals
- Subroutines with user code for handling WTM watch timer and interval-timer interrupts (consisting of the Applilet-generated stub interrupt-service routines, with user code added).
- Subroutines for pushbutton input and LED display

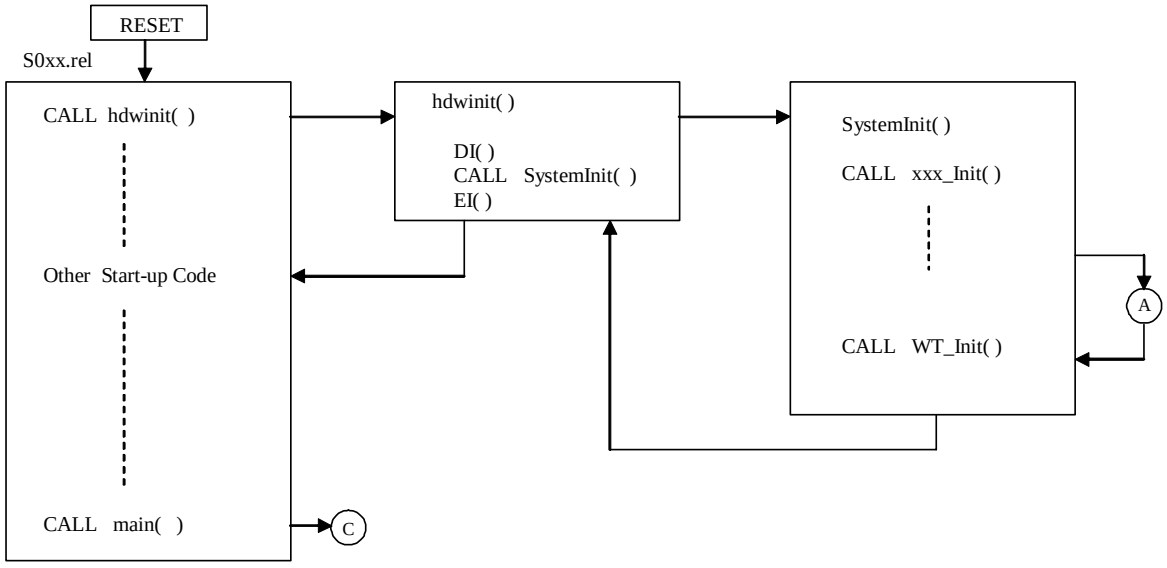
The following flowcharts describe the initialization, main program, and subroutines related to the watch timer.

9.3.1 Program Startup and Initialization

For 78K0 programs written in the C language, an object code file such as s01.rel supplies startup code that you link into the user program. This startup code calls a function named hdwinit( ); you can place any specialized hardware-initialization code here.

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

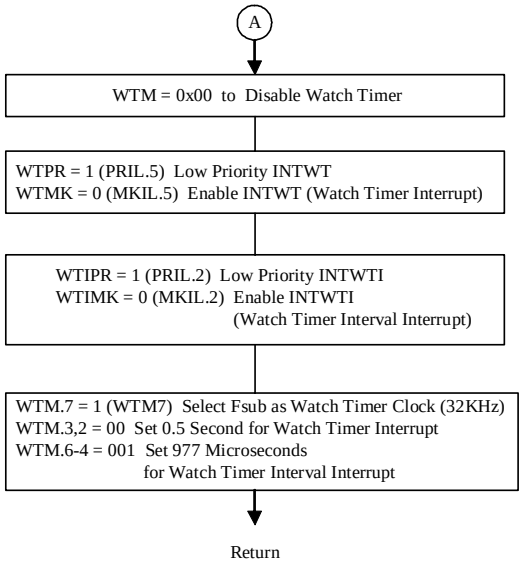
Figure 96. Program Startup Flow



When hdwinit( ) completes, the startup code calls the main( ) function of the user program. So at the start of the main( ) function, all peripherals have been initialized, and the main( ) function does not need to call individual initialization routines.

9.3.2 WT_Init( ) – Watch-timer Initialization

Figure 97. Initializing Watch Timer



SystemInit() calls the WT_Init() routine, which initializes the watch timer for watch-timer and interval-timer functions. The routine disables the watch timer by setting WTM.0 to zero while other WTM register settings are changed.

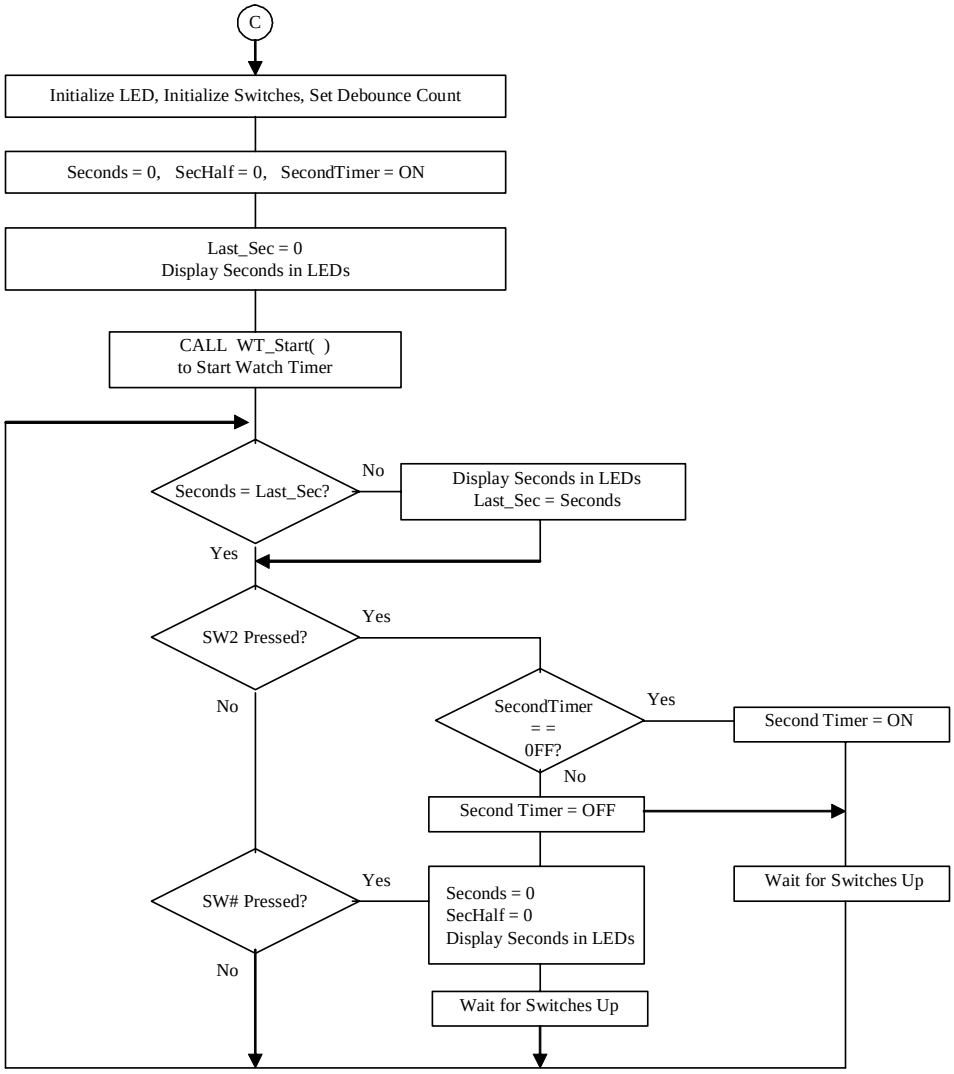
Setting the WTPR (PR1L.5) and WTMK (MK1L.5) bits gives the WTM watch-timer interrupt INTWT low priority and enables the interrupt.

Setting the WTIPR (PR1L.2) and WTIMK (MK1L.2) bits gives the WTM interval-timer interrupt INTWTI low priority and enables the interrupt.

The routine sets WTM.7 to one to establish the 32.768-kHz subclock as fw, the time base for WTM. Setting WTM bits 3 and 2 to zero sets the watch-timer interrupt to 16384/fw, or an interrupt every 0.5 seconds. Setting WTM bits 6 through 4 to 001 sets the interval-timer interrupt to 32/fw, or 977 microseconds, for an interval time of approximately 1 millisecond.

9.3.3 Main( ) –Main Program – Timekeeping with Watch Timer

Figure 98. Main Program Flow



The main() function initializes the LEDs and sets the debounce counter for the switches. The variables used to keep track of seconds are cleared, and the SecondTimer variable is set to ON. The program then calls WT_Start() to start the watch timer.

The program then enters the main loop. In the loop, the program checks to see if the current value of seconds matches the last value, Last_Sec. If the values are not the same, the main program displays the seconds variable and sets Last_Sec to the seconds variable. In this way, the main program reflects any changes to the seconds variable in the display.

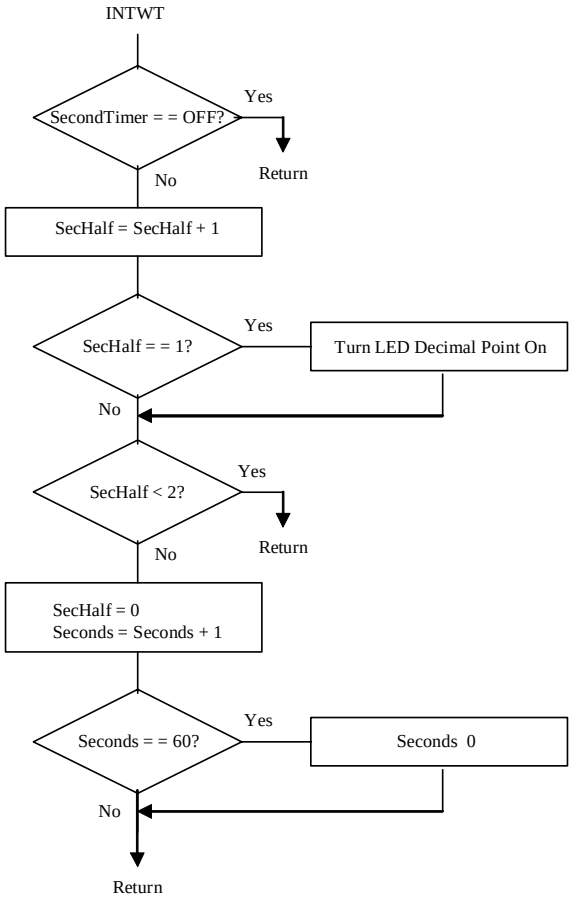
The program then checks to see if switches are down by checking the debounced value of the switch states. If no switch is down, the program returns to the top of the main loop.

If SW2 is down, the program inverts the state of the SecondTimer variable, turning the variable ON if it was OFF, and turning it OFF if it was ON. The program then waits for the switches to be up.

If SW3 is down, the program clears the variables tracking seconds to zero and waits for the switches to return to the up position.

9.3.4 MD_Init( ) – Watch-timer Interrupt-Service Routine

Figure 99. Flowchart for Interrupt-Service Routine



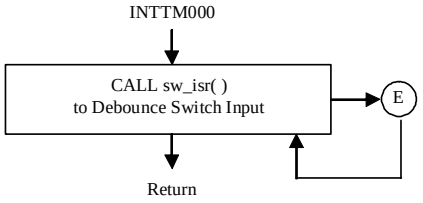
When the INTWT interrupt occurs — every 0.5 second — the program calls the MT_INTWT() interrupt-service routine. If the SecondTimer variable is set to OFF, the routine just returns, and seconds are not counted up.

If SecondTimer is ON, however, then SecHalf (the half-second counter) counts up. If the counter has gone from zero to one (end of the first half second), program turns on the decimal point on the LED display. This decimal point provides a visual indicator of half seconds.

If SecHalf is less than two, the routine just returns. If SecHalf is two or more, then one second has elapsed. The Second variable is counted up, and SecHalf is set to zero. If 60 seconds have elapsed, the Second variable is set to zero.

9.3.5 MD_INTWTI( ) – Watch-Timer Interval Interrupt-Service Routine

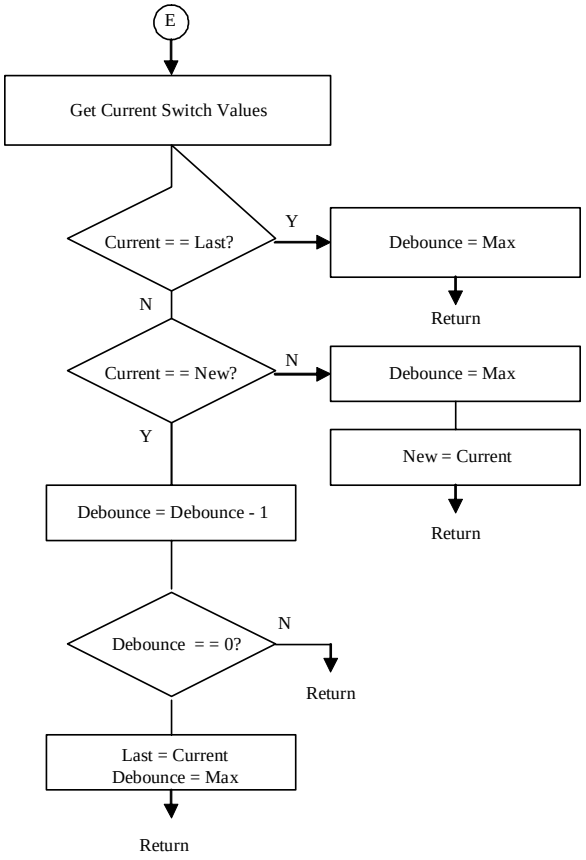
Figure 100. Flowchart for Watch-Timer Interval Interrupt-Service Routine



When the watch-timer interval interrupt, INTWTI, occurs — about every millisecond — the MD_INTWTI() interrupt-service routine is called. This routine, in turn, calls the sw_isr() function in sw_0537.c to debounce the switches.

9.3.6 SW_isr( ) — Switch Debounce Interrupt-Service Routine

Figure 101. Flowchart for Switch-Debounce Interrupt-Service Routine



MD_INTWTI( ) calls the debounce routine (not generated by the Applilet) once every 1 millisecond. The routine compares the current value of the switches to previous values. Once the values have been stable for the specified number of debounce counts, the routine updates the sw_new variable to provide a debounced switch setting. The main program checks this debounced value by calling the routine sw_get();

9.4 Applilet's Reference Driver

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

The Applilet produces the basic initialization code and driver code for Timer 00, as well as code to manage the I/O ports used for switch input and LED display output. After the Applilet produces the basic code, you can insert additional code to customize the functioning of the program.

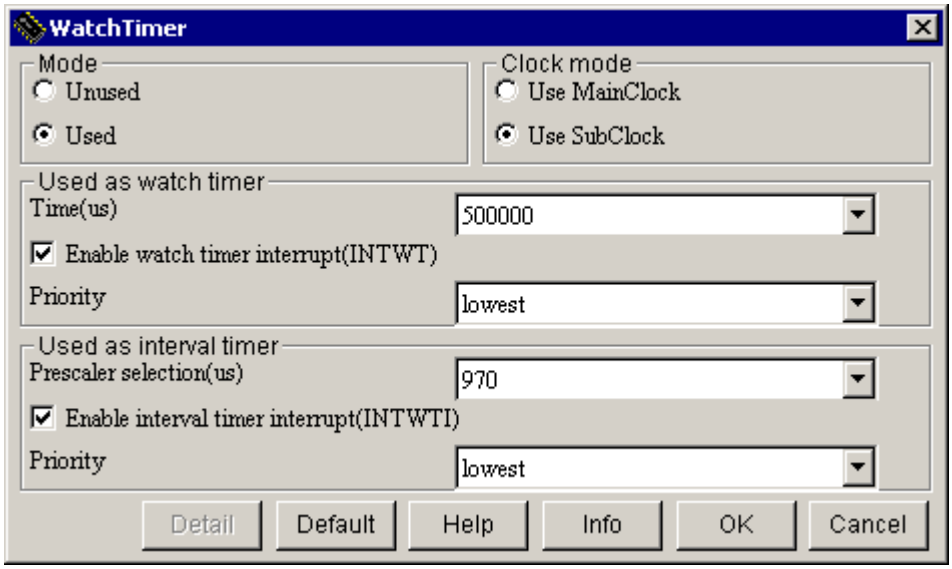
This section describes how to configure the Applilet to produce code for Timer 00 and lists the routines produced.

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

9.4.1 Configuring Applilet for Watch Timer Operation

Selecting **WatchTimer** brings up a dialog box displaying the watch timer settings.

Figure 102. Watch Timer Setup Dialog Box in Applilet



Set **Mode** to **used** and click **Use subclock** to enable the 32.768-kHz subclock as the time base.

In the **Used as watch timer** section, select 0.5 second (500,000 microseconds) from the drop-down menu, and click **Enable watch timer interrupt**.

In the **Used as interval timer** section, set **Prescaler selection** to 970 microseconds and click **Enable interval timer interrupt**.

9.4.2 Generating Code with Applilet

Once you have set up the watch-timer dialog box, select **Generate code** from the Applilet's main dialog box. The Applilet shows the peripherals and functions and allows you to select a directory in which to store the source code. The Applilet then generates the initialization code, interrupt-service routines, and driver subroutines.

When you press **generate**, the Applilet creates the code in several C-language source files (extension .c) and header files (extension .h), and shows the list of files created in a dialog box. To support the watch timer, the Applilet generates the subroutines timer.h, timer.c and timer_user.c, as well as several others not related to Timer 00.

9.4.3 Applilet-Generated Watch Timer Files and Functions

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

9.4.3.1 Watchtimer.h

The header file watchtimer.h contains declarations for the functions controlling the watch timer. The header file macrodriver.h, used for all of the Applilet-generated code, also defines some data types and values, such as the MD_STATUS values returned by some watch timer functions. Declarations of variables and values related to seconds timekeeping are added here.

9.4.3.2 Watchtimer.c

The source file watchtimer.c contains the following functions:

void WT_Init(void)
The WT_Init() routine initializes the watch timer.

void WT_Start(void)
The WT_Start() routine enables the timer and interrupts INTWT and INTWTI.

void WT_Stop(void)
The WT_Stop() routine stops the watch timer by disabling the timer and interrupts. The demonstration program does not use this routine.

9.4.3.3 Watchtimer_user.c

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

__interrupt void MD_INTWT(void)
This is the interrupt-service routine for the watch-timer interrupt INTWT. This service routine is blank when the Applilet generates it. You add code to use the watch timer to count seconds.

__interrupt void MD_INTWTI(void)
This is the interrupt-service routine for interval-timer interrupt INTWTI. The Applilet generates

this routine as a blank interrupt-service routine. To use the watch-timer interval timer to debounce switches, place a call to function sw_isr() in MD_INTTM000().

The sw_isr() routine, located in sw_0537.c, has code that checks the value of the switches every millisecond. Once the value has stayed stable for the required number of milliseconds, the routine reports the new value as the debounced setting of the switches. Function sw_isr() then returns to MD_INTWTI(), which then returns to the main program where the interrupt occurred.

The sw_isr() code could have been placed directly in MD_INTWTI(); routine. This location would be suitable for saving the time of the call and returning to sw_isr(), but, the arrangement described allows you to locate all switch-related functions in the sw_0537.c source file, for clarity.

9.4.4 Applilet-Generated Files Not Related to Watch Timer

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

Table 24. Files Not Related to Watch Timer

File	Function
Macrodriver.h	General header file for Applilet-generated programs
Systeminit.c	SystemInit() and hdwinit() functions for initialization
Main.c	The main program function
System.h	Clock-related definitions
System.c	Clock_Init() function
Port.h	Definitions for default I/O port states
Port.c	PORT_Init() function
Option.asm	Defines the option byte and security bytes
Option.inc	Defines settings for the option byte and security settings

9.4.5 Demonstration Program Files Not Generated by Applilet

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

Table 25. Files Not Generated by Applilet

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

9.5 Demonstration Platform

The demonstration platform for timekeeping with the watch timer is 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.

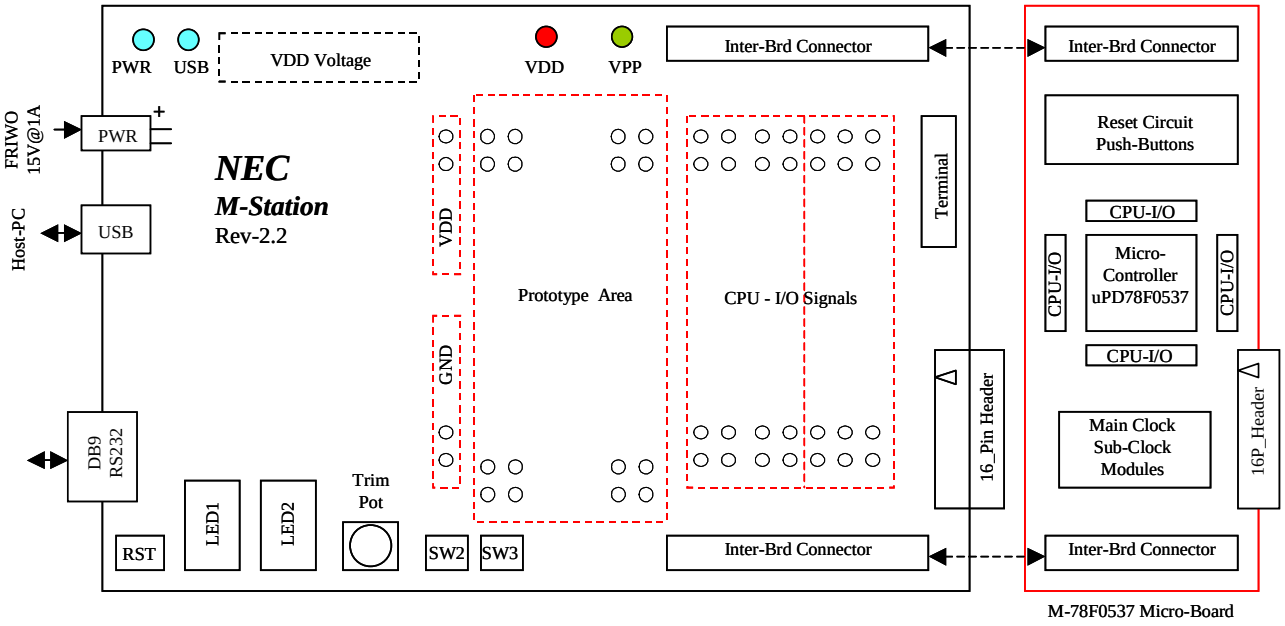
9.5.1 Resources

The demonstration program uses the following resources:

- M-78F0537 Micro-Board, with uPD78F0537 8-bit microcontroller mounted
- M-Station II Evaluation System, using M-Station II resources:
  - Pushbutton switches SW2 and SW3
  - 7-segment LEDs LED1 and LED2

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

Figure 103. Demonstration Hardware



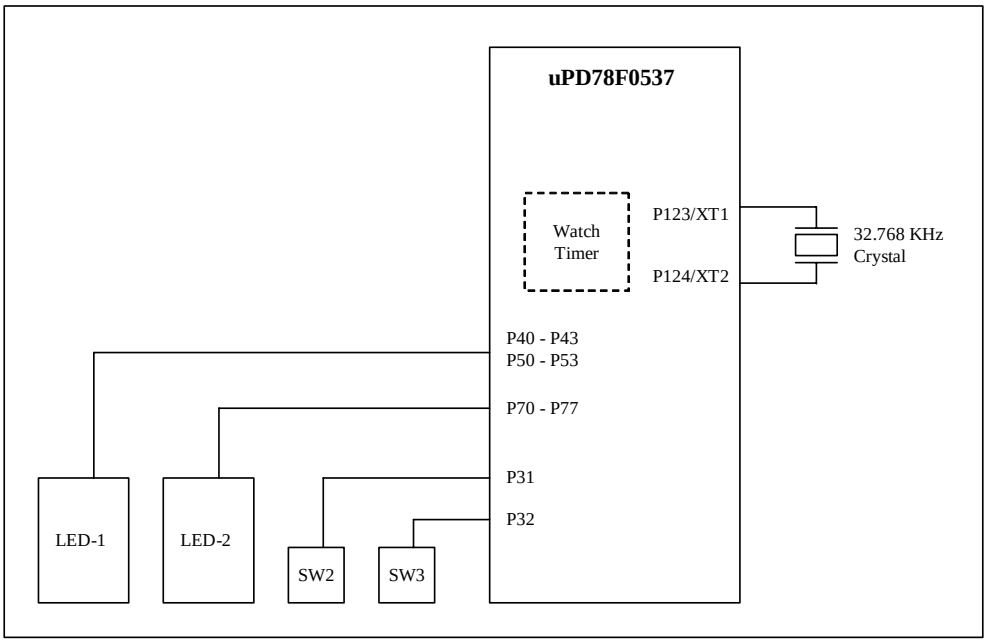
9.5.2 Demonstration of Program

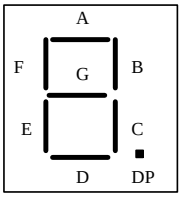
With the hardware configured and the uPD78F0537 microcontroller programmed with the demonstration code, the demonstration includes the following steps:

- Observe seconds counting up
- Press SW2 Observe seconds counting stop/start
- Press SW3 Observe seconds count reset to zero

9.6 Hardware Block Diagram

Figure 104. Demonstration Hardware Block Diagram





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

9.7 Software Modules in Demonstration Program

The following files make up the software modules for the demonstration program. The table shows which files are generated by the Applilet and which of those files need modification to create the demonstration program.

The listings for these files are located in the Appendix.

Table 26. Demonstration Software Modules

File	Purpose	Applilet Generated	User Modified
Main.c	Main program	Yes	Yes
Macrodriver.h	General definitions used by the Applilet	Yes	No
System.h	Clock-related definitions	Yes	No
Systeminit.c	SystemInit() and hdwinit() functions	Yes	No
System.c	Clock_Init() function	Yes	No
Watchtimer.h	Watch Timer-related definitions	Yes	Yes[Note 1]
Watchtimer.c	Watch Timer functions	Yes	No
Watchtimer_user.c	User code for timer interrupt handling	Yes	Yes
Port.h	Port-related definitions	Yes	No
Port.c	Port_Init() function	Yes	No
Led_0537.h	LED display definitions	--	Yes
Led_0537.c	LED display functions	--	Yes
Sw_0537.h	Switch input definitions	--	Yes
Sw_0537.c	Switch input functions	--	Yes
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

Note 1: The file watchtimer.h must be modified to add declarations of variables used in timekeeping; these variables are defined in watchtimer_user.c.

10. Appendix A — Development Tools

The following software and hardware tools were used in the development of this application note.

10.1 Software Tools

Table 27. Software Tools

Tool	Version	Comments
Applilet for 78K0KE2	V1.51	Source-code generation tool for 78K0/KE2 devices
PM Plus	V5.21	Project manager for program compilation and linking
CC78K0	V3.70	C Compiler for NEC Electronics 78K0 devices
RA78K0	V3.80	Assembler for NEC Electronics 78K0 devices
DF053764.78K	V2.00	Device file for uPD78F0537_64 device

10.2 Hardware Tools

Table 28. Hardware Tools

Tool	Version	Comments
M-Station 2	V2.1E	Base platform for NEC Electronics' Micro-board demonstration
M-78F0537	V1.0	NEC Electronics' Micro-board for uPD78F0537; CPU chip is uPD78F0537DGB

11. Appendix B — Software Listings

The demonstration programs described in this application note were all developed on the same hardware platform using the Applilet program generation tool. Many of the files that make up the programs are identical across all of the programs.

To avoid duplication, the listings in the next eight sections cover files that are unique to each demonstration program. These files are typically the following:

- Main.c—The main program
- Systeminit.c—Code for initializing peripherals, which may differ from one program to another
- Timer.h—Headers and code for timers, which differ from one program to another
- Timer.c
- Timer_user.c

In the case of the watch-timer demonstration program, the timer.h, timer.c and timer_user.c files are replaced by watchtimer.h, watchtimer.c, and watchtimer_user.c.

The final section of this appendix provides listings for files that are common to all or most of the programs.

11.1 Interval Timer Using 16-Bit Timer 00 (TM00)

11.1.1 Main.c

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

```
**
************************************************************************
*/
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"

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

/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void main( void )
{
unsigned char count;       /* count to display */
unsigned char sw_val;      /* value of switches */

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       count = 0;          /* set initial count */

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


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

       while(1){
              led_dig(count);           /* display count in LEDs as hex number */

              sw_val = sw_get();  /* get debounced switch value */

              if (sw_val != SW_LU_RU) {
                     /* not both up, so one or both are down */
                     if (sw_val == SW_LD_RU) {
                            /* SW2 (left) is down, right (SW3) is up */
                            count = count - 1;  /* decrement the count */
```

```
                        led_dig(count);            /* display it */
                        /* wait for switches to change */
                        do {
                              sw_val = sw_get();
                        } while (sw_val == SW_LD_RU);
                  }

                  if (sw_val == SW_LU_RD) {
                        /* SW2 up, SW3 down */
                        count = count + 1;  /* increment the count */
                        led_dig(count);            /* display it */
                        /* wait for switches to change */
                        do {
                              sw_val = sw_get();
                        } while (sw_val == SW_LU_RD);
                  }

                  if (sw_val == SW_LD_RD) {
                        /* both SW2 and SW3 are down */
                        count = 0;                 /* reset count */
                        led_dig(count);            /* display it */
                        /* wait for switches to be both up */
                        do {
                              sw_val = sw_get();
                        } while (sw_val != SW_LU_RU);
                  }
            } /* end of if (switch down) */
      } /* end of while (1) loop */
} /* end of main() */
```

11.1.2 Systeminit.c

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

```
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
/*
*******************************************************************************
**  MacroDefine
*******************************************************************************
*/

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

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

11.1.3 Timer.h

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

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

#ifndef _MDTIMER_
#define _MDTIMER_
/*
*******************************************************************************
**  MacroDefine
*******************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x4e1f
#define      TM_TM00_SQUAREWIDTH 0x4e1f
#define      TM_TM00_PPGCYCLE    0x4e1f
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x4e1f
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x00
#define      TM_TM01_SQUAREWIDTH 0x00
#define      TM_TM01_PPGCYCLE    0x00
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE       0x00
#define      TM_TM01_ONEPULSEDELAY      0x00
#define      TM_TM50_CLOCK 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x00
#define      TM_TM51_SQUAREWIDTH 0x00
#define      TM_TM51_PWMACTIVEVALUE     0x00
#define      TM_TMH0_CLOCK 0x0
#define      TM_TMH0_INTERVALVALUE      0x00
#define      TM_TMH0_SQUAREWIDTH 0x00
#define      TM_TMH0_PWMCYCLE    0x00
#define      TM_TMH0_PWMDELAY    0x00
#define      TM_TMH1_CLOCK 0x0
#define      TM_TMH1_INTERVALVALUE      0x00
#define      TM_TMH1_SQUAREWIDTH 0x00
#define      TM_TMH1_PWMCYCLE    0x00
#define      TM_TMH1_PWMDELAY    0x00
#define      TM_TMH1_CARRIERDELAY       0x00
#define      TM_TMH1_CARRIERWIDTH       0x00
```

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

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

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

#endif       /* _MDTIMER_*/
```

11.1.4 Timer.c

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

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


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

/*TM01 pulse width measure*/

/*
```

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

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**     This function stops the TM00 counter and clear the count register.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**---------------------------------------------------------------------------
*/
void TM00_Stop()
{
       TMC00=0x0;
       SetIORBit(MK0H, 0x40);                 /* INTTM000 stop */
}
```

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

11.1.5 Timer_user.c

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

#pragma sfr
#pragma interrupt  INTTM000 MD_INTTM000
/*
************************************************************************
```

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

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

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

/* Timer00, Timer01 pulse width measure */
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     TM00 INTTM000 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
       /* call switch ISR routine for debouncing switches */
       sw_isr();
}
```

11.2 External Event Counter Using 16-Bit Timer 00 (TM00)

11.2.1 Main.c

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

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

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

/*
************************************************************************
**  Global Variables
************************************************************************
*/
unsigned char count;       /* count to display */

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

```
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**--------------------------------------------------------------------------
*/
void main( void )
{
unsigned char sw_val;      /* value of switches */

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

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


       TMH0_Start();/* start timer for switch debouncing */

       count = 0;          /* set count initial value and display */
       led_dig(count);

       TM00_Start();/* start External Event Counter */

       TM50_Start();/* start Square Wave Generator */

       while(1){
              sw_val = sw_get();  /* get switch value */

              if (sw_val == SW_LD_RU) {
                     /* SW2 (left) is down, right (SW3) is up */
                     TM50_Stop();  /* stop timer */
                     /* set timer for half of original period, so double original frequency
(100 Hz) */
                     TM50_ChangeTimerCondition(TM_TM50_SQUAREWIDTH / 2);
                     TM50_Start();/* restart the timer */
                     while (SW_LU_RU != sw_get())
                            ;                   /* wait for switches to be up */
              }

              if (sw_val == SW_LU_RD) {
                     /* SW2 (left) is up, right (SW3) is down */
                     TM50_Stop();  /* stop timer */
                     /* set timer for original period, so original frequency (50 Hz) */
                     TM50_ChangeTimerCondition(TM_TM50_SQUAREWIDTH);
                     TM50_Start();/* restart the timer */
                     while (SW_LU_RU != sw_get())
                            ;                   /* wait for switches to be up */
              }

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

11.2.2 Systeminit.c

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

/*
**------------------------------------------------------------------------
**
**  Abstract:
**     Init every Macro
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* Clock generator initiate */
       Clock_Init();
       /* Port initiate */
       PORT_Init();
       /* TM00 initiate */
       TM00_Init();
```

```
      /* TM50 initiate */
      TM50_Init();
      /* TMH0 initiate */
      TMH0_Init();
}

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

11.2.3 Timer.h

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

#ifndef _MDTIMER_
#define _MDTIMER_
/*
************************************************************************
**  MacroDefine
************************************************************************
*/
```

```
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x00
#define      TM_TM00_SQUAREWIDTH 0x00
#define      TM_TM00_PPGCYCLE    0x00
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x00
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x00
#define      TM_TM01_SQUAREWIDTH 0x00
#define      TM_TM01_PPGCYCLE    0x00
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE       0x00
#define      TM_TM01_ONEPULSEDELAY      0x00
#define      TM_TM50_CLOCK 0x7
#define      TM_TM50_INTERVALVALUE      0x17
#define      TM_TM50_SQUAREWIDTH 0x17
#define      TM_TM50_PWMACTIVEVALUE     0x17
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x00
#define      TM_TM51_SQUAREWIDTH 0x00
#define      TM_TM51_PWMACTIVEVALUE     0x00
#define      TM_TMH0_CLOCK 0x4
#define      TM_TMH0_INTERVALVALUE      0x12
#define      TM_TMH0_SQUAREWIDTH 0x12
#define      TM_TMH0_PWMCYCLE    0x12
#define      TM_TMH0_PWMDELAY    0x00
#define      TM_TMH1_CLOCK 0x0
#define      TM_TMH1_INTERVALVALUE      0x00
#define      TM_TMH1_SQUAREWIDTH 0x00
#define      TM_TMH1_PWMCYCLE    0x00
#define      TM_TMH1_PWMDELAY    0x00
#define      TM_TMH1_CARRIERDELAY       0x00
#define      TM_TMH1_CARRIERWIDTH       0x00
#define      TM_TM00_EXTERNALCOUNT      0x31


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM00_Init(void);
void TM50_Init(void);
void TMH0_Init(void);

/*timer start*/
void TM00_Start(void);
void TM50_Start(void);
void TMH0_Start(void);

/*timer stop*/
void TM00_Stop(void);
void TM50_Stop(void);
void TMH0_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TM50_ChangeTimerCondition(UCHAR value);
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num);
__interrupt void MD_INTTM000(void);
__interrupt void MD_INTTMH0(void);

#endif       /* _MDTIMER_*/
```

11.2.4 Timer.c

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

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


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

/*TM01 pulse width measure*/

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TM00_module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void TM00_Init( )
{
       TMC00=0x00;
       SetIORBit(PR0H, 0x40);                  /* low priority level */
```

```
      ClrIORBit(IF0H, 0x40);
      /* TM00 external event counter */
      ClrIORBit(CRC00,0x01);
      SetIORBit(PRM00,0x03);
      CR000 = TM_TM00_EXTERNALCOUNT;
      CR010 = 0xffff;

      SetIORBit(PM0,0x1);             /* TI000(P00) input */
      ClrIORBit(ISC,0x2);
      ClrIORBit(PRM00,0x30);                /* falling edge of TI000 */
}
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function starts the TM00 counter.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TM00_Start()
{
      TMC00 = 0x0c;                   /* external event count start */
      ClrIORBit(MK0H, 0x40);                /* INTTM000 enable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function stops the TM00 counter and clear the count register.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TM00_Stop()
{
      TMC00=0x0;
      SetIORBit(MK0H, 0x40);                /* INTTM000 stop */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function changes TM00 condition.
**
**  Parameters:
**    USHORT* :    array_reg
**    USHORT :     array_num
**  Returns:
**    MD_OK
**    MD_ERROR
```

```
**
**-------------------------------------------------------------------------
*/
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
   switch (array_num){
         case 2:
                  CR010=*(array_reg + 1);
         case 1:
                  CR000=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
         return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can initialize TM50_module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM50_Init( void )
{
       ClrIORBit(TMC50, 0x80);
       TCL50 = TM_TM50_CLOCK;                  /* internal countclock */
       /* TM50 squarewave output */
       ClrIORBit(P1, 0x80);
       ClrIORBit(PM1, 0x80);                   /* TO50 output */
       SetIORBit(TMC50, 0xb);
       CR50 = TM_TM50_SQUAREWIDTH;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can start the TM50 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM50_Start( void )
{
       /* TM50 squarewave output */
       SetIORBit(TMC50, 0x8b);
}

/*
```

```
**------------------------------------------------------------------------------
**
**  Abstract:
**     This function can stop the TM50 counter and clear the count register.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**------------------------------------------------------------------------------
*/
void TM50_Stop( void )
{
       ClrIORBit(TMC50, 0x80);
}

/*
**------------------------------------------------------------------------------
**
**  Abstract:
**     This function can change TM50 condition.
**
**  Parameters:
**     UCHAR :         value
**  Returns:
**     MD_OK
**     MD_ERROR
**
**------------------------------------------------------------------------------
*/
MD_STATUS TM50_ChangeTimerCondition(UCHAR value)
{
       CR50 =value;
       return MD_OK;
}

/*
**------------------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TMH0 module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**------------------------------------------------------------------------------
*/
void TMH0_Init( void )
{
       ClrIORBit(TMHMD0, 0x80);
                     /* countclock=fx/1024 */
       TMHMD0 |= (TM_TMH0_CLOCK << 4);

       SetIORBit(PR0H, 0x10);                  /* low priority level */
       ClrIORBit(IF0H, 0x10);
       /* TMH0 interval timer */
       ClrIORBit(TMHMD0, 0x8c);
       CMP00 = TM_TMH0_INTERVALVALUE;
```

```
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can start the TMH0 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Start( void )
{
       SetIORBit(TMHMD0, 0x80);
       ClrIORBit(MK0H, 0x10);                  /* INTTMH0 enable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can stop the TMH0 counter operation.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Stop( void )
{
       ClrIORBit(TMHMD0, 0x80);
       SetIORBit(MK0H, 0x10);                  /* INTTMH0 disable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can change TMH0 condition.
**
**  Parameters:
**     UCHAR* :     array_reg
**     UCHAR :      array_num
**  Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num)
{
   switch (array_num){
         case 2:
                  CMP10=*(array_reg + 1);
         case 1:
```

```
                CMP00=*(array_reg + 0);
                break;
        default:
                return MD_ERROR;
        }
        return MD_OK;
}
```

11.2.5 Timer_user.c

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

#pragma sfr
#pragma interrupt  INTTM000 MD_INTTM000
#pragma interrupt  INTTMH0 MD_INTTMH0
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

/* add includes for switch routines and LED routines */
#include "sw_0537.h"
#include "led_0537.h"

/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/* define external global variable count so ISR can increment it */
extern unsigned char count;

/* Timer00, Timer01 pulse width measure */
/*
**------------------------------------------------------------------------
```

```
**
**  Abstract:
**     TM00 INTTM000 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**--------------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
       count = count + 1;
       led_dig(count);
}
/*
**--------------------------------------------------------------------------
**
**  Abstract:
**     TMH0 INTTMH0 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**--------------------------------------------------------------------------
*/
__interrupt void MD_INTTMH0( )
{
       /* call switch ISR routine for debouncing switches */
       sw_isr();
}
```

11.3 One-Shot Pulse Generation Using 16-Bit Timer 01 (TM01)

11.3.1 Main.c

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

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

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

/*
************************************************************************
**  Global Variables
************************************************************************
*/
unsigned char count;       /* count to display */

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

```
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void main( void )
{
unsigned char sw_val;      /* value of switches */
int i;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

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


       TMH0_Start();/* start timer for switch debouncing */

       count = 0;          /* set count initial value and display */
       led_dig(count);

       TM00_Start();/* start External Event Counter */

       TM01_Start();/* get One-Shot pulse generator ready */

       while(1){
              sw_val = sw_get();  /* get switch value */

              if (sw_val == SW_LD_RU) {
                     /* SW2 (left) is down, right (SW3) is up */
                     /* create a single pulse */
                     TM01_OneshotTriggerOn();
                     while (SW_LU_RU != sw_get())
                            ;                   /* wait for switches to be up */
              }

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

11.3.2 Systeminit.c

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

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

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     Init every Macro
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* Clock generator initiate */
       Clock_Init();
       /* Port initiate */
       PORT_Init();
       /* TM00 initiate */
       TM00_Init();
       /* TM01 initiate */
       TM01_Init();
       /* TMH0 initiate */
       TMH0_Init();
}

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

```
**
**  Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void hdwinit( void )
{
       DI( );
       SystemInit( );
       EI( );
}
```

11.3.3 Timer.h

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

#ifndef _MDTIMER_
#define _MDTIMER_
/*
*******************************************************************************
**  MacroDefine
*******************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x00
#define      TM_TM00_SQUAREWIDTH 0x00
#define      TM_TM00_PPGCYCLE    0x00
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x00
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x7cf
#define      TM_TM01_SQUAREWIDTH 0x7cf
#define      TM_TM01_PPGCYCLE    0x7cf
#define      TM_TM01_PPGWIDTH    0x3e7
#define      TM_TM01_ONESHOTCYCLE       0x7cf
```

```
#define      TM_TM01_ONEPULSEDELAY      0x3e7
#define      TM_TM50_CLOCK 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x00
#define      TM_TM51_SQUAREWIDTH 0x00
#define      TM_TM51_PWMACTIVEVALUE     0x00
#define      TM_TMH0_CLOCK 0x4
#define      TM_TMH0_INTERVALVALUE      0x12
#define      TM_TMH0_SQUAREWIDTH 0x12
#define      TM_TMH0_PWMCYCLE    0x12
#define      TM_TMH0_PWMDELAY    0x00
#define      TM_TMH1_CLOCK 0x0
#define      TM_TMH1_INTERVALVALUE      0x00
#define      TM_TMH1_SQUAREWIDTH 0x00
#define      TM_TMH1_PWMCYCLE    0x00
#define      TM_TMH1_PWMDELAY    0x00
#define      TM_TMH1_CARRIERDELAY       0x00
#define      TM_TMH1_CARRIERWIDTH       0x00
#define      TM_TM00_EXTERNALCOUNT      0x4


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM00_Init(void);
void TM01_Init(void);
void TMH0_Init(void);

/*timer start*/
void TM00_Start(void);
void TM01_Start(void);
void TMH0_Start(void);

/*timer stop*/
void TM00_Stop(void);
void TM01_Stop(void);
void TMH0_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TM01_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
void TM01_OneshotTriggerOn();
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num);
__interrupt void MD_INTTM000(void);
__interrupt void MD_INTTMH0(void);

#endif       /* _MDTIMER_*/
```

11.3.4 Timer.c

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

```
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename : timer.c
**  Abstract : This file implements a device driver for the timer module
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :  uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
*************************************************************************
*/

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


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

/*TM01 pulse width measure*/

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TM00_module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void TM00_Init( )
{
       TMC00=0x00;
       SetIORBit(PR0H, 0x40);                  /* low priority level */
       ClrIORBit(IF0H, 0x40);
       /* TM00 external event counter */
       ClrIORBit(CRC00,0x01);
       SetIORBit(PRM00,0x03);
       CR000 = TM_TM00_EXTERNALCOUNT;
       CR010 = 0xffff;

       SetIORBit(PM0,0x1);              /* TI000(P00) input */
       ClrIORBit(ISC,0x2);
       ClrIORBit(PRM00,0x30);                  /* falling edge of TI000 */
}
/*
**-----------------------------------------------------------------------
**
```

```
**  Abstract:
**     This function starts the TM00 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void TM00_Start()
{
       TMC00 = 0x0c;                    /* external event count start */
       ClrIORBit(MK0H, 0x40);                  /* INTTM000 enable */
}

/*
**----------------------------------------------------------------------------
**
**  Abstract:
**     This function stops the TM00 counter and clear the count register.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void TM00_Stop()
{
       TMC00=0x0;
       SetIORBit(MK0H, 0x40);                  /* INTTM000 stop */
}

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

```
        return MD_OK;
}
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function can generate a One-shot pulse output trigger via software.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TM01_OneshotTriggerOn()
{
      SetIORBit(TOC01, 0x40);
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function can initialize TM01_module.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TM01_Init( )
{
      TMC01=0x00;
                                    /* internal count clock */
      PRM01 |= TM_TM01_CLOCK;
      /*TM01 oneshot pulse output*/
      ClrIORBit(CRC01, 0x05);               /* used as compare register */

      CR001 = TM_TM01_ONESHOTCYCLE;
      CR011 = TM_TM01_ONEPULSEDELAY;

      ClrIORBit(P0,0x40);             /* TO01(p06) as output */
      ClrIORBit(PM0,0x40);
      TOC01 = 0x37;                   /* initial low */
      ClrIORBit(TOC01, 0x40);               /* software trigger */
}
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function start the TM01 counter.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
```

```
**-------------------------------------------------------------------------
*/
void TM01_Start(void)
{
      TMC01 = 0x04;             /* one shot pulse output start */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This fnction stop the TM01 module.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TM01_Stop(void)
{
      TMC01=0x0;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function can change TM01 condition.
**
**  Parameters:
**    USHORT* :    array_reg
**    USHORT :     array_num
**  Returns:
**    MD_OK
**    MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TM01_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
   switch (array_num){
         case 2:
                  CR011=*(array_reg + 1);
         case 1:
                  CR001=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
        return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function initializes TMH0 module.
**
**  Parameters:
**    None
```

```
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Init( void )
{
       ClrIORBit(TMHMD0, 0x80);
                    /* countclock=fx/1024 */
       TMHMD0 |= (TM_TMH0_CLOCK << 4);

       SetIORBit(PR0H, 0x10);                  /* low priority level */
       ClrIORBit(IF0H, 0x10);
       /* TMH0 interval timer */
       ClrIORBit(TMHMD0, 0x8c);
       CMP00 = TM_TMH0_INTERVALVALUE;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can start the TMH0 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Start( void )
{
       SetIORBit(TMHMD0, 0x80);
       ClrIORBit(MK0H, 0x10);                  /* INTTMH0 enable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can stop the TMH0 counter operation.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Stop( void )
{
       ClrIORBit(TMHMD0, 0x80);
       SetIORBit(MK0H, 0x10);                  /* INTTMH0 disable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
```

```
**     This function can change TMH0 condition.
**
**  Parameters:
**     UCHAR* :      array_reg
**     UCHAR :       array_num
**  Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num)
{
   switch (array_num){
         case 2:
                  CMP10=*(array_reg + 1);
         case 1:
                  CMP00=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
         return MD_OK;
}
```

11.3.5 Timer_user.c

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

#pragma sfr
#pragma interrupt  INTTM000 MD_INTTM000
#pragma interrupt  INTTMH0 MD_INTTMH0
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
```

```
#include "timer.h"

/* add includes for switch routines and LED routines */
#include "sw_0537.h"
#include "led_0537.h"

/*
*****************************************************************************
**  MacroDefine
*****************************************************************************
*/
/* define external global variable count so ISR can increment it */
extern unsigned char count;

/* Timer00, Timer01 pulse width measure */
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     TM00 INTTM000 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
       count = count + 1;
       led_dig(count);
}
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     TMH0 INTTMH0 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTTMH0( )
{
       /* call switch ISR routine for debouncing switches */
       sw_isr();
}
```

11.4 PWM Output For D/A Conversion Using 8-Bit Timer 51 (TM51)

11.4.1 Main.c

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

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

/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/*
**------------------------------------------------------------------------
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
```

```
**     None
**
**------------------------------------------------------------------------------
*/
void main( void )
{
int i;
USHORT value;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       led_init();         /* initialize LEDs */

       TM51_Start();/* start PWM with default value */

       AD_Start();         /* start A/D converter */

       while(1){
              /* get 10-bit A/D converter value */
              AD_Read(&value);

              /* shift down to 8-bit value, most significant bits, and display */
              value = value >> 2;
              led_dig((UCHAR)(value & 0xFF));

              /* Set as PWM width */
              /* delay for a short time between settings of CR51 in PWM */
              /* must be 3 or more count clocks */
              /* since TM51 PWM is running at fPRS, no actual delay is necessary */
              /* but this is here as a reminder in case PWM clock is lower */
              for (i=0; i < 10; i++)
                     ;      /* short delay */
              TM51_ChangeTimerCondition((UCHAR)(value & 0xFF));

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

11.4.2 Systeminit.c

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

```
**
************************************************************************************
*/
/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "ad.h"
#include "port.h"
#include "timer.h"
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**     Init every Macro
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* Clock generator initiate */
       Clock_Init();
       /* Port initiate */
       PORT_Init();
       /* AD initiate */
       AD_Init();
       /* TM51 initiate */
       TM51_Init();
}

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

```
    EI( );
}
```

11.4.3 Timer.h

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

#ifndef _MDTIMER_
#define _MDTIMER_
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x00
#define      TM_TM00_SQUAREWIDTH 0x00
#define      TM_TM00_PPGCYCLE    0x00
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x00
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x00
#define      TM_TM01_SQUAREWIDTH 0x00
#define      TM_TM01_PPGCYCLE    0x00
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE       0x00
#define      TM_TM01_ONEPULSEDELAY      0x00
#define      TM_TM50_CLOCK 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x77
#define      TM_TM51_SQUAREWIDTH 0x77
#define      TM_TM51_PWMACTIVEVALUE     0x77
#define      TM_TMH0_CLOCK 0x0
```

```
#define       TM_TMH0_INTERVALVALUE      0x00
#define       TM_TMH0_SQUAREWIDTH 0x00
#define       TM_TMH0_PWMCYCLE    0x00
#define       TM_TMH0_PWMDELAY    0x00
#define       TM_TMH1_CLOCK 0x0
#define       TM_TMH1_INTERVALVALUE      0x00
#define       TM_TMH1_SQUAREWIDTH 0x00
#define       TM_TMH1_PWMCYCLE    0x00
#define       TM_TMH1_PWMDELAY    0x00
#define       TM_TMH1_CARRIERDELAY       0x00
#define       TM_TMH1_CARRIERWIDTH       0x00


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM51_Init(void);

/*timer start*/
void TM51_Start(void);

/*timer stop*/
void TM51_Stop(void);
MD_STATUS TM51_ChangeTimerCondition(UCHAR value);

#endif       /* _MDTIMER_*/
```

11.4.4 Timer.c

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

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

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

/*TM01 pulse width measure*/

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**     This function Initializes TM51_module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM51_Init( void )
{
       ClrIORBit(TMC51, 0x80);
       TCL51 = TM_TM51_CLOCK;                  /* countclock=fx */
       /* TM51 PWM output */
       ClrIORBit(P3, 0x08);
       ClrIORBit(PM3, 0x08);                   /* TO51 output */
       SetIORBit(TMC51, 0x1);
       SetIORBit(TMC51, 0x40);
       ClrIORBit(TMC51, 0x8);
       SetIORBit(TMC51, 0x4);
       ClrIORBit(TMC51, 0x2);                  /* active high */
       CR51 = TM_TM51_PWMACTIVEVALUE;
}

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**     This function starts the TM51 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM51_Start( void )
{
       /* TM51 PWM output */
       ClrIORBit(TMC51, 0x8);
       SetIORBit(TMC51, 0x4);
       SetIORBit(TMC51, 0x80);
}

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

```
**  Abstract:
**     This function stops the TM51 counter and clear the count register.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void TM51_Stop( void )
{
       ClrIORBit(TMC51, 0x80);
}

/*
**----------------------------------------------------------------------------
**
**  Abstract:
**     This function can change TM51 condition.
**
**  Parameters:
**     UCHAR :       value
**  Returns:
**     MD_OK
**     MD_ERROR
**
**----------------------------------------------------------------------------
*/
MD_STATUS TM51_ChangeTimerCondition(UCHAR value)
{
       CR51 =value;
       return MD_OK;
}
```

11.4.5 Timer_user.c

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

```
*/

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


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

/* Timer00, Timer01 pulse width measure */
```

11.5 Carrier Generation for Remote Control Using TM51 And TMH1

11.5.1 Main.c

```
/*  main.c for NEC Timer Application Note, Carrier Generation Section */
/*
************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename : main.c
**  Abstract : This file implements main function
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :  uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
************************************************************************
*/
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "timer.h"
/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/*
**------------------------------------------------------------------------
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**------------------------------------------------------------------------
*/
void main( void )
{
```

```
UCHAR bval[2];
int i;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       /* for debug, make P75 - P77 outputs */
       PM7 &= 0x1F;
       P7 = 0x00;

       while(1){
              P7 = 0x00;          /* return to zero for P76 and P75 */
              P7 ^= 0x80;         /* make trigger pulse on P77 at start */
              for (i=0; i<25; i++)
                     ;
              P7 ^= 0x80;

              ByteValue = 0xAA;   /* bit 0 = 0, bit 1 = 1 */
              BitCount  = 2;             /* send out two bits (0 then 1) */
              SendDone  = 0;

              /* set up TimerH1 values before enable */
              bval[0] = TM_TMH1_CARRIERDELAY;
              bval[1] = TM_TMH1_CARRIERWIDTH;
              TMH1_ChangeTimerCondition(bval, 1);
              TMH1_ChangeTimerCondition(bval, 2);

              TMH1_CarrierOutputEnable();
              TMH1_Start();

              TM51_ChangeTimerCondition(2);    /* set low value to first interrupt */
              TM51_Start();

              while (SendDone == 0)
                     ;
              TM51_Stop();
              TMH1_Stop();
       }
}
```

11.5.2 Systeminit.c

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

```
**
**  Compiler :      NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  Include files
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "timer.h"
/*
************************************************************************************
**  MacroDefine
************************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**     Init every Macro
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* Clock generator initiate */
       Clock_Init();
       /* TM51 initiate */
       TM51_Init();
       /* TMH1 initiate */
       TMH1_Init();
}

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

11.5.3 Timer.h

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

#ifndef _MDTIMER_
#define _MDTIMER_
/*
************************************************************************
**  MacroDefine
************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x00
#define      TM_TM00_SQUAREWIDTH 0x00
#define      TM_TM00_PPGCYCLE    0x00
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x00
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x00
#define      TM_TM01_SQUAREWIDTH 0x00
#define      TM_TM01_PPGCYCLE    0x00
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE       0x00
#define      TM_TM01_ONEPULSEDELAY      0x00
#define      TM_TM50_CLOCK 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x6
#define      TM_TM51_INTERVALVALUE      0x82
#define      TM_TM51_SQUAREWIDTH 0x82
#define      TM_TM51_PWMACTIVEVALUE     0x82
#define      TM_TMH0_CLOCK 0x0
#define      TM_TMH0_INTERVALVALUE      0x00
#define      TM_TMH0_SQUAREWIDTH 0x00
```

```
#define      TM_TMH0_PWMCYCLE    0x00
#define      TM_TMH0_PWMDELAY    0x00
#define      TM_TMH1_CLOCK 0x1
#define      TM_TMH1_INTERVALVALUE      0x59
#define      TM_TMH1_SQUAREWIDTH 0x59
#define      TM_TMH1_PWMCYCLE    0x59
#define      TM_TMH1_PWMDELAY    0x29
#define      TM_TMH1_CARRIERDELAY       0x59
#define      TM_TMH1_CARRIERWIDTH       0x29


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM51_Init(void);
void TMH1_Init(void);

/*timer start*/
void TM51_Start(void);
void TMH1_Start(void);

/*timer stop*/
void TM51_Stop(void);
void TMH1_Stop(void);
MD_STATUS TM51_ChangeTimerCondition(UCHAR value);
MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num);
void TMH1_CarrierOutputEnable();
void TMH1_CarrierOutputDisable();
__interrupt void MD_INTTM51(void);
__interrupt void MD_INTTMH1(void);

/* added global variables */
extern UCHAR ByteValue;
extern UCHAR BitCount;
extern UCHAR SendDone;

/* added defines for CR51 values */
#define TM_TM51_CR51_START 1
#define TM_TM51_CR51_HI          43
#define      TM_TM51_CR51_ZLO    43
#define      TM_TM51_CR51_OLO    130

#endif       /* _MDTIMER_*/
```

11.5.4 Timer.c

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

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

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


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

/*TM01 pulse width measure*/

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function Initializes TM51_module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM51_Init( void )
{
       ClrIORBit(TMC51, 0x80);
       TCL51 = TM_TM51_CLOCK;                  /* countclock=fx/256 */
       SetIORBit(PR1L, 0x08);                  /* low priority level */
       ClrIORBit(IF1L, 0x08);
       /* TM51 interval */
       CR51 = TM_TM51_INTERVALVALUE;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function starts the TM51 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
```

```
*/
void TM51_Start( void )
{
       /* TM51 interval */
       SetIORBit(TMC51, 0x80);
       ClrIORBit(MK1L, 0x08);                  /* INTTM51 enable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function stops the TM51 counter and clear the count register.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM51_Stop( void )
{
       ClrIORBit(TMC51, 0x80);
       SetIORBit(MK1L, 0x08);                  /* INTTM51 disable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can change TM51 condition.
**
**  Parameters:
**     UCHAR :       value
**  Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TM51_ChangeTimerCondition(UCHAR value)
{
       CR51 =value;
       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TMH1_module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH1_Init( void )
```

```
{
      ClrIORBit(TMHMD1, 0x80);
                  /* internal countclock */
      TMHMD1 |= (TM_TMH1_CLOCK << 4);

      SetIORBit(PR0H, 0x08);                  /* low priority level */
      ClrIORBit(IF0H, 0x08);
      /* TMH1 carrier generate output */
      SetIORBit(TMHMD1, 0x04);
      ClrIORBit(P1, 0x40);
      ClrIORBit(PM1, 0x40);                   /* P16/TOH1 */
      SetIORBit(TMHMD1, 0x1);
      SetIORBit(TMCYC1, 0x7);
      CMP01 = TM_TMH1_CARRIERDELAY;
      CMP11 = TM_TMH1_CARRIERWIDTH;
}

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

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

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can change TMH1 condition.
```

```
**
**  Parameters:
**    UCHAR* :     array_reg
**    UCHAR :      array_num
**  Returns:
**    MD_OK
**    MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num)
{
   switch (array_num){
         case 2:
                  CMP11=*(array_reg + 1);
         case 1:
                  CMP01=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
         return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function can enable carrier output.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TMH1_CarrierOutputEnable( void )
{
      SetIORBit(TMCYC1, 0x02);
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This functin can disable carrier output.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void TMH1_CarrierOutputDisable( void )
{
      ClrIORBit(TMCYC1, 0x02);
}
```

11.5.5 Timer_user.c

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

#pragma sfr
#pragma interrupt  INTTM51 MD_INTTM51
#pragma interrupt  INTTMH1 MD_INTTMH1
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

/* added global variables */
UCHAR ByteValue;
UCHAR BitCount;
UCHAR SendDone;

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

/* Timer00, Timer01 pulse width measure */
/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     TM51 INTTM51 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
```

```
*/
#define NRZB1 TMCYC1.1
__interrupt void MD_INTTM51( )
{
	/* TODO */
	P7 ^= 0x20;  /* flip P75 on INTTM51 */

	/* if the bit count is down to zero, we are done */
	if (BitCount == 0) {
		TMH1_CarrierOutputDisable();
		SendDone = 1;
		return;
	}

	/* wait until NRZ1 bit is changed to follow NRZB1 */
	while (NRZB1 != NRZ1)
		;
	if (NRZB1 == 1) {
		/* starting first (high) part of bit */
		NRZB1 = 0;
		CR51 = TM_TM51_CR51_HI;
	} else {
		/* NRZB1 == 0, starting second (low) part of bit */
		if ((ByteValue & 0x01) == 0x01) {
			/* LSB is 1, set timer for low width for 1 */
			CR51 = TM_TM51_CR51_OLO;
		} else {
			/* LSB is zero, set timer for low width for zero */
			CR51 = TM_TM51_CR51_ZLO;
		}
		ByteValue = ByteValue >> 1;	/* shift to next bit */
		BitCount--;  /* count down number of bits to do */
		if (BitCount != 0) {
			/* we have more bits to do, set NRZB1 to 1 for next time */
			NRZB1 = 1;
		} else {
			/* no more bits to do, set (leave) NRZB1 as 0 so NRZ1 doesn't go high
next time */
			NRZB1 = 0;
		}
	}
}
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     TMH1 INTTMH1 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTTMH1( )
{
	/* TODO */
	P7 ^= 0x40;		/* flip P76 on carrier edge */
}
```

11.6 PPG Output Tone Generation Using 16-Bit Timer 00 (TM00)

11.6.1 Main.c

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

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

/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/*
**------------------------------------------------------------------------
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
**     None
```

```
**
**-------------------------------------------------------------------------
*/
void main( void )
{
unsigned char sw_val;      /* value of switches */
USHORT crval[2];           /* array for setting timer compare registers */
USHORT cr000_value = TM_TM00_PPGCYCLE;  /* initial value for CR000 register */
USHORT cr010_value = TM_TM00_PPGWIDTH;  /* initial value for CR010 register */
USHORT temp;

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

       sw_init();          /* initialize switches */
       sw_set_debounce(10);       /* set debounce counter to 10 for 10 msec stable time */


       TMH0_Start();/* start timer for switch debouncing */

       TM00_Start();/* start initial tone generation */
       /* PPG set for 1704 us cycle time, TM_TM00_PPGCYCLE (586 Hz) */
       /* PPG set for 1136 us pulse width low time, TM_TM00_PPGWIDTH ( 1/2 440 Hz) */
       /* remainder is (1704 - 1136) 568 us high time ( 1/2 880 Hz) */
       /* This is 66.7/33.3 duty cycle */

       while(1){
              sw_val = sw_get();  /* get debounced switch value */

              if (sw_val == SW_LD_RU) {
                     /* SW2 (left) is down, right (SW3) is up */
                     TM00_Stop();  /* stop timer while changing PPG values */
                     temp = ((TM_TM00_PPGWIDTH * 2) / 3) + 1;
                     /* temp = pulse width for 1/2 440 Hz * 2/3 = pulse width for 1/2 660 Hz
*/
                     /* temp will be 3787 (0x0ECB), so time is 3787 * 0.2 us = 757.4 us */
                     cr000_value = TM_TM00_PPGWIDTH + temp;
                     /* 1/2 440 (1136 us) + 1/2 660 (757.4 us) = 1893.4 us = about 528 Hz */
                     cr010_value = TM_TM00_PPGWIDTH;               /* 1/2 440 (1136 us) */
                     crval[0] = cr000_value;
                     crval[1] = cr010_value;
                     TM00_ChangeTimerCondition(crval, 2);
                     /* PPG set for 1893.4 us cycle time (528 Hz) */
                     /* PPG set for 1136 us pulse width low time, 1/2 440 Hz */
                     /* remainder is (1893.4 - 1136) = 757.6 us high time, 1/2 660 Hz */
                     /* this is 60%/40% duty cycle */

                     TOC00 = 0x17;/* set for pin init low */
                     TM00_Start();/* restart timer */
                     while (SW_LU_RU != sw_get())
                            ;     /* wait for switches to be up */
              }

              if (sw_val == SW_LU_RD) {
                     /* SW2 up, SW3 down */
                     /* SW2 (left) is down, right (SW3) is up */
                     TM00_Stop();  /* stop timer while changing PPG values */
                     temp = ((TM_TM00_PPGWIDTH * 2) / 3) + 1;
                     /* temp = pulse width for 1/2 440 Hz * 2/3 = pulse width for 1/2 660 Hz
*/
                     /* temp will be 3787 (0x0ECB), so time is 3787 * 0.2 us = 757.4 us */
                     cr000_value = (((TM_TM00_PPGWIDTH + 1 ) / 2) - 1) + temp;
                     /* 1/2 880 (568 us) + 1/2 660 (757.4 us) = 1325.4 us = about 754 Hz */
```

```
                    cr010_value = ((TM_TM00_PPGWIDTH + 1) / 2) - 1;            /* 1/2 880
*/
                    crval[0] = cr000_value;
                    crval[1] = cr010_value;
                    TM00_ChangeTimerCondition(crval, 2);
                    /* PPG set for 1325.4 us cycle time (754 Hz) */
                    /* PPG set for 568 us pulse width low time, 1/2 880 Hz */
                    /* remainder is (1325.4 - 568) = 757.4 us high time, 1/2 660 Hz */
                    /* this is 42.8%/67.2% duty cycle */

                    TOC00 = 0x17;/* set for pin init low */
                    TM00_Start();/* restart timer */
                    while (SW_LU_RU != sw_get())
                           ;     /* wait for switches to be up */
             }

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

11.6.2 Systeminit.c

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

/*
```

```
**------------------------------------------------------------------------------
**
**  Abstract:
**     Init every Macro
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**------------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* Clock generator initiate */
       Clock_Init();
       /* Port initiate */
       PORT_Init();
       /* TM00 initiate */
       TM00_Init();
       /* TMH0 initiate */
       TMH0_Init();
}

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

11.6.3 Timer.h

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

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

#ifndef _MDTIMER_
#define _MDTIMER_
/*
*****************************************************************************
**  MacroDefine
*****************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x1
#define      TM_TM00_INTERVALVALUE      0x2147
#define      TM_TM00_SQUAREWIDTH 0x2147
#define      TM_TM00_PPGCYCLE    0x2147
#define      TM_TM00_PPGWIDTH    0x162f
#define      TM_TM00_ONESHOTCYCLE       0x2147
#define      TM_TM00_ONEPULSEDELAY      0x162f
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x00
#define      TM_TM01_SQUAREWIDTH 0x00
#define      TM_TM01_PPGCYCLE    0x00
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE       0x00
#define      TM_TM01_ONEPULSEDELAY      0x00
#define      TM_TM50_CLOCK 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x00
#define      TM_TM51_SQUAREWIDTH 0x00
#define      TM_TM51_PWMACTIVEVALUE     0x00
#define      TM_TMH0_CLOCK 0x4
#define      TM_TMH0_INTERVALVALUE      0x12
#define      TM_TMH0_SQUAREWIDTH 0x12
#define      TM_TMH0_PWMCYCLE    0x12
#define      TM_TMH0_PWMDELAY    0x00
#define      TM_TMH1_CLOCK 0x0
#define      TM_TMH1_INTERVALVALUE      0x00
#define      TM_TMH1_SQUAREWIDTH 0x00
#define      TM_TMH1_PWMCYCLE    0x00
#define      TM_TMH1_PWMDELAY    0x00
#define      TM_TMH1_CARRIERDELAY       0x00
#define      TM_TMH1_CARRIERWIDTH       0x00


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM00_Init(void);
void TMH0_Init(void);

/*timer start*/
void TM00_Start(void);
void TMH0_Start(void);
```

```
/*timer stop*/
void TM00_Stop(void);
void TMH0_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num);
__interrupt void MD_INTTMH0(void);

#endif       /* _MDTIMER_*/
```

11.6.4 Timer.c

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

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


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

/*TM01 pulse width measure*/

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TM00_module.
**
```

```
**  Parameters:
**    None
**
**  Returns:
**    None
**
**--------------------------------------------------------------------------
*/
void TM00_Init( )
{
      TMC00=0x00;
                                    /* internal count clock */
      PRM00 |= TM_TM00_CLOCK;
      /* TM00 PPG output function */
      ClrIORBit(CRC00,0x05);
      CR000 = TM_TM00_PPGCYCLE;
      CR010 = TM_TM00_PPGWIDTH;

      ClrIORBit(P0,0x2);            /* TO00(P01) output */
      ClrIORBit(PM0,0x2);
      TOC00 = 0x17;                 /* init low */
}
/*
**--------------------------------------------------------------------------
**
**  Abstract:
**    This function starts the TM00 counter.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**--------------------------------------------------------------------------
*/
void TM00_Start()
{
      TMC00 = 0x0c;                 /* PPG output start */
}

/*
**--------------------------------------------------------------------------
**
**  Abstract:
**    This function stops the TM00 counter and clear the count register.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**--------------------------------------------------------------------------
*/
void TM00_Stop()
{
      TMC00=0x0;
}

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

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

/*
**--------------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TMH0 module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**--------------------------------------------------------------------------
*/
void TMH0_Init( void )
{
       ClrIORBit(TMHMD0, 0x80);
                    /* countclock=fx/1024 */
       TMHMD0 |= (TM_TMH0_CLOCK << 4);

       SetIORBit(PR0H, 0x10);                  /* low priority level */
       ClrIORBit(IF0H, 0x10);
       /* TMH0 interval timer */
       ClrIORBit(TMHMD0, 0x8c);
       CMP00 = TM_TMH0_INTERVALVALUE;
}

/*
**--------------------------------------------------------------------------
**
**  Abstract:
**     This function can start the TMH0 counter.
**
**  Parameters:
**     None
**
**  Returns:
```

```
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Start( void )
{
       SetIORBit(TMHMD0, 0x80);
       ClrIORBit(MK0H, 0x10);                  /* INTTMH0 enable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can stop the TMH0 counter operation.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Stop( void )
{
       ClrIORBit(TMHMD0, 0x80);
       SetIORBit(MK0H, 0x10);                  /* INTTMH0 disable */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can change TMH0 condition.
**
**  Parameters:
**     UCHAR* :      array_reg
**     UCHAR :       array_num
**  Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num)
{
   switch (array_num){
         case 2:
                  CMP10=*(array_reg + 1);
         case 1:
                  CMP00=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
         return MD_OK;
}
```

11.6.5 Timer_user.c

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

#pragma sfr
#pragma interrupt  INTTMH0 MD_INTTMH0
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

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

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

/* Timer00, Timer01 pulse width measure */
/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     TMH0 INTTMH0 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
*/
__interrupt void MD_INTTMH0( )
{
       /* call switch ISR routine for debouncing switches */
       sw_isr();
```

```
}
```

11.7 Square-wave Tone Generation Using 16-Bit Timer 00 (TM00)

11.7.1 Main.c

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

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

/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
```

```
**  Returns:
**     None
**
**---------------------------------------------------------------------------
*/
void main( void )
{
unsigned char sw_val;      /* value of switches */
USHORT value;              /* input A/D value */
USHORT crval[2];           /* array for setting timer compare registers */
USHORT cr000_value = TM_TM00_SQUAREWIDTH;      /* initial value for CR000 register */

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;

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


       TMH0_Start();/* start timer for switch debouncing */

       TM00_Start();/* start initial tone generation */
       AD_Start();         /* start A/D converter */
       led_dig((UCHAR)(cr000_value >> 8));     /* display initial high byte of counter value
*/

       while(1){
              sw_val = sw_get();  /* get debounced switch value */

              if (sw_val == SW_LD_RU) {
                     /* SW2 (left) is down, right (SW3) is up */
                     while (SW_LD_RU == (sw_val = sw_get())) {
                            /* do this as long as SW2 is down */
                            AD_Read(&value);           /* get 10-bit A/D value */
                            value = value >> 2;        /* shift to 8-bit */
                            led_dig((UCHAR)(value & 0xFF));  /* display */
                            cr000_value = cr000_value & 0x00FF;     /* mask off top byte */
                            cr000_value |= (value << 8);            /* new top byte */
                            crval[0] = cr000_value;                       /* put in array
for setting */
                            TM00_ChangeTimerCondition(crval, 1);    /* set CR000 */
                     }
              }

              if (sw_val == SW_LU_RD) {
                     /* SW2 up, SW3 down */
                     while (SW_LU_RD == (sw_val = sw_get())) {
                            /* do this as long as SW3 is down */
                            AD_Read(&value);           /* get 10-bit A/D value */
                            value = value >> 2;        /* shift to 8-bit */
                            led_dig((UCHAR)(value & 0xFF));  /* display */
                            cr000_value = cr000_value & 0xFF00;     /* mask off low byte */
                            cr000_value |= value & 0xFF;            /* new low byte */
                            crval[0] = cr000_value;                       /* put in array
for setting */
                            TM00_ChangeTimerCondition(crval, 1);    /* set CR000 */
                     }
              }

              if (sw_val == SW_LD_RD) {
                     /* both SW2 and SW3 are down - no action */
              }
```

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

11.7.2 Systeminit.c

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

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**     Init every Macro
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void SystemInit( void )
{
```

```
      /* Clock generator initiate */
      Clock_Init();
      /* Port initiate */
      PORT_Init();
      /* AD initiate */
      AD_Init();
      /* TM00 initiate */
      TM00_Init();
      /* TMH0 initiate */
      TMH0_Init();
}

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

11.7.3 Timer.h

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

#ifndef _MDTIMER_
```

```
#define _MDTIMER_
/*
*****************************************************************************
**  MacroDefine
*****************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x1
#define      TM_TM00_INTERVALVALUE      0x162f
#define      TM_TM00_SQUAREWIDTH 0x162f
#define      TM_TM00_PPGCYCLE    0x162f
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x162f
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x00
#define      TM_TM01_SQUAREWIDTH 0x00
#define      TM_TM01_PPGCYCLE    0x00
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE       0x00
#define      TM_TM01_ONEPULSEDELAY      0x00
#define      TM_TM50_CLOCK 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x00
#define      TM_TM51_SQUAREWIDTH 0x00
#define      TM_TM51_PWMACTIVEVALUE     0x00
#define      TM_TMH0_CLOCK 0x4
#define      TM_TMH0_INTERVALVALUE      0x7a0
#define      TM_TMH0_SQUAREWIDTH 0x7a0
#define      TM_TMH0_PWMCYCLE    0x7a0
#define      TM_TMH0_PWMDELAY    0x00
#define      TM_TMH1_CLOCK 0x0
#define      TM_TMH1_INTERVALVALUE      0x00
#define      TM_TMH1_SQUAREWIDTH 0x00
#define      TM_TMH1_PWMCYCLE    0x00
#define      TM_TMH1_PWMDELAY    0x00
#define      TM_TMH1_CARRIERDELAY       0x00
#define      TM_TMH1_CARRIERWIDTH       0x00


/* timer00 to 01,50,51,H0,H1 configurator initiation */
void TM00_Init(void);
void TMH0_Init(void);

/*timer start*/
void TM00_Start(void);
void TMH0_Start(void);

/*timer stop*/
void TM00_Stop(void);
void TMH0_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num);
__interrupt void MD_INTTMH0(void);

#endif       /* _MDTIMER_*/
```

11.7.4 Timer.c

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

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


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

/*TM01 pulse width measure*/

/*
**------------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TM00_module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**------------------------------------------------------------------------
*/
void TM00_Init( )
{
       TMC00=0x00;
                                       /* internal count clock */
```

```
       PRM00 |= TM_TM00_CLOCK;
       /* TM00 squarewave output */
       ClrIORBit(CRC00,0x01);
       CR000 = TM_TM00_SQUAREWIDTH;
       CR010 = 0xffff;

       ClrIORBit(P0,0x2);              /* TO00(P01) output */
       ClrIORBit(PM0,0x2);
       TOC00=0x7;                      /* init low */
}
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function starts the TM00 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Start()
{
       TMC00 = 0x0c;                   /* squarewave output start */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function stops the TM00 counter and clear the count register.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TM00_Stop()
{
       TMC00=0x0;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function changes TM00 condition.
**
**  Parameters:
**     USHORT* :     array_reg
**     USHORT :      array_num
**  Returns:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
```

```
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
   switch (array_num){
         case 2:
                  CR010=*(array_reg + 1);
         case 1:
                  CR000=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
         return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function initializes TMH0 module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Init( void )
{
       ClrIORBit(TMHMD0, 0x80);
                     /* countclock=fx/1024 */
       TMHMD0 |= (TM_TMH0_CLOCK << 4);

       SetIORBit(PR0H, 0x10);                  /* low priority level */
       ClrIORBit(IF0H, 0x10);
       /* TMH0 interval timer */
       ClrIORBit(TMHMD0, 0x8c);
       CMP00 = TM_TMH0_INTERVALVALUE;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can start the TMH0 counter.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void TMH0_Start( void )
{
       SetIORBit(TMHMD0, 0x80);
       ClrIORBit(MK0H, 0x10);                  /* INTTMH0 enable */
}

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

```
**
**  Abstract:
**     This function can stop the TMH0 counter operation.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**----------------------------------------------------------------------------
*/
void TMH0_Stop( void )
{
       ClrIORBit(TMHMD0, 0x80);
       SetIORBit(MK0H, 0x10);                  /* INTTMH0 disable */
}

/*
**----------------------------------------------------------------------------
**
**  Abstract:
**     This function can change TMH0 condition.
**
**  Parameters:
**     UCHAR* :      array_reg
**     UCHAR :       array_num
**  Returns:
**     MD_OK
**     MD_ERROR
**
**----------------------------------------------------------------------------
*/
MD_STATUS TMH0_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num)
{
   switch (array_num){
         case 2:
                  CMP10=*(array_reg + 1);
         case 1:
                  CMP00=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
        return MD_OK;
}
```

11.7.5 Timer_user.c

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

```
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename : timer_user.c
**  Abstract : This file implements a device driver for the timer module
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :  uPD78F0537
**
**  Compiler:  NEC/CC78K0
**
************************************************************************
*/

#pragma sfr
#pragma interrupt  INTTMH0 MD_INTTMH0
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

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

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

/* Timer00, Timer01 pulse width measure */
/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     TMH0 INTTMH0 interrupt service routine
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
*/
__interrupt void MD_INTTMH0( )
{
       /* call switch ISR routine for debouncing switches */
       sw_isr();
}
```

11.8 Timekeeping with the Watch Timer (WTM)

11.8.1 Main.c

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

/* added to Applilet-generated includes */
#include "sw_0537.h"       /* switch functions for M-Station */
#include "led_0537.h"      /* LED functions for M-Station */

/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/*
**------------------------------------------------------------------------
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
```

```
**     None
**
**------------------------------------------------------------------------------
*/
void main( void )
{
unsigned char sw_val;
unsigned char last_sec;

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

       led_init();         /* initialize LEDs */
       sw_init();          /* initialize switches */
       sw_set_debounce(10);       /* set debounce counter to 10 (10 x 1 msec.) */

       Seconds = 0;  /* initialize second counter */
       SecHalf = 0;
       SecondTimer = ST_ON;
       led_dig_bcd(Seconds);      /* display second count */
       last_sec = 0;

       WT_Start();         /* start watch timer */

       while(1){
              /* new second? */
              if (Seconds != last_sec) {
                     led_dig_bcd(Seconds);      /* display new second */
                     last_sec = Seconds;
              }

              /* check switches */
              sw_val = sw_get();  /* check for switch pressed */
              switch (sw_val) {
              case SW_LD_RU:
                     Seconds = 0;  /* clear seconds */
                     SecHalf = 0;
                     led_dig_bcd(Seconds);      /* display it */
                     last_sec = 0;
                     while ( (sw_val = sw_get()) != SW_LU_RU)
                            ;      /* wait for switches up again */
                     break;
              case SW_LU_RD:
                     /* SW3 pressed, stop or start second timer */
                     if (SecondTimer == ST_OFF)
                            SecondTimer = ST_ON;
                     else
                            SecondTimer = ST_OFF;
                     while ( (sw_val = sw_get()) != SW_LU_RU)
                            ;      /* wait for switches up again */
                     break;
              default:
                     /* both up or both down - ignore */
                     break;
              }
       }
}
```

11.8.2 Systeminit.c

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

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

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

11.8.3 Watchtimer.h

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

```
/* added global variables for second counting */
extern UCHAR SecHalf;            /* counter of half seconds */
extern UCHAR Seconds;            /* count of seconds */
extern UCHAR SecondTimer;  /* control for timer on/off */
/* added definitions for SecondTimer */
#define ST_ON 1
#define ST_OFF      0

#endif
```

11.8.4 Watchtimer.c

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

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

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     This function initializes the watch timer module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
```

```
**
**-------------------------------------------------------------------------
*/
void WT_Init( void )
{
      WTM = 0;
      WTPR = 1;           /* low priority level */
      WTMK = 0;
      WTIPR = 1;          /* low priority level */
      WTIMK = 0;
      WTM.7 = 1;          /* watch timer clock: fw = fsub */
      ClrIORBit(WTM, 0x0c);      /* watch time: 2^14/fw (0.5s at 32.768Kz) */
      ClrIORBit(WTM, 0x70);      /* interval time: 2^5/fw */
      SetIORBit(WTM, 0x10);
      WT_User_Init( );
}

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

      return MD_OK;
}

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

```
        /* stop watch timer operation */
        WTM0 = 0;

        return MD_OK;
}
```

11.8.5 Watchtimer_user.c

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

#pragma interrupt INTWT MD_INTWT
#pragma interrupt INTWTI MD_INTWTI
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "watchtimer.h"

/* added include to find definition for sw_isr() and LED_PAT_DP */
#include "sw_0537.h"
#include "led_0537.h"

/*
************************************************************************
**  MacroDefine
************************************************************************
*/
/* added global variables */
UCHAR SecHalf;             /* counter of half seconds */
UCHAR Seconds;             /* count of seconds */
UCHAR SecondTimer;  /* control for timer on/off */

/*
**-----------------------------------------------------------------------
**
**  Abstract:
```

```
**    This function is an empty function for user code when initializing.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
void WT_User_Init( void )
{
      /* TODO. Add user code here */
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    INTWT interrupt service routine.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTWT( void )
{
      /* TODO. Add user defined interrupt service routine */
      /* this interrupt occurs once every 0.5 seconds */
      if (SecondTimer == ST_OFF)
            return;      /* timer is stopped */

      SecHalf++;                /* increment half-second counter */
      if (SecHalf == 1) {
            P7 &= LED_PAT_DP;   /* turn on decimal point to show half second */
      }

      if (SecHalf > 1) {
            /* two half seconds occured */
            SecHalf = 0;        /* reset half counter */
            Seconds++;                /* increment seconds */
            if (Seconds == 60) {
                  Seconds = 0; /* wrap around at 60 secconds */
            }
      }
}

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

```
**-------------------------------------------------------------------------
*/
__interrupt void MD_INTWTI( )
{
       /* TODO. Add user defined interrupt service routine */
       /* this interrupt occurs about once every millisecond */
       sw_isr();     /* handle switch debouncing */
}
```

11.9 Common Listings for Files in All Demonstration Programs

The listings in this section are for files which are identical for all of the demonstration programs where they are used. Some demonstration programs may use only some of the files in this section.

11.9.1 Macrodriver.h

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

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

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

#define      ON     1
#define      OFF    0

#define      TRUE   1
#define      FALSE  0

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

```
#define WRITE 2                  /* write mode */

#define SET   1
#define CLEAR 0

#define MD_STATUS          unsigned short
#define MD_STATUSBASE             0x0

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

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

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

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

enum INTLevel { Highest, Lowest };


#define       SYSTEMCLOCK   20000000
#define       SUBCLOCK      32768
#define       MAINCLOCK     20000000
#define       FRCLOCK       8000000
#define       FRCLOCKLOW    240000


#endif
```

11.9.2 System.h

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

```
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      system.h
**  Abstract :      This file implements device driver for SYSTEM module.
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :  uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************
*/
#ifndef      _MDSYSTEM_
#define      _MDSYSTEM_
/*
************************************************************************
**  MacroDefine
************************************************************************
*/
#define      CG_X1STAB_SEL 0x5
#define      CG_X1STAB_STA 0x1f
#define      CG_CPU_CLOCKSEL     0x0
#define      CG_Mainosc    0x14

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

void  Clock_Init( void );

#endif
```

11.9.3 System.c

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

```
**  Device :  uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************
*/
/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
/*
************************************************************************
**  MacroDefine
************************************************************************
*/

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     Init the Clock Generator and Oscillation stabilization time.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void Clock_Init( void )
{
       USHORT i;
       UCHAR temp_stabset, temp_stabwait;
       SetIORBit(PM12, 0x06);                  /* P121/122 input mode */
       ClrIORBit(OSCCTL, 0x80);         /* X1/X2 input mode */
       SetIORBit(OSCCTL, 0x40);
       ClrIORBit(MOC, 0x80);
       /* OSC stabilization time */
       temp_stabset = CG_X1STAB_STA;
       do{
              temp_stabwait = OSTC;
              temp_stabwait &= temp_stabset;
       }while(temp_stabwait != temp_stabset);
       OSTS = CG_X1STAB_SEL;
       for(i = 0; i <= 20; i++){        /* wait 5us */
              NOP();
       }
       SetIORBit(OSCCTL, 0x01);         /* 10MHz<fx<=20MHz */
       SetIORBit(MCM, 0x05);                   /* X1 operate for CPU */
       SetIORBit(PM12, 0x18);                  /* P123/124 input mode */
       ClrIORBit(OSCCTL, 0x20);         /* XT1 input mode */
       SetIORBit(OSCCTL, 0x10);
       ClrIORBit(RCM, 0x01);
       PCC = CG_CPU_CLOCKSEL;
}
```

11.9.4 Port.h

```
/*
************************************************************************
**
**  This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
**  78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  All rights reserved by NEC Electronics Corporation.
**
**  This program should be used on your own responsibility.
**  NEC Electronics Corporation assumes no responsibility for any losses
**  incurred by customers or third parties arising from the use of this file.
**
**  Filename :      port.h
**  Abstract :      This file implements device driver for PORT module.
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  Device :  uPD78F0537
**
**  Compiler :      NEC/CC78K0
**
************************************************************************
*/
#ifndef      _MDPORT_
#define      _MDPORT_
/*
************************************************************************
**  MacroDefine
************************************************************************
*/
#define      PORT_PM0     0xff
#define      PORT_PU0     0x0
#define      PORT_P0      0x0
#define      PORT_PM1     0xff
#define      PORT_PU1     0x0
#define      PORT_P1      0x0
#define      PORT_PM2     0xff
#define      PORT_P2      0x0
#define      PORT_PM3     0xff
#define      PORT_PU3     0x6
#define      PORT_P3      0x0
#define      PORT_PM4     0xf0
#define      PORT_PU4     0x0
#define      PORT_P4      0x0
#define      PORT_PM5     0xf0
#define      PORT_PU5     0x0
#define      PORT_P5      0x0
#define      PORT_PM6     0xff
#define      PORT_P6      0x0
#define      PORT_PM7     0x0
#define      PORT_PU7     0x0
#define      PORT_P7      0x0
#define      PORT_PM12    0xff
#define      PORT_PU12    0x0
#define      PORT_P12     0x0
#define      PORT_P13     0x0
#define      PORT_PM14    0xff
#define      PORT_PU14    0x0
#define      PORT_P14     0x0
#define      PORT_ADPC    0x0
```

```
void PORT_Init( void );

#endif
```

11.9.5 Port.c

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

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**     This function initializes the I/O module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------
*/
void PORT_Init( void )
{
      /* initialize the port registers */
```

```
        P0 = PORT_P0;
        P1 = PORT_P1;
        P2 = PORT_P2;
        P3 = PORT_P3;
        P4 = PORT_P4;
        P5 = PORT_P5;
        P6 = PORT_P6;
        P7 = PORT_P7;
        P12 = PORT_P12;
        P13 = PORT_P13;
        P14 = PORT_P14;

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

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

11.9.6 Ad.h

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

```
*/
#ifndef      _MDAD_
#define      _MDAD_
/*
************************************************************************
**  MacroDefine
************************************************************************
*/
#define      AD_ADPC      0x0
#define      AD_ADS 0x0
#define      AD_PM2 0xff
#define      AD_ADM 0x38

enum INTMode { INTMode0, INTMode1, INTMode2 };

void AD_Init( void );
MD_STATUS AD_Start( void );
MD_STATUS AD_Read( USHORT* buffer );
MD_STATUS AD_Stop( void );
MD_STATUS AD_Stop( void );

#endif
```

11.9.7 Ad.c

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

/*
**------------------------------------------------------------------------
**
**  Abstract:
```

```
**     This function initializes A/D module.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------
*/
void AD_Init( void )
{
       ADPC = AD_ADPC;
       PM2 = PM2 | AD_PM2;
       ADS = AD_ADS;
       ClrIORBit(ADM, 0x02);                   /* 2.7V<=AVref<=5.5V */
       ADM = ADM & AD_ADM;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function starts the A/D converter.
**
**  Parameters:
**     None
**
**  Returns:
**     MD_OK
**
**-------------------------------------------------------------------------
*/
MD_STATUS AD_Start( void )
{
       int delay = 200;
       SetIORBit(ADM, 0x01);                   /* comparator operation */
       while(delay--);

       SetIORBit(ADM, 0x80);
       return MD_OK;
}

/*
**-------------------------------------------------------------------------
**
**  Abstract:
**     This function can be called after an A/D conversion is completed,
**     and returns the conversion result(s) in the buffer.
**
**  Parameters:
**     buffer The address where to write the conversion result.
**
**  Returns:
**     MD_OK
**
**-------------------------------------------------------------------------
*/
MD_STATUS AD_Read( USHORT* buffer )
{
       *buffer = (USHORT)( ADCR >> 6 );
       return MD_OK;
}
```

```
/*
**-------------------------------------------------------------------------
**
**  Abstract:
**    This function stops the A/D converter.
**
**  Parameters:
**    None
**
**  Returns:
**    MD_OK
**
**-------------------------------------------------------------------------
*/
MD_STATUS AD_Stop( void )
{

      ClrIORBit(ADM, 0x81);
      return MD_OK;
}
```

11.9.8 Ad_user.c

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

/*
************************************************************************
**  Include files
************************************************************************
*/
#include "macrodriver.h"
#include "ad.h"
```

11.9.9 Led_0537.h

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

#ifndef _LED_0537_H
#define _LED_0537_H

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

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

#define LED_PAT_0   0xC0
#define LED_PAT_1   0xF9
#define LED_PAT_2   0xA4
#define LED_PAT_3   0xB0
#define LED_PAT_4   0x99
#define LED_PAT_5   0x92
#define LED_PAT_6   0x82
#define LED_PAT_7   0xF8
#define LED_PAT_8   0x80
#define LED_PAT_9   0x98
#define LED_PAT_A   0x88
#define LED_PAT_B   0x83
#define LED_PAT_C   0xC6
#define LED_PAT_D   0xA1
#define LED_PAT_E   0x86
#define LED_PAT_F   0x8E
#define LED_PAT_BLANK      0xFF
#define LED_PAT_DP         0x7F
#define LED_PAT_DASH       0xBF
#define LED_PAT_ULINE      0xF7
#define LED_PAT_OLINE      0xFE
#define LED_PAT_EQUAL      0xB7

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

#endif /* _LED_0537_H */
```

11.9.10 Led_0537.c

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

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

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

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

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

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

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

#include "led_0537.h"

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

```
   LED_PAT_F  /* F */
};

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

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

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

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

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

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

/* void led_dig_bcd(unsigned char bcdnum) */
```

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

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

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

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

11.9.11 Sw_0537.h

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

#ifndef _SW_0537_H
#define _SW_0537_H

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

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

#define      SW_DEF_DEB_COUNT    8     /* default debounce counter      */

/*****************************************************************/
/* Export functions                                              */
```

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

#endif /* _SW_0537_H */
```

11.9.12 Sw_0537.c

```
/*     sw_0537.c - routines for switch input                      */
/*           for M-78F0537 CPU board on M-Station base board      */
/*  Version: 1.1    01-13-2006                                    */

/*           P31 = input for left switch  (SW2)                        */
/*           P32 = input for right switch (SW3)                        */

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

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

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

#include "sw_0537.h"

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

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

/* unsigned char sw_chk(void) */
/*     return input from switches, undebounced */
```

```
unsigned char sw_chk(void)
{
       return P3 & 0x06;
}

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

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

unsigned char sw_get(void)
{
       return sw_last;
}


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

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

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

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

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

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

11.9.13 Option.inc

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

;
;*******************************************************************************
;**  MacroDefine
;*******************************************************************************
;
OPTION_BYTE   EQU    00H
POC81  EQU    00H
POC82  EQU    00H
POC83  EQU    00H
CG_ONCHIP     EQU    02H
CG_SECURITY0  EQU    0ffH
CG_SECURITY1  EQU    0ffH
CG_SECURITY2  EQU    0ffH
CG_SECURITY3  EQU    0ffH
CG_SECURITY4  EQU    0ffH
CG_SECURITY5  EQU    0ffH
CG_SECURITY6  EQU    0ffH
CG_SECURITY7  EQU    0ffH
CG_SECURITY8  EQU    0ffH
CG_SECURITY9  EQU    0ffH
```

11.9.14 Option.asm

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

```
;**
;**  Filename : option.asm
;**  Abstract : This file implements OPTION-BYTES/SECURITY-ID setting.
;**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;**  Device : uPD78F0537
;**
;**  Compiler :     NEC/CC78K0
;**
;*****************************************************************************

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

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