Application Note

Key Return Input and Sound Output

For NEC Electronics Microcontrollers

Document no. U18272EU1V0AN00

Revision History

Date	Revision	Section	Description
July 2006	—	—	First release

Contents

1. Introduction

This document provides simple examples that illustrate the use of the peripherals included in NEC Electronics’ microcontrollers. The document includes:

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

The Applilet is a software tool that easily generates driver code for on-chip peripherals.

Reference the target microcontroller’s user manual and other related documents for further details.

1.1 Overview of Key Return Functions and Sound Output

This application note describes key-input functions and two different ways to generate sound from an NEC Electronics’ 78K0 microcontroller.

The key-return features provided by NEC Electronics’ microcontrollers recognize key inputs by scanning key-switch matrices and processor inputs. If you connect the key-return I/O port pins to individual switches, then pressing any switch causes a key-return interrupt. You can also scan keys with a general-purpose I/O port, and you can configure a key-switch matrix between the key-return and general-purpose ports.

Pressing switches generally produces noise (sometimes referred to as chattering), so you must debounce them by sampling the key inputs over a preset interval and recognizing the key input only when there is no state change over that time. For example, if you sample the key input every millisecond and require that the data be stable for five sampling periods, you debounce the switch in five milliseconds.

For sound generation, most NEC Electronics’ microcontrollers provide a buzzer (BUZ) output capable of directly driving an external speaker or buzzer with a square wave. If, however, the microcontroller you chose does not have a dedicated buzzer output, you can use any of several timer/counters that feature square-wave, programmable pulse generator (PPG) or PWM features to drive a speaker or buzzer.

2. Key Return Input and Sound Output

2.1 Key Return Input and Sound Output Features

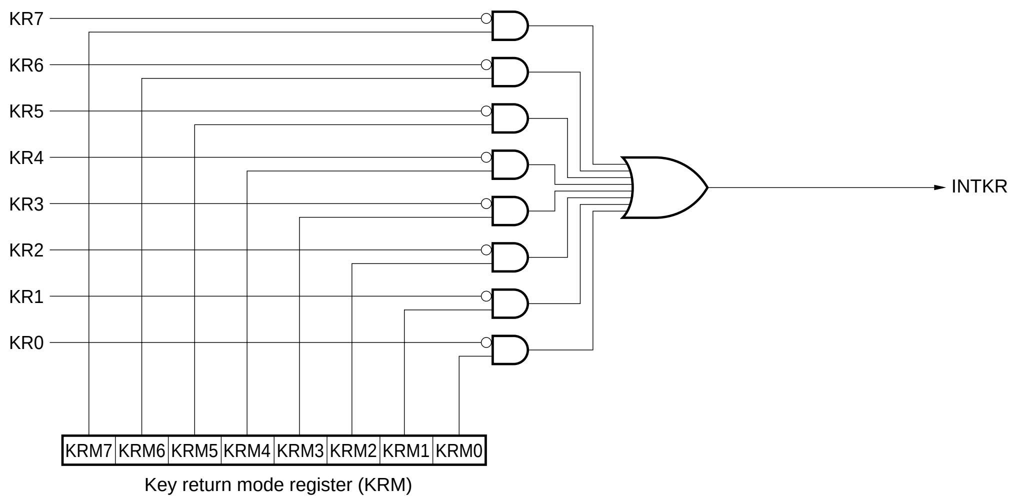
Below is a brief description of the key-return and sound-output features of NEC Electronics' microcontrollers. Most NEC Electronics microcontrollers use similar physical circuits for these functions.

2.1.1 Key Return (KR) Input Features

- ♦ Interrupt triggered by a negative edge on any enabled input
- ♦ Individual enables for each KR input

Any KR port pins not used for key return can function as independent ports.

Figure 1. Configuration of Key-Return Interrupt



After you set the Key-Return Mode Register (KRM0 to KRM7), the port pin becomes an alternate function for key-interrupt input. A negative edge on any of the selected inputs (KR0 to KR7) triggers an interrupt. Then you read the inputs as an I/O port to determine which inputs are low.

The microcontroller's internal Pull-up Resistor Control Register (PU7) sets internal pull-up resistors for KR0 to KR7, respectively.

To configure the key-return function:

- ♦ Set the KRMK mask bit to disable the key-return interrupt.
- ♦ Use the appropriate pull-up (PU) register bits to enable pull-up resistors on the port pins.
- ♦ Set the port pins as inputs with the appropriate port mode (PM) register bits.
- ♦ Set the KRM register to enable the desired KR inputs.
- ♦ Set the interrupt priority, and enable the interrupt by clearing the KRMK bit.

2.1.2 Features of Timer Output for Sound Generation

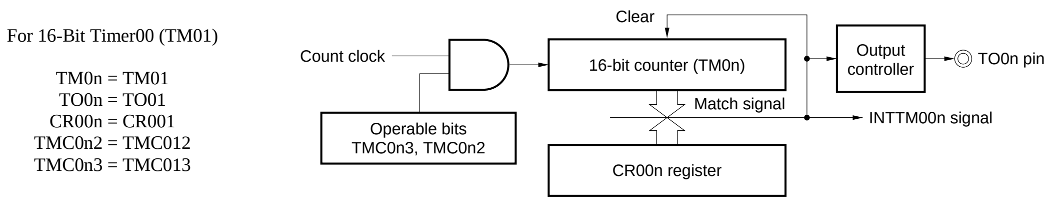
Some NEC Electronics microcontrollers have a buzzer output. For those that do not, you can use an internal timer to generate a square wave to drive a speaker or buzzer. For example, most NEC Electronics microcontrollers have 16-bit timers, such as Timer 01 (TM01)

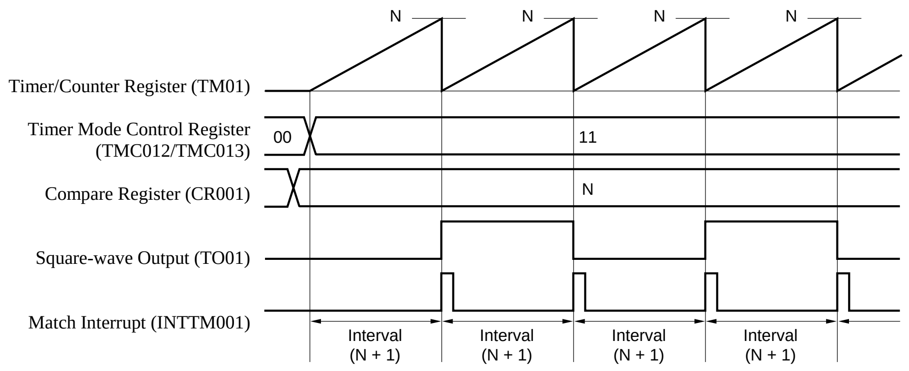
In square-wave generation mode, Timer 01 offers the following features:

- ♦ Variable time base based on divisions of the peripheral clock, using the PRM01 register
- ♦ Variable square-wave half-cycle time of 1 to 65536 clocks, using the CR001 register
- ♦ A precise 50% duty cycle; CR001 counts both high and low periods
- ♦ Optional interrupts on the end of each half cycle

When you set the Timer-Mode Control Register to its clear-and-start mode, the timer outputs a square wave from output pin TO01, with a pulse width set by the Compare Register (CR001).

Figure 2. Timer 01 Generates Square Wave for Buzzer





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

To configure the timer for square-wave output:

- Set the time base using PRM01.
- Set CR001 as a compare register using CRC01.
- Set the half-cycle time in CR001.
- 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 TOC01 register for output enable, and invert on CR001 match.
- Enable the timer by setting the Timer-Mode Register TMC01 to go to its clear-and-start mode on compare of TM01 and CR001.

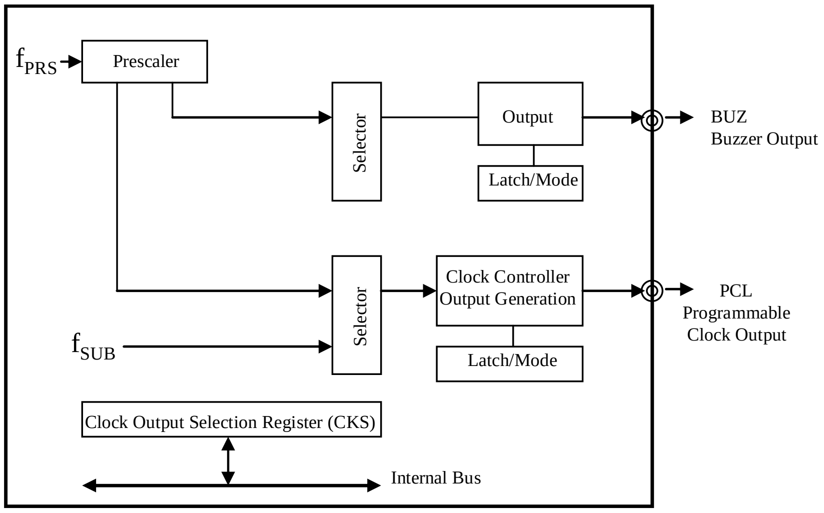
2.1.3 Features of Buzzer Output for Sound Generation

The buzzer output (BUZ) provided by NEC Electronics' microcontrollers offers:

- ♦ Multiple output frequencies derived from the peripheral clock
- ♦ Simple enable/disable bit for control

The buzzer output is a square-wave at the frequency selected by clock-output selection register CKS. The CKS, port-mode and port registers control buzzer operation.

Figure 3. Controlling the Buzzer Frequency



fPRS
Prescaler
Selector
Output
Latch/Mode
BUZ
Buzzer Output
Selector
Clock Controller Output Generation
Latch/Mode
PCL
Programmable Clock Output
fSUB
Clock Output Selection Register (CKS)
Internal Bus

To configure the buzzer:

- ♦ Use the appropriate PM register to set the port pin used for the BUZ output.
- ♦ Select the buzzer frequency with the CKS register.
- ♦ Use the BZOE bit in the CKS register to enable the buzzer output.

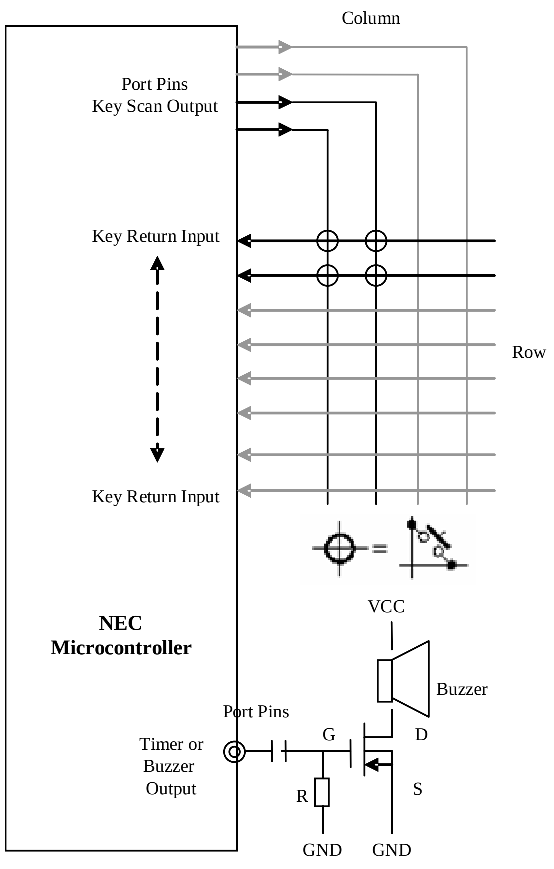
2.2 Program Description and Specification

The demonstration program uses a 2x2-switch matrix and an NEC Electronics microcontroller to illustrate the key-return function and key-switch matrix scanning. The microcontroller generates one of four different tones, depending on the switch pressed. You can use more key-scan or key-return inputs to accommodate a larger switch matrix.

The key-return port checks key status, while a general-purpose port (column lines) scans the keys. Internal pull-up registers pull the key-return port inputs (row lines) high when no key is pressed.

The key-scan output port forces all column lines low. Pressing a switch pulls one of the key-interrupt port input pins low, triggering a key-return interrupt. The interrupt-service sets the key-scan output ports to drive only one column line low at a time and reads the state of the key return port to determine which key (or keys) you have pressed.

Figure 4. Configuring Microcontroller for Key Scanning



Specifications:

- ♦ Switch data must be consistent over a 5-milliseconds interval.
- ♦ Pressing each switch selects a 500-millisecond tone.
- ♦ Timer 01 generates the square-wave output.
  - Switch 1 selects 3520 Hz.
  - Switch 2 selects 1760 Hz.
  - Switch 3 selects 880 Hz.
  - Switch 4 selects 440 Hz.
- ♦ An alternative use of the buzzer output (described but not used) generates different frequencies.
  - Switch 1 selects 7812 Hz.
  - Switch 2 selects 3906 Hz.
  - Switch 3 selects 1953 Hz.
  - Switch 4 selects 976 Hz.

2.3 Software Flow Charts

The demonstration program consists of:

- ♦ Program-initialization code, called before the main() program starts (includes key-return and timer or buzzer initialization)
- ♦ The main program loop, which reads the switches and starts tone generation
- ♦ Subroutines (generated by the Applilet) to handle timer starting, stopping, and interval setting
- ♦ Alternatively, subroutines (generated by the Applilet) to start and stop the buzzer
- ♦ Subroutines with user code for handling key-return and timer interrupts (Applilet-generated stub interrupt service routines, with user code added)

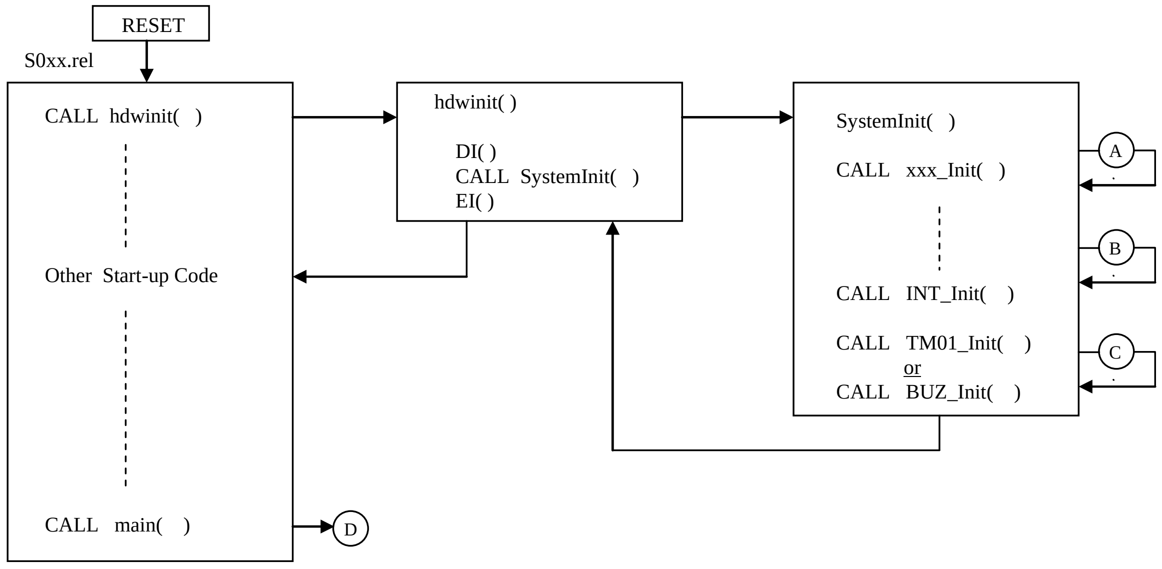
The flowcharts describe this initialization, the main program, key-return interrupt and key scanning, timer routines for square-wave generation, buzzer routines, and user interrupt service.

2.3.1 Program Startup and Initialization

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

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

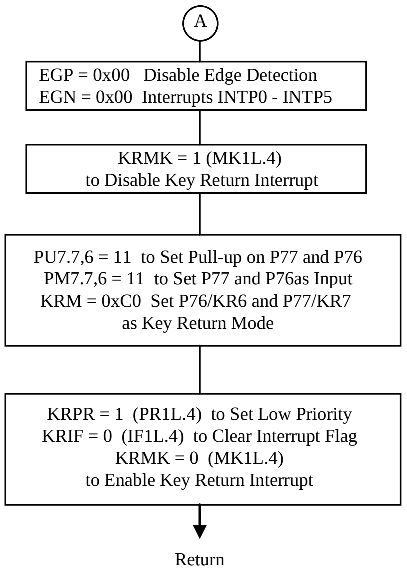
Figure 5. Flowchart for System Initialization



When the hdwinit() function finishes, the startup code calls the user program's main() function. Thus, when the main() function starts, peripheral initialization is complete for the ports, key-return interrupt and the timers.

2.3.2 INT_Init( ) – Key-Return Interrupt Initialization

Figure 6. Flowchart for Key-Return Interrupt Initialization


A
EGP = 0x00 Disable Edge Detection
EGN = 0x00 Interrupts INTP0 - INTP5
KRMK = 1 (MK1L.4)
to Disable Key Return Interrupt
PU7.7,6 = 11 to Set Pull-up on P77 and P76
PM7.7,6 = 11 to Set P77 and P76as Input
KRM = 0xC0 Set P76/KR6 and P77/KR7
as Key Return Mode
KRPR = 1 (PR1L.4) to Set Low Priority
KRIF = 0 (IF1L.4) to Clear Interrupt Flag
KRMK = 0 (MK1L.4)
to Enable Key Return Interrupt
Return

SystemInit(), calls the INT_Init() routine to set up the key-return function. This demonstration uses pins P76/KR6 and P77/KR7 as key-return inputs from the key-switch matrix. The initialization routine sets the EGP and EGN registers to disable other external interrupts.

The routine sets the key-return interrupt mask bit KRMK to 1 to mask the interrupt while modifying other registers.

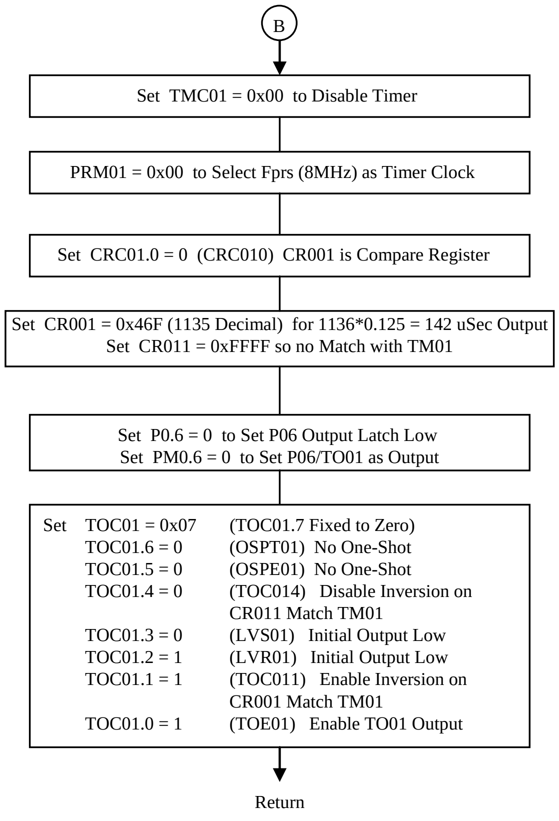
The initialization routine writes 0xC0 to the PU7 register to provide pull-up resistors on the P77-P76 port pins, and writes the same value to the port-mode register PM7 to configure the pins as inputs. INT_Init() also writes 0xC0 to the key-return mode register, KRM, to specify that the key-return interrupt sources KR6 and KR7 trigger a key-return interrupt. A negative edge on either pin P76/KR6 or P77/KR7 triggers the interrupt.

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

See the section on the MD_INTKR() key-return interrupt-service routine for a description of how the program handles key-return interrupts.

2.3.3 TM01_Init( ) – Timer 01 Initialization for Square-Wave Generation

Figure 7. Flowchart for Initializing Timer


B
Set TMC01 = 0x00 to Disable Timer
PRM01 = 0x00 to Select Fprs (8MHz) as Timer Clock
Set CRC01.0 = 0 (CRC010) CR001 is Compare Register
Set CR001 = 0x46F (1135 Decimal) for 1136*0.125 = 142 uSec Output
Set CR011 = 0xFFFF so no Match with TM01
Set P0.6 = 0 to Set P06 Output Latch Low
Set PM0.6 = 0 to Set P06/TO01 as Output
Set TOC01 = 0x07 (TOC01.7 Fixed to Zero)
TOC01.6 = 0 (OSPT01) No One-Shot
TOC01.5 = 0 (OSPE01) No One-Shot
TOC01.4 = 0 (TOC014) Disable Inversion on CR011 Match TM01
TOC01.3 = 0 (LVS01) Initial Output Low
TOC01.2 = 1 (LVR01) Initial Output Low
TOC01.1 = 1 (TOC011) Enable Inversion on CR001 Match TM01
TOC01.0 = 1 (TOE01) Enable TO01 Output
Return

SystemInit() calls the TM01_Init() routine to initialize Timer 01 for square-wave output. The routine disables the timer before writing to the control registers.

TM01_Init() sets the prescaler-mode register PRM01 to 0x00 to select f_{PRS} , the main 8-MHz clock, as the count clock. Thus, each TM01 count takes 0.125 microseconds.

TM01_Init() sets CRC01 to use CR001 as a compare register and sets CR001 to 0x046F (1135) for a square-wave width of 1136 * 0.125 = 142 microseconds. This setting results in a square wave with a high time of 142 microseconds, and low time of 142 microseconds for a total frequency of 1/(142 + 142) = about 3520 Hz.

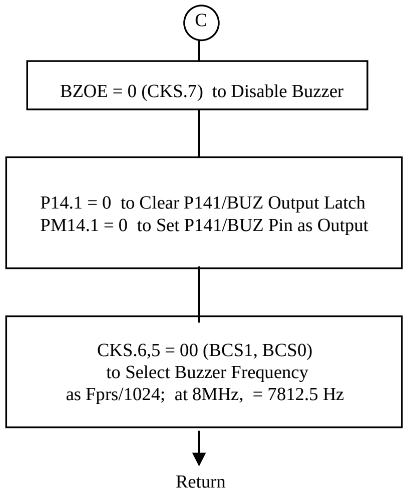
TM01_Init() sets the P0.6 output latch and the port-mode register PM0 bit 6 to zero, which directs the timer's output to pin P06/TO01.

The initialization routine sets the TOC01 control register for initially low output to invert the timer output when TM01 and CR001 match and to enable the TO01 output.

At this point, the program has set the Timer 01 registers, but the timer is not running. Setting the timer to clear-and-start mode begins square-wave output.

2.3.4 BUZ_Init( ) – Alternative Buzzer Initialization for Square-Wave Buzzer Output

Figure 8. Flowchart for Alternative Buzzer Initialization


C
BZOE = 0 (CKS.7) to Disable Buzzer
P14.1 = 0 to Clear P141/BUZ Output Latch
PM14.1 = 0 to Set P141/BUZ Pin as Output
CKS.6,5 = 00 (BCS1, BCS0)
to Select Buzzer Frequency
as Fprs/1024; at 8MHz, = 7812.5 Hz
Return

Note that the demonstration program uses the uPD78F0397 (78K0/LG2) device, which does not support buzzer output. The uPD78F0537 (78K0/KE2) does support buzzer output, and the Appendix of this application note includes the code to initialize the on-chip buzzer (files buzzer.h and buzzer.c). The demonstration program does not use these files, however.

In the alternative initialization, SystemInit() calls the BUZ_Init() routine. The CKS register, which controls the clock output function, also controls the buzzer function. The routine sets the BZOE enable bit (bit 7 in the CKS register) to zero to disable the buzzer.

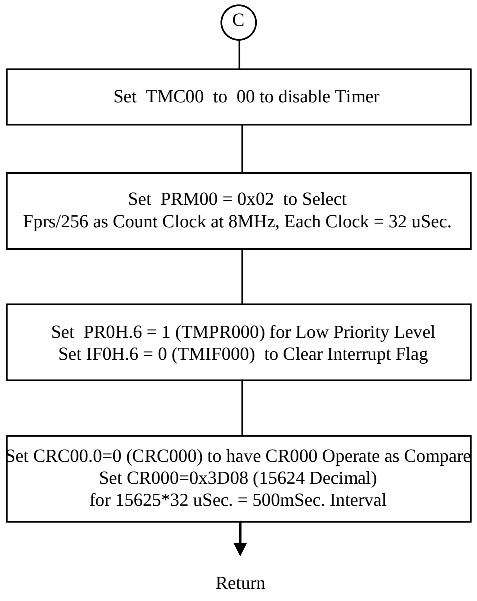
The NEC Electronics microcontroller that supports buzzer output provides the buzzer on pin P141/BUZ. To make this pin an output, the initialization routine sets both the P14.1 output-port latch and the port-mode register PM14 bit 1 to zero.

The BCS1 and BCS0 bits in the CKS register (CKS.6 and CKS.5) control the buzzer frequency by selecting one of four possible divisions of the main peripheral clock. The initialization routine sets the initial value for BCS1,0 = 00, which selects f_{PRS}/1024. The main peripheral clock runs at 8 MHz, resulting in a 7812.5-Hz buzzer frequency.

These settings initialize, but do not start, the buzzer. Setting the BZOE bit to 1 starts buzzer output.

2.3.5 TM00_Init( ) – Timer 00 Initialization for 500-Millisecond Interval

Figure 9. Initializing 500-millisecond Interval


C
Set TMC00 to 00 to disable Timer
Set PRM00 = 0x02 to Select
Fprs/256 as Count Clock at 8MHz, Each Clock = 32 uSec.
Set PR0H.6 = 1 (TMPR000) for Low Priority Level
Set IF0H.6 = 0 (TMIF000) to Clear Interrupt Flag
Set CRC00.0=0 (CRC000) to have CR000 Operate as Compare
Set CR000=0x3D08 (15624 Decimal)
for 15625*32 uSec. = 500mSec. Interval
Return

SystemInit() calls the TM00_Init() routine, which sets the Timer 00 registers to prepare for interval-timer operation. First the routine disables the timer while changing register settings.

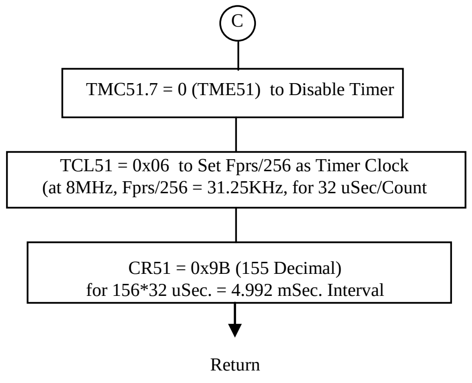
The initialization routine sets the prescaler-mode register, PRM00, which controls the timer clock, to 0x02. This setting causes Timer 00 to use f_{PRS}/256 as the count clock. With f_{PRS} (the main peripheral clock) at 8 MHz, the Timer 00 count clock will be 8,000,000/256, or 31.250 kHz. The count register TM00 counts once every 1/31250 seconds, or once every 32 microseconds.

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

TM00_Init() sets the CRC00 register for CR000 to act as a compare register and sets the compare value to 0x3D08 (15624). This value causes CR000 to match TM00 every (15624 + 1), or 15625 counts. This match occurs once every 15625 * 32 microseconds = 500 milliseconds. The Applilet calculates the 0x3D08 value to provide the 500-millisecond interval.

2.3.6 TM51_Init( ) – Timer 51 Initialization for 5 Millisecond Interval

Figure 10. Flowchart for Initializing 5-millisecond Timer


C
TMC51.7 = 0 (TME51) to Disable Timer
TCL51 = 0x06 to Set Fprs/256 as Timer Clock
(at 8MHz, Fprs/256 = 31.25KHz, for 32 uSec/Count
CR51 = 0x9B (155 Decimal)
for 156*32 uSec. = 4.992 mSec. Interval
Return

SystemInit() calls the TM51_Init() routine to initialize Timer51 as an interval timer with a 5-millisecond duration. TM51_Init() first sets the Timer51 enable bit TME51 to zero to disable the timer while changing control registers.

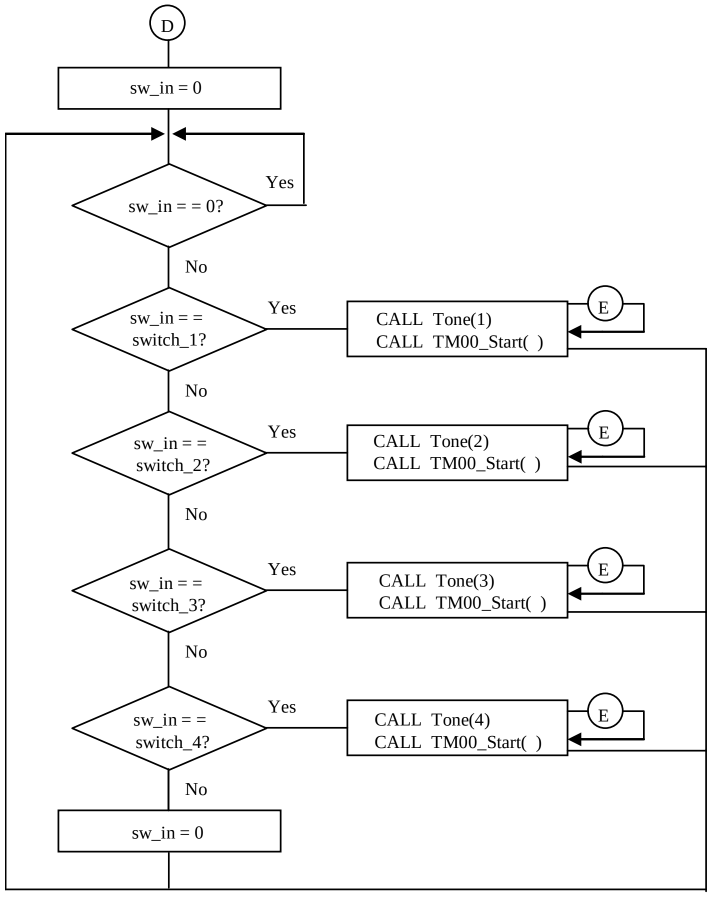
TM51_Init() sets the TCL51 register to 0x06 to set the clock to Timer51 as f_{PRS}/256. This setting causes the timer to divide the 8-MHz main peripheral clock by 256 for a 31.250-kHz clock. Each clock count lasts 32 microseconds.

The initialization then sets the CR51 compare register to 0x9B (155), causing the timer to compare with CR51 every 156 counts. This comparison rate results in an interval of 156 * 32 microseconds = 4.992 milliseconds — close enough to the desired 5-millisecond interval.

The key-return interrupt-service routine uses TM51 to check switch data 5 milliseconds after the initial interrupt.

2.3.7 Main( ) –Main Program – Key Return and Buzzer Output

Figure 11. Flowchart for Key Return and Buzzer Output



Main() sets the initial value of the sw_in variable to zero, indicating that no switch is down. After this initialization, the program loops, checking to see if the value changes. When a switch is detected and debounced, the key-return interrupt-service routine sets its value in sw_in.

If you do not press a switch, the program does nothing but loop.

When the main program detects a new (non-zero) variable in sw_in, the routine verifies that this value matches a pattern indicating that you pressed only one switch. The program ignores any patterns indicating that you pressed more than one switch.

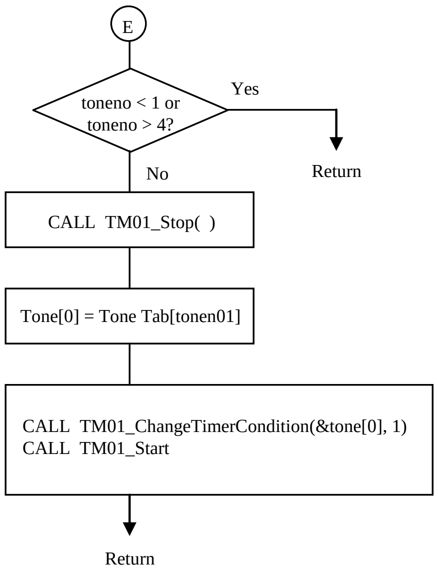
If the pattern in sw_in indicates that you pressed one of the four switches, the main program calls the Tone(n) routine to start the corresponding tone.

The main program calls TM00_Start() to start the interval timer. After the 500-millisecond interval, the timer triggers the INTTM000 interrupt, invoking the MD_INTTM000() interrupt-service routine, which stops the tone.

The main program then clears the sw_in variable to zero again.

2.3.8 Tone(UCHAR toneno) – Tone Generation Using Timer 00 Square-Wave Generation

Figure 12. Generating Tone with Timer 00


E
toneno < 1 or toneno > 4?
Yes
Return
No
CALL TM01_Stop( )
Tone[0] = Tone Tab[tonen01]
CALL TM01_ChangeTimerCondition(&tone[0], 1)
CALL TM01_Start
Return

When you use Timer 01 for square-wave tone generation, you can select the output frequency by writing a value from 1 to 65535 to the CR001 register. The Tone() routine sets one of four frequencies.

The Tone(toneno) routine first checks to see if the toneno parameter is in the range of 1 to 4. If the value is outside that range, the routine returns without starting TM01.

If toneno is 1 through 4, the routine stops the Timer 01 output by calling TM01_Stop().

The toneno parameter indexes into the ToneTab[ ] array of USHORT (16-bit) values, selecting one of four possible values, which it places in the tone[0] variable.

Tone(toneno) calls the TM01_ChangeTimerCondition(&tone[0], 1) routine, passing the address of tone[0], and a second parameter of 1. This action writes the value contained in tone[0] to the CR001 compare register, changing the square-wave frequency.

Table 1 shows the values corresponding to the toneno parameter, the selected ToneTab location and value, the resulting square-wave width (half-cycle time) and square-wave frequency. The square-wave width time is (value + 1) * (clock period). The demonstration uses f_{PRS} (8 MHz) as the TM01 clock for a clock period of 0.125 microseconds. (The square-wave width is value * 0.125 usec.) The square-wave frequency is 1/(2 * square-wave width).

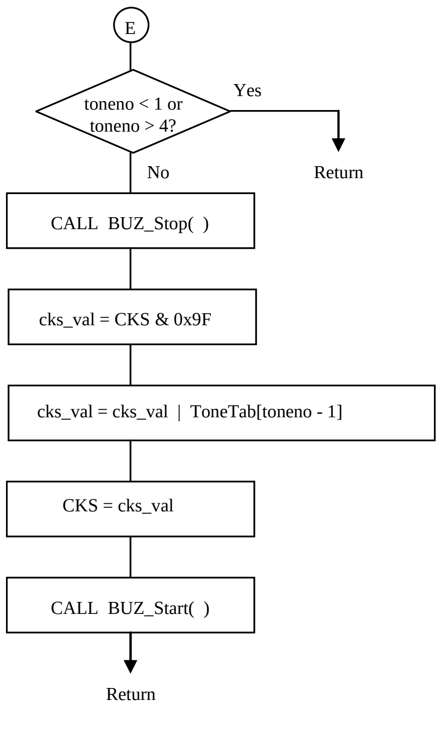
Table 1. Results for Toneno Parameters

Toneno	ToneTab[ ]	Value	Value + 1 (decimal)	Square Wave Width (μsec)	Square Wave Frequency
1	ToneTab[0]	0x046F	1136	142	3521.1 Hz
2	ToneTab[1]	0x08DF	2272	284	1760.6 Hz
3	ToneTab[2]	0x11BF	4544	568	880.3 Hz
4	ToneTab[3]	0x237F	9088	1136	440.1 Hz

Once Tone(toneno) writes a new value into CR001, the routine starts Timer 01 by calling TM01_Start(). Once started, the timer generates a square wave at the selected frequency until stopped.

2.3.9 Tone(UCHAR toneno) – Alternate Tone Generation Using Buzzer Output

Figure 13. Flowchart for Alternate Tone Generation



When you use the buzzer output, you can choose among four output frequencies, selecting the appropriate frequency by setting the BCS1 and BCS0 bits in the CKS register. BCS1 is CKS.6, and BCS0 is CKS.5.

The Tone(toneno) routine first checks to ensure the toneno parameter lies within the range of 1 to 4. If the parameter is outside of that range, the routine returns without starting the buzzer.

If toneno is 1 through 4, the routine calls BUZ_Stop() to stop buzzer output. The routine reads the current value in the CKS register and masks out bits 6 and 5 using a logical AND with 0x9F. The routine stores this masked value in variable cks_val.

Using the toneno parameter to index into the ToneTab[ ] array, the routine ORs in one of four possible bit patterns for BCS1 and BCS0. The routine combines the result in cks_val. Now cks_val holds the new value for the CKS register, with bits 6 and 5 selecting the buzzer output frequency. Table 2 shows the values corresponding to the toneno parameter, the corresponding bits for BCS1 and BCS0, and the resulting buzzer frequency.

Table 2. Toneno Values for Buzzer-Frequency Selection

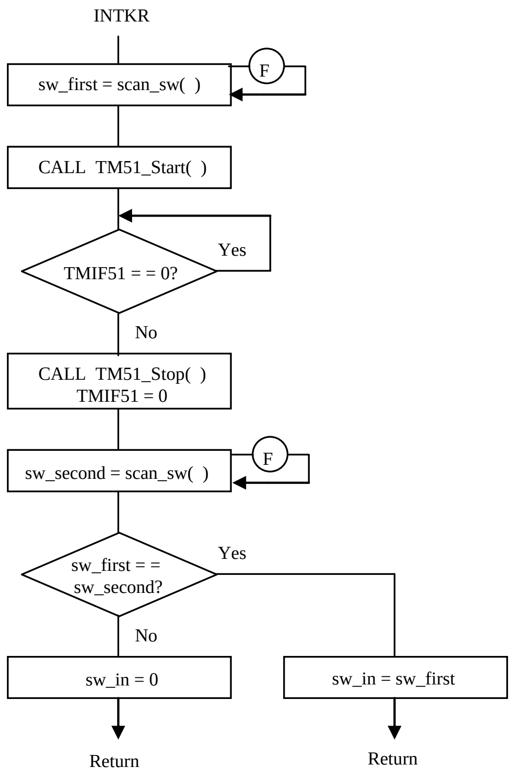
Toneno	ToneTab[ ]	Value	BCS1	BCS0	Clock division	Frequency at f_{PRS} = 8 MHz
1	ToneTab[0]	0x00	0	0	f_{PRS}/1024	7812.5 Hz
2	ToneTab[1]	0x20	0	1	f_{PRS}/2048	3906.2 Hz
3	ToneTab[2]	0x40	1	0	f_{PRS}/4096	1953.1 Hz
4	ToneTab[3]	0x60	1	1	f_{PRS}/8192	976.6 Hz

Tone(toneno) writes the new value in cks_val to the CKS register to select the desired frequency. The routine then calls the BUZ_Start() routine to start buzzer output by setting the BZOE bit (CKS.7) to 1.

The demonstration program does not use this version of the Tone() routine, but the routine is included with the source code listings in the Appendix.

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

Figure 14. Flowchart for Key-Return Interrupt-Service Routine



The MD_INTKR() routine handles the INTKR interrupt. Pressing any switch in the key-switch matrix connects the switch to one of the key-scan output lines, driving the key-return input line low. The resulting negative edge triggers the INTKR interrupt.

The MD_INTKR() routine calls the scan_sw() routine to read the state of all of the switches and stores the result in the sw_first variable.

The TM51_Start() routine starts Timer TM51, which counts out a 5-millisecond interval. After that interval, the routine sets its interrupt flag TMIF51 to true. TM51_Start() does not use an interrupt-service routine to indicate when the flag is set; MD_INTKR() simply tests the flag repeatedly. When the flag is set, the routine stops TM51 and clears the flag.

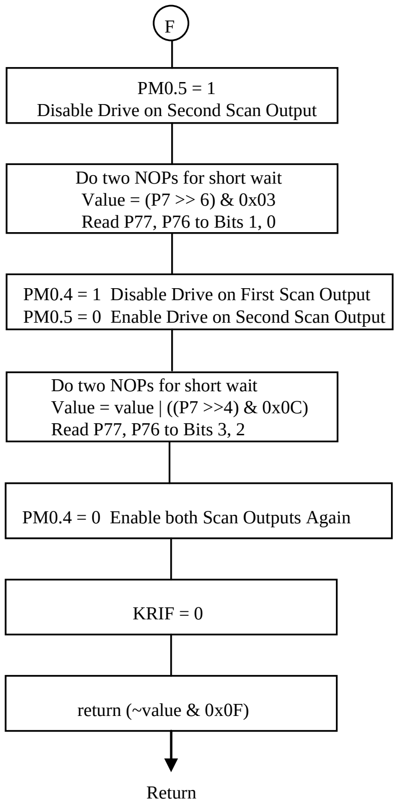
The interrupt handler then calls the scan_sw() routine to read the switch state again. If this reading matches the first value, the switches have been stable for more than 5 milliseconds, and the routine considers them debounced. MD_INTKR() sets the global variable sw_in to the debounced value and the routine returns.

If the first and second switch readings do not agree, MD_INTKR() sets sw_in to zero, indicating that the switch is not stable. The routine then returns.

If the switches continue to bounce, the next negative edge on the key-return inputs causes another INTKR interrupt, and the handler checks the state of the switches again.

2.3.11 Unsigned char scan_sw( ) – Scan Key-Switch Matrix

Figure 15. Flowchart for Scan Key-Switch Matrix


F
PM0.5 = 1
Disable Drive on Second Scan Output
Do two NOPs for short wait
Value = (P7 >> 6) & 0x03
Read P77, P76 to Bits 1, 0
PM0.4 = 1 Disable Drive on First Scan Output
PM0.5 = 0 Enable Drive on Second Scan Output
Do two NOPs for short wait
Value = value | ((P7 >>4) & 0x0C)
Read P77, P76 to Bits 3, 2
PM0.4 = 0 Enable both Scan Outputs Again
KRIF = 0
return (~value & 0x0F)
Return

MD_INTKR() calls the scan_sw() routine to scan the key-switch matrix and return a value indicating which switch or switches are down. At the start of this routine, output ports P04 and P05 drive a low logic value. Any depressed switch connects either P76 or P77 (or both) to P04 or P05 and is seen as low. To scan the matrix to see which switch or switches are actually down, the routine needs to turn off the drive on all but one column and check the inputs.

The scan_sw() routine first turns off the P05 key-scan output-port line by setting PM0.5 to one. This change sets P05 as an input. P04 is still an output and driving low, but P05 is no longer driving. P05 is pulled up by an internal pull-up resistor, because the PORT_Init() routine previously set the PU0.5 bit to one.

After a short wait to allow any input lines not being driven low to be pulled up, scan_sw() reads, shifts and masks the P7 port. These actions store the state of P77 in value variable bit 1 and the state of P76 in bit 0. Pressing switch 1 connects P04 and P76, causing P76 to go low and resulting in a zero in bit 0. Pressing switch 2 connects P04 and P77, causing P77 to go low and resulting in a zero in bit 1. Pressing either switch 3 or 4 connects P05 to either P77 or P76. Since P05 is not driving, however, connecting P05 has no effect, so pressing switches 3 and 4 does not affect the value read.

The scan_sw() routine next turns off the P04 key-scan output-port line. This action sets PM0.4 to one, which sets P04 as an input. The routine then sets PM0.5 to zero, which sets P05 as an output again. At this point, P05 is driving low, and P04 is not driving. Because the PORT_Init() routine set the PU0.4 bit to one, an internal pull-up resistor pulls up P04.

After a short wait to allow inputs to stabilize, scan_sw() reads, shifts, and masks the P7 port. This action passes the state of P77to variable bit 3 and the state of P76 to bit 2. Pressing switch 3 connects P05 and P76. P76 will be low, resulting in a zero in bit 2. Pressing switch 4 connects P05 and P77. P77 will be low, resulting in a zero in bit 3. Pressing switch 1 or 2 connects P04 to either P77 or P76, but since P04 is not driving, this connection has no effect.

The value variable now contains four bits (0 through 3) which reflect the state of switches 4 through 1, respectively. Pressing a switch sets its corresponding bit to zero; if the switch is up, its corresponding bit gets set to one.

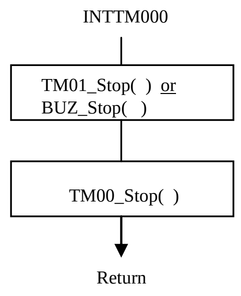
The scan_sw() routine then sets PM0.4 to zero again to enable drive on P04, setting both key-scan outputs driving low again — ready for the next scan or key-return input.

The scan_sw() routine sets KRIF to zero to clear any possible interrupts caused by the scanning process. This precaution is necessary because scanning can cause all key-return inputs to go high and then one or more of them to go low again, which can trigger the key-return interrupt. The routine inverts and masks the variable, so that a one represents a switch that is down, and a zero represents a switch that is up. The scan_sw() routine returns this value to the calling routine.

You can expand this procedure to accommodate a larger switch matrix by setting one out of several key scan outputs to drive low, with others as inputs, and reading a larger number of key-return input ports for each set of switches connected to the scan-output line.

2.3.12 MD_INTTM000( ) – Timer 00 Interrupt-Service Routine

Figure 16. Flowchart for Timer 00 Interrupt-Service Routine



INTTM000 invokes the MD_INTTM000() interrupt-service routine. INTTM000 is the interrupt that occurs when the TM00 timer-count register matches the CR000 timer compare register. In the demonstration program, this match occurs 500 milliseconds after tone-generation begins.

When using a timer for tone generation, the program needs to call the TM01_Stop() routine to end tone generation. If the tone is generated with the buzzer output, then you stop tone generation by calling the BUZ_Stop() routine. The demonstration program listings use TM01_Stop().

To prevent further interrupts every 500 milliseconds, stop Timer 00 with the TM00_Stop() routine.

2.4 Applilet's Reference Driver

NEC Electronics’ Applilet program generator automatically generates C or assembly-language source code to manage peripherals for NEC Electronics’ microcontrollers.

The Applilet produces the basic program-initialization code and main functions, as well as driver code for the key-return interrupt, for the timers, and for buzzer output. After the Applilet produces the basic code, you can add additional code to customize the program.

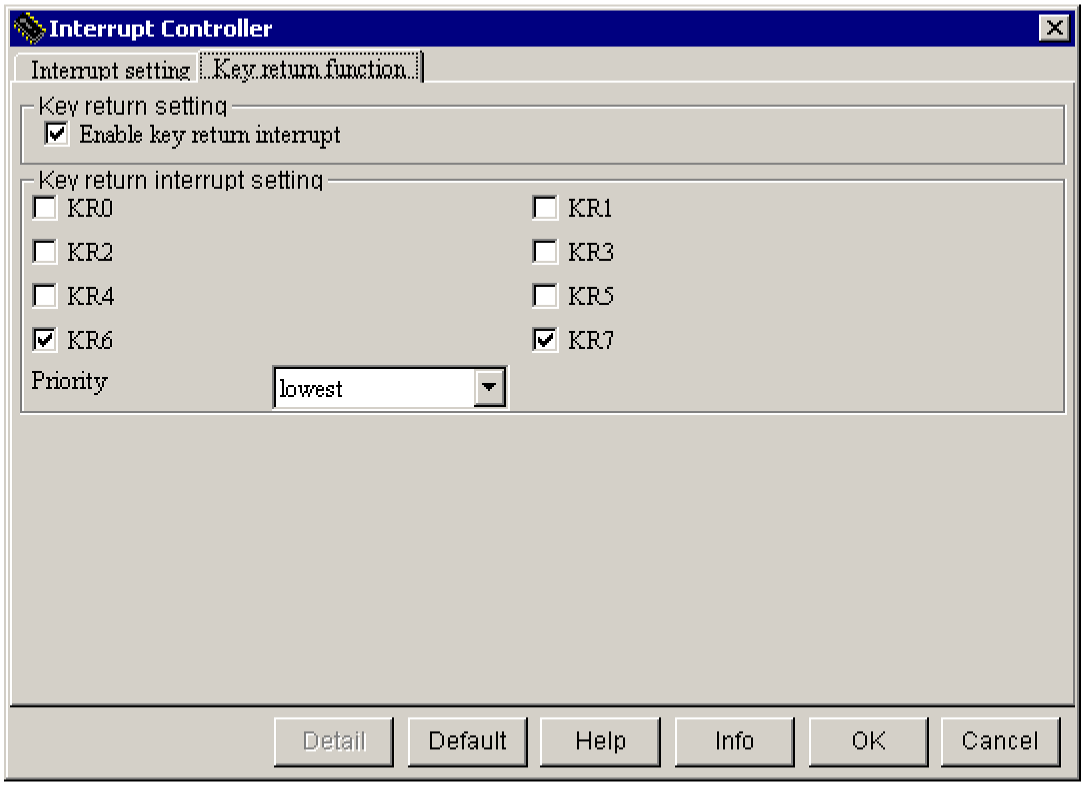
This section describes how to configure the Applilet to produce code for the key-return interrupt, timers, and buzzer output.

When you start the Applilet and select the target device, save your settings to a new project (.prx) file. The Applilet begins by displaying a dialog box that lets you select different peripheral blocks for setup.

2.4.1 Configuring Applilet for Key-Return Interrupt

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

Figure 17. Applilet Peripheral-Selection Screen



Key return setting

[✓] Enable key return interrupt

Key return interrupt setting

☐ KR0	☐ KR1
☐ KR2	☐ KR3
☐ KR4	☐ KR5
☑ KR6	☑ KR7

Priority: lowest

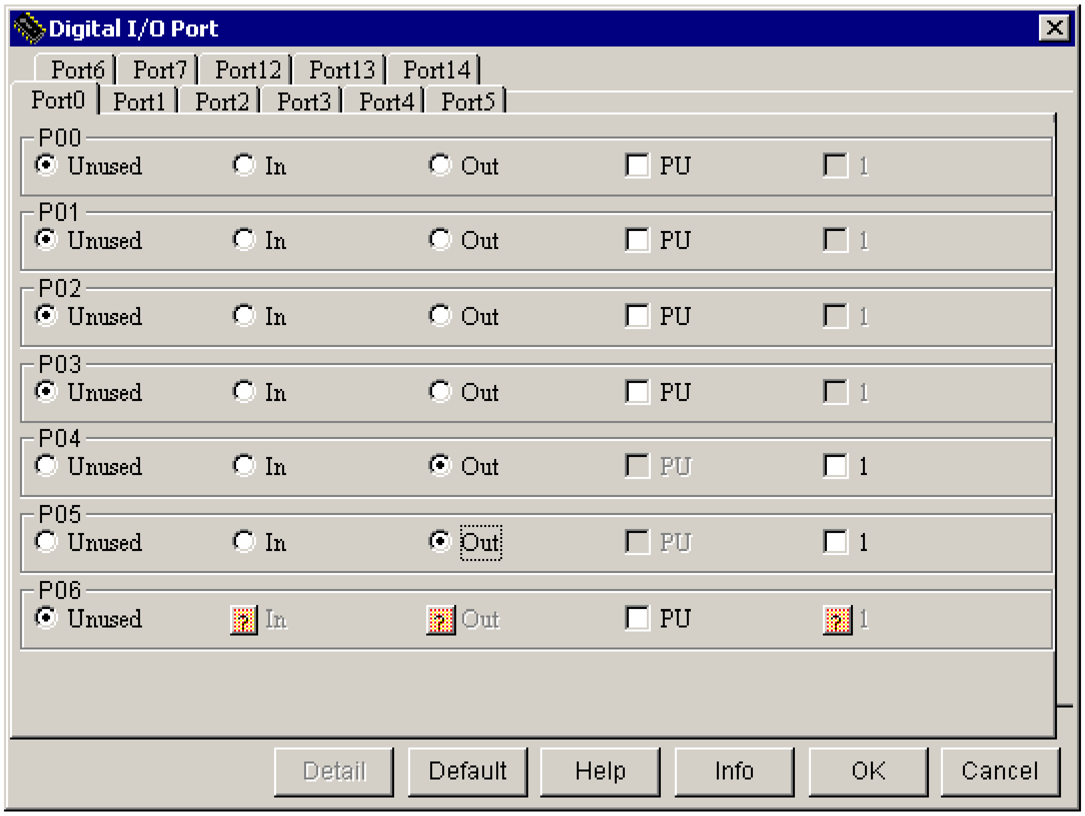
Check **Enable key-return interrupt** to generate an interrupt when the key-return pins go low. Set the priority for **lowest.**

Check the boxes for key-return interrupts **KR6** and **KR7**. The demonstration does not use KR0 through KR5 for key return, leaving pins P70/KR0 through P75/KR5 available as general-purpose I/O pins.

2.4.2 Configuring Applilet for Port Outputs Used as Key Scan Outputs

Select the **Digital I/O port** block to bring up a dialog box showing the I/O ports available. Select the **Port0** tab to set the options for port P0 pins.

Figure 18. Applilet Port-Selection Screen


Digital I/O Port
Port6 Port7 Port12 Port13 Port14
Port0 Port1 Port2 Port3 Port4 Port5
Detail Default Help Info OK Cancel

P00	[✓ Unused]	In	Out	☐ PU	☐ 1
P01	[✓ Unused]	In	Out	☐ PU	☐ 1
P02	[✓ Unused]	In	Out	☐ PU	☐ 1
P03	[✓ Unused]	In	Out	☐ PU	☐ 1
P04	Unused	In	[✓ Out]	☐ PU	☐ 1
P05	Unused	In	[✓ Out]	☐ PU	☐ 1
P06	[✓ Unused]	In	Out	☐ PU	☐ 1

Select pins **P04** and **P05** as key-scan output ports. The Applilet generates the code to initialize them in the PORT_Init() routine.

Note that the Applilet can configure pins as inputs, outputs or unused. When you choose a pin as an input, you can select optional pull-up resistors by checking the **PU** checkbox. This selection causes the corresponding bits to be set in the PU register for the port. Setting pins as outputs grays-out the **PU** checkbox so you cannot select it. The Applilet sets the corresponding PU register bits to zero for any pin selected as an output.

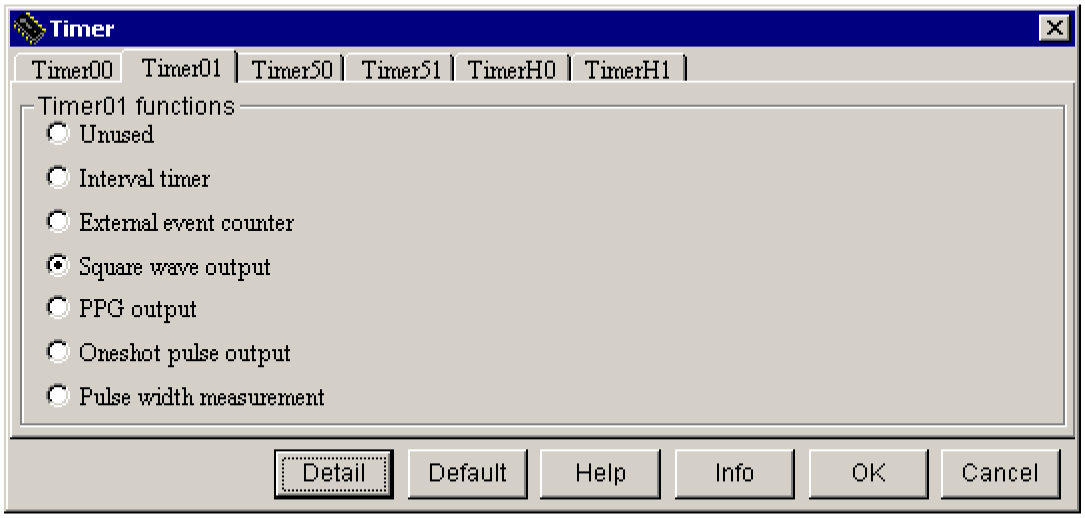
For the key-scan routine, the state of the port pins P04 and P05 dynamically changes from output (that drives low) to input. When the pins become inputs, the microcontroller connects the pins to internal pull-up resistors. The routine that scans the switches can make this change dynamically; but once the PU register is set to provide pull-ups on the input pins, you can leave it that way. The pull-up resistors have no effect when the port functions as an output.

Rather than select the pull-up resistors in the Applilet, the demonstration program changes the definition of PORT_PU0 in the file port.h. This change modifies the value written to the PU0 register in the PORT_Init() routine, so that pull-up resistors are enabled on pins P04 and P05 when you use them as inputs.

2.4.3 Configuring Applilet for Timer TM01 Square-Wave Generation

Select **Timer** to bring up a dialog box showing the various timer blocks. Select **Timer01** and click **Square-wave output**.

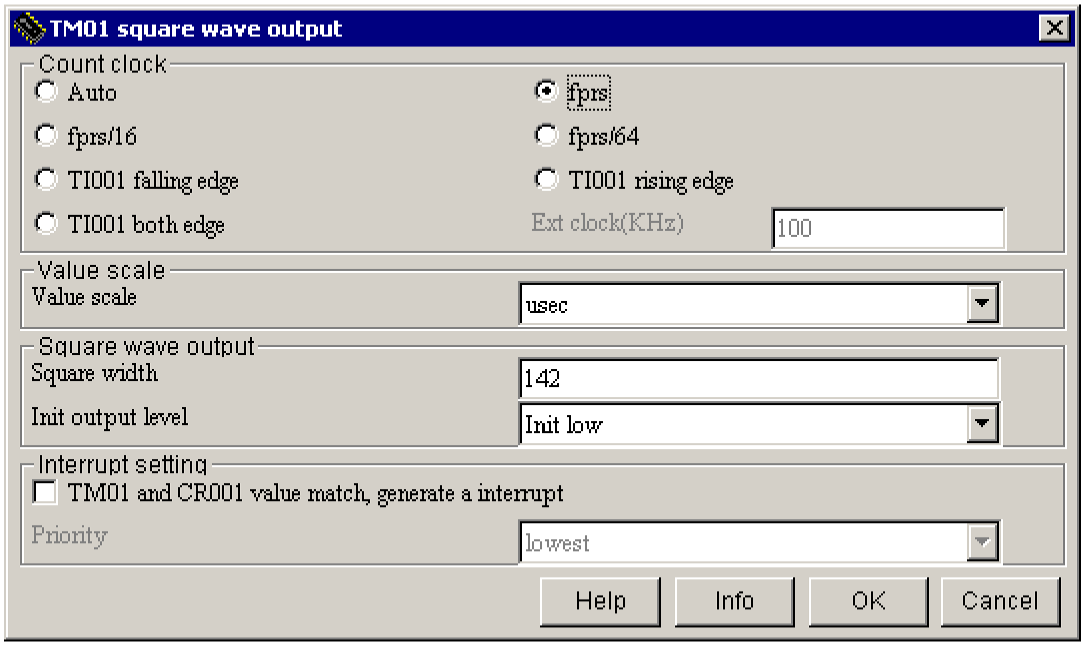
Figure 19. Applilet Timer-Selection Dialog



Timer01 functions
- Unused
- Interval timer
- External event counter
- [✓ Square wave output]
- PPG output
- Oneshot pulse output
- Pulse width measurement

Once you have selected **TM01**, click **Detail** for TM01 settings in the selected mode. This choice brings up the TM01 detail dialog box.

Figure 20. TM01 Detail Dialog Box


TM01 square wave output

Count clock
- Auto
- [✓ fprs]
- fprs/16
- fprs/64
- TI001 falling edge
- TI001 rising edge
- TI001 both edge
- Ext clock(KHz): 100

Value scale
- Value scale: usec

Square wave output
- Square width: 142
- Init output level: Init low

Interrupt setting
- ☐ TM01 and CR001 value match, generate a interrupt
- Priority: lowest

The timer should initially generate a 3520-Hz square wave. This frequency requires a cycle time of 1/3520 seconds and a half-cycle time of 1/7040 seconds, or about 142 microseconds. To generate a square wave with this frequency, enter 142 in **Square width** and set **Value scale** to microseconds.

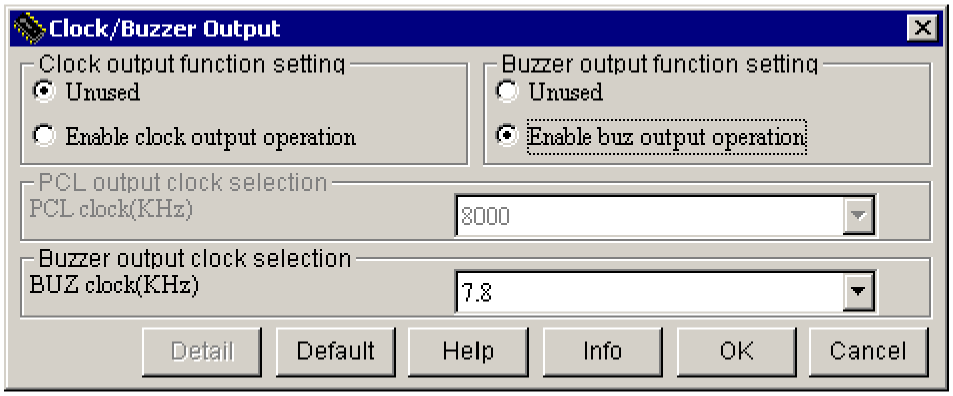
Set **Count clock** to f_{PRS}, to choose the main clock as the TM01 clock. The demonstration requires the fastest available clock to ensure that the square-wave frequency gets as close to the desired tone as possible. The Applilet calculates an appropriate setting for the timer comparison register, using the 142-microsecond square width and the 8-MHz peripheral clock.

Leave **Interrupt setting** unchecked to prevent the timer from generating an interrupt after the timer interval.

2.4.4 Alternative — Configuring Applilet for Buzzer Square-Wave Generation

Selecting **Buzzer** in the Applilet brings up a dialog box that lets you choose settings for buzzer and clock output.

Figure 21. Configuring Buzzer Output


Clock/Buzzer Output

Clock output function setting
( ) Unused [✓]
( ) Enable clock output operation

Buzzer output function setting
( ) Unused
( ) Enable buz output operation [✓]

PCL output clock selection
PCL clock(KHz) 8000

Buzzer output clock selection
BUZ clock(KHz) 7.8

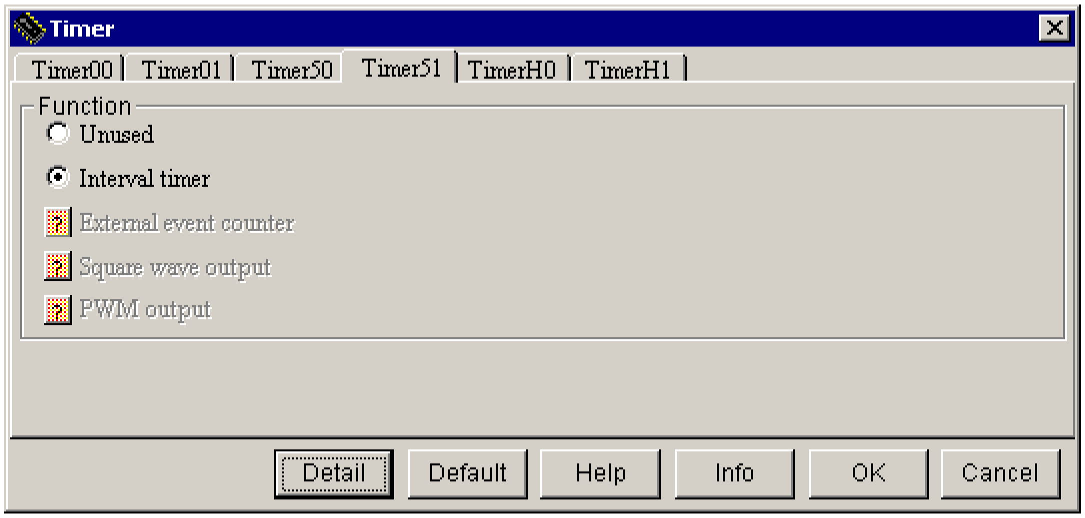
Select **Enable buz output operation**, and then you can select the buzzer-clock frequency from a drop-down menu. The main clock frequency determines the frequencies available in this menu. Select 7.8 kHz as the default, as this is the highest frequency available with an 8-MHz peripheral clock.

You do not use this dialog box for the demonstration program because the demonstration does not use the buzzer.

2.4.5 Setting Applilet to Configure Timer TM51 for 5-Millisecond Interval

In the Applilet's **Timer** selection dialog, select **Timer51** for use as an interval timer.

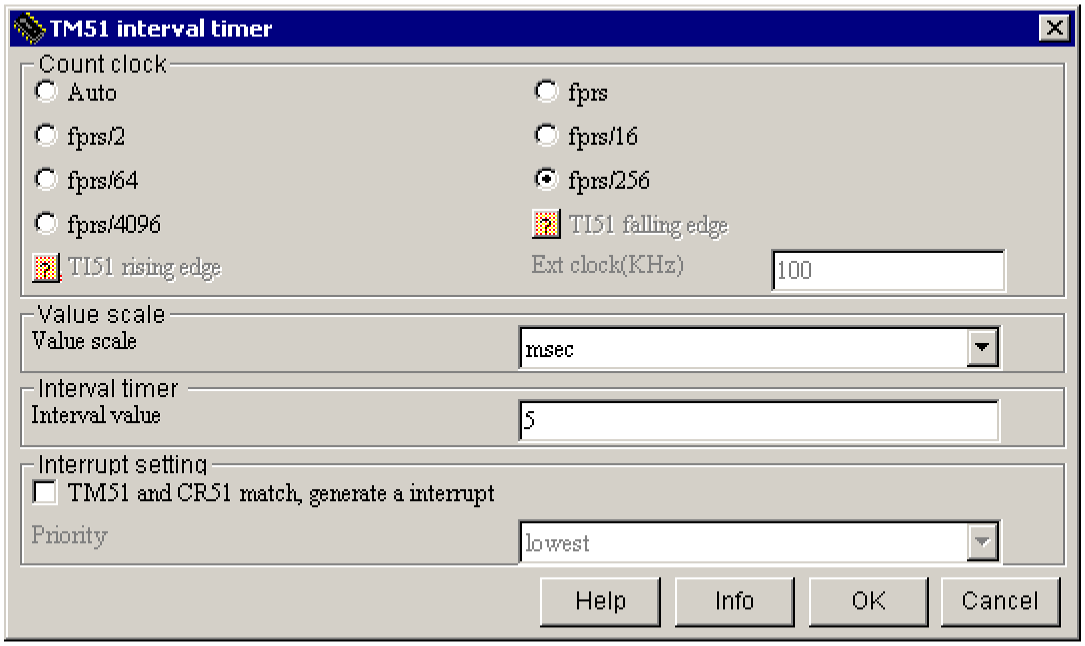
Figure 22. Configuring TM51 as Interval Timer


Timer

Function
( ) Unused
[✓ Interval timer]
External event counter
Square wave output
PWM output

Once you have set the TM51 function, click **Detail** to bring up the TM51 detail dialog box for interval-timer settings.

Figure 23. Detail Dialog for Interval-Timer Settings


TM51 interval timer

Count clock
- ( ) Auto
- ( ) fprs
- ( ) fprs/2
- ( ) fprs/16
- ( ) fprs/64
- (●) fprs/256
- ( ) fprs/4096
- TI51 falling edge
- TI51 rising edge

Ext clock(KHz): 100

Value scale
Value scale: msec

Interval timer
Interval value: 5

Interrupt setting
☐ TM51 and CR51 match, generate a interrupt
Priority: lowest

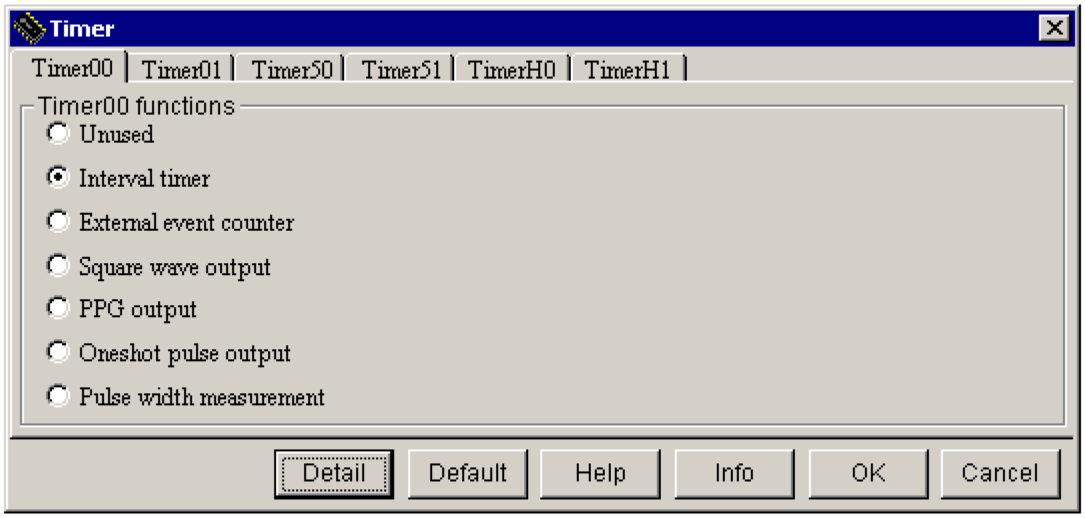
The timer should set its interrupt flag every 5 milliseconds, so set **Value scale** to **msec** (milliseconds) and **Interval value** to 5. Set **Count clock** to fprs/256 to define the basic clock as the main peripheral clock divided by 256. The Applilet calculates an appropriate setting for the timer-comparison register to create the 5-millisecond interval.

You do not want to generate an interrupt after the interval, so leave **Interrupt setting** unchecked. Instead of using this interrupt, the program checks the state of the TMIF00 flag to determine when the interval has elapsed.

2.4.6 Setting Applilet to Configure Timer TM00 for 500-Millisecond Interval

In the Applilet's **Timer** dialog box, select **Timer00** and check the box for use as an interval timer.

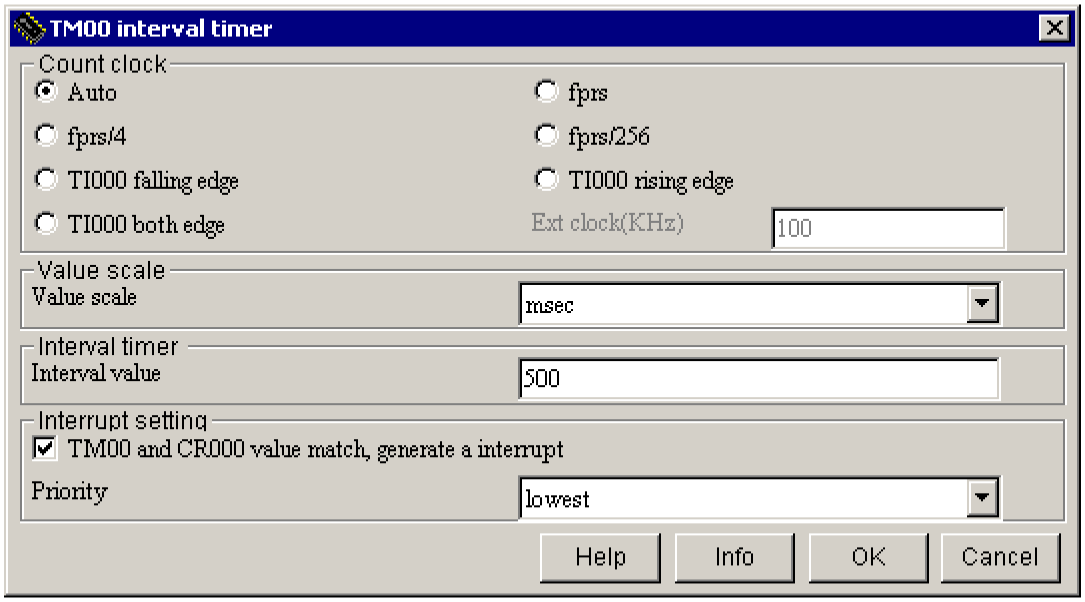
Figure 24. Selecting Timer00 as Interval Timer



Timer00 functions
- ○ Unused
- ● Interval timer
- ○ External event counter
- ○ Square wave output
- ○ PPG output
- ○ Oneshot pulse output
- ○ Pulse width measurement

Now click **Detail** to bring up the TM00 detail dialog box for interval-timer settings.

Figure 25. Setting Interval-Timer Operation



Count clock
- ● Auto
- ○ fprs
- ○ fprs/4
- ○ fprs/256
- ○ TI000 falling edge
- ○ TI000 rising edge
- ○ TI000 both edge

Ext clock(KHz): 100

Value scale
Value scale: msec

Interval timer
Interval value: 500

Interrupt setting
☑ TM00 and CR000 value match, generate a interrupt
Priority: lowest

This timer should generate an interrupt after 500 milliseconds. Set **Value scale** to **msec** (milliseconds) and **Interval value** to 500. Check **Auto** for the count clock to allow the Applilet to select an appropriate clock division as the TM00 clock. The Applilet uses the clock frequency and the desired interval to calculate an appropriate setting for the timer comparison register.

In this case, check **Interrupt setting** to generate an interrupt when TM00 (timer-count register) and CR000 (timer-compare register) match.

2.4.7 Generating Code with Applilet

Once you have set up the various dialog boxes, select **Generate code**. The Applilet displays the peripherals and functions, and lets you select a source-code directory.

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

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

To support TM00 and TM51, the Applilet generates timer.h, timer.c and timer_user.c. If you are using TM01 to generate the square wave, the Applilet also puts that code in these files.

If you are using the buzzer to generate the tones, the Applilet generates buzzer.h and buzzer.c.

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

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

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

2.4.8.1 Int.h

The header file int.h contains declarations for the functions controlling the key-return interrupt and definitions of values for key-return initialization. The header file macrodriver.h, used for all Applilet-generated code, also defines some data types and values, such as the MD_STATUS values, which some functions return.

You need to add code for the external declaration of the sw_in variable, used to report the state of the key-switch matrix. Other routines need to examine this variable by including int.h. You must also add definitions of the bit patterns to be set in the sw_in variable corresponding to switch 1 down, switch 2 down, etc.

2.4.8.2 Int.c

The source file Int.c contains functions for the key-return interrupt:

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

2.4.8.3 Int_user.c

The source file int_user.c contains stub functions for user code. When the Applilet generates these functions, they are empty.

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

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

unsigned char scan_sw(void)

You need to add this routine to provide a function that scans the key-switch matrix. It should return a value indicating which switch (or switches) is pressed.

Add the declaration of the sw_in variable to this file.

2.4.9 Applilet-Generated Files and Functions for TM00, TM01 and TM51

The files timer.h, timer.c and timer_user.c contain the code generated for TM00, TM01 and TM51 support.

2.4.9.1 Timer.h

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

2.4.9.2 Timer.c

The source file Timer.c contains the following functions generated by the Applilet for TM00, TM01 and TM51:

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

void TM00_Start(void)
The TM00_Start() routine enables Timer00 to start operation, and enables interrupt INTTM000

void TM00_Stop(void)
The TM00_Stop() routine disables the timer, stopping it, and disables the timer interrupt.

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, thus changing the timer's interval.

The array_reg parameter points to an array of values to be stored 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).

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, starting operation.

void TM01_Stop(void)
The TM01_Stop() routine disables the timer, stopping it.

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 counter-compare value for Timer 01.

The array_reg parameter points to an array of values to be stored in one or both of the compare registers; the array_num parameter is either 1 (to select CR001) or 2 (to select both CR011 and CR001).

void TM51_Init(void)
The TM51_Init() routine initializes the TM51 peripheral as specified in the Applilet TM51 detail dialog.

void TM51_Start(void)
The TM51_Start() routine starts TM51 operation by enabling the timer.

void TM51_Stop(void)
The TM51_Stop() routine stops TM51 operation by disabling the timer.

MD_STATUS TM51_ChangeTimerCondition(USHORT value)
The TM50_ChangeTimerCondition() function changes the value in the CR51 compare registers. This new value changes TM50's interval. The demonstration program does not use this routine.
Timer_user.c

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

__interrupt void MD_INTTM000(void)
This is the interrupt-service routine for Timer 00 interrupt INTTM000. A match of TM00 and CR000 values triggers the interrupt. Once you start the timer, it generates this interrupt every 500 milliseconds.

The Applilet generates a blank interrupt-service routine. You can add code to have the timer stop tone generation after 500 milliseconds. You must add code to stop the TM01 timer generating the tone, and to stop the TM00 timer itself.

2.4.10 Applilet-Generated Files and Functions for Buzzer Output

The demonstration program does not use the buzzer output, but the Applilet generates code to show the available functions. The listing section contains these files—buzzer.h and buzzer.c.

2.4.10.1 Buzzer.h

The header file buzzer.h contains declarations for the functions used to initialize, start and stop the buzzer.

2.4.10.2 Buzzer.c

The source file buzzer.c contains the following function generated by the Applilet for buzzer operation:

void BUZ_Init(void)
The BUZ_Init() routine initializes the buzzer.

void BUZ_Start(void)
The BUZ_Start() routine starts the buzzer by setting the BZOE bit to one in the CKS register.

void BUZ_Stop(void)
The BUZ_Stop() routine stops the buzzer by clearing the BZOE bit to zero in the CKS register.

2.4.11 Applilet-Generated Files and Functions for Port Initialization

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

2.4.11.1 Port.h

The header file port.h contains declarations for the PORT_Init() function, which initializes the ports and defines values for the initialization of port-output latches, port-mode registers, and port pull-up registers. For example, to initialize port P0, the initialization routine writes values defined in port.h to P0, PM0, and PU0.

You can edit the definitions to change the port-initialization values. For the demonstration program, you should modify the value of PORT_PU0 to set pull-up resistors on pins P04 and P05 when the key-scan routine does not use these pins as outputs.

2.4.11.2 Port.c

The source file port.c contains:

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

2.4.12 Other Applilet-Generated Files

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

Table 3. Additional Applilet-Generated Source Files

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

2.4.13 Demonstration Program Files Not Generated by Applilet

In this case, the Applilet generates all of the demonstration-program files.

2.5 Demonstration Platform

The demonstration platform for key return and buzzer 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.

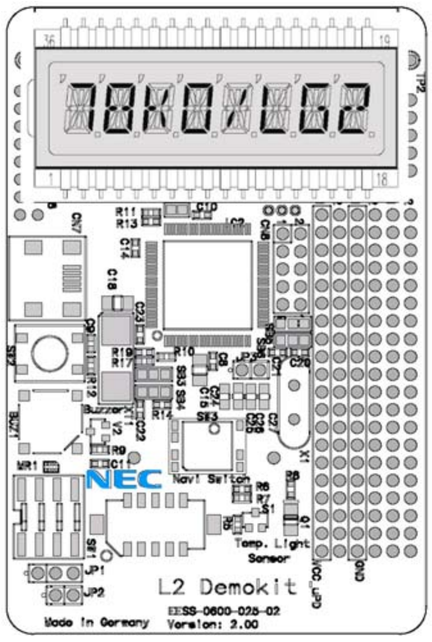
2.5.1 Resources

The demonstration uses the following resources:

- DemoKit-LG2 demonstration board, with uPD78F0397 8-bit microcontroller mounted
- DemoKit-LG2 resources:
  - BUZ1 buzzer driven by timer output P06/TO01
  - Added key-switch matrix to prototyping area, connected to P76-P77, P04-P05

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

Figure 26. Demonstration Platform



2.5.2 Program Demonstration

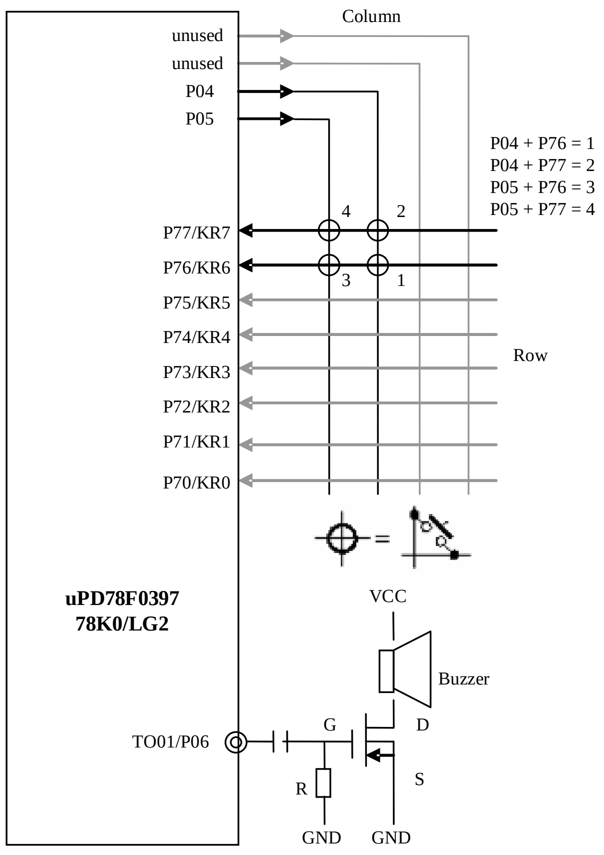
With the hardware configured and the uPD78F0397 microcontroller programmed with the demonstration code, the demonstration runs as follows:

- ♦ Press key-switch matrix keys 1, 2, 3, 4
- ♦ Observe the square wave generated at P06/TO01 on an oscilloscope
- ♦ Listen to the BUZ1 buzzer tone

2.6 Hardware Block Diagram

The 2x2 key switch matrix connects to key-return port pins P76/KR6 and P77/KR7. General-purpose port pins P05 and P04 provide key-scan outputs.

Figure 27. Hardware Block Diagram



Because the uPD78F0397 does not have a dedicated buzzer output, 16-bit timer/counter output pin P06/TO01 drives the buzzer with a square wave.

2.7 Software Modules

The demonstration program consists of software modules made up of the files shown in Table 4. The table indicates which files the Applilet generates and which you have to modify.

The listings for these files are located in the Appendix.

Table 4. 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
Int.h	Interrupt-related definitions	Yes	Yes[Note 1]
Int.c	Key return Interrupt-related functions	Yes	No
Int_user.h	User code for key return interrupt	Yes	Yes[Note 1]
Timer.h	Timer-related definitions	Yes	No
Timer.c	Timer functions	Yes	No
Timer_user.c	User code for timer interrupt handling	Yes	Yes[Note 2]
Port.h	Port-related definitions	Yes	Yes[Note 3]
Port.c	Port_Init() function	Yes	Yes[Note 3]
Option.inc	Option-byte, POC, and security definitions	Yes	No
Option.asm	Option-byte, POC, and security data	Yes	No

Note 1: You must modify Int.h to add the sw_in declaration (the variable used to store debounced input from the navigation switch) and definitions for switch input values. Int_user.c requires the addition of the variable sw3_in, additional code for handling the key-return interrupt in MD_INTKR(), and the addition of the scan_sw() routine.

Note 2: Timer_user.c requires the addition of code to handle the INTTM000 interrupt in the MD_INTTM000() routine, which turns off the tone generation.

Note 3: The version of the Applilet used for generating source code for the demonstration does not support the 78K0/LG2 (uPD78F0397) device, but does support the 78K0/Kx2 family of devices, which are very similar. The Applilet generates the files port.h and port.c for the uPD78F0537_64 in the 78K0/Kx2 family. Port.c must be modified for the uPD78F0397 by commenting out the lines in PORT_Init() which set ports not available on the uPD78F0397. You must modify Port.h to change the pull-up register (PU0) initialization value, to provide pull-ups on key scan-output port lines P04 and P05.

The following files, not used in the demonstration program, support buzzer output.

Table 5. Buzzer-Output Files

File	Purpose	Generated By Applilet	Modified By User
Buzzer.h	Buzzer routine declarations	Yes	No
Buzzer.c	Buzzer-related routines	Yes	No

3. Appendix A — Development Tools

This application note uses the following software and hardware tools.

3.1 Software Tools

Table 6. Software Tools Used for Application Note

Tool	Version	Comments
Applilet for 78K0KX2	V1.51	Source-code generation tool for 78K0/KE2 devices
PM Plus	V5.20	Project manager for program compilation and linking
CC78K0	V3.60	C Compiler for NEC Electronics’ 78K0 devices
RA78K0	V3.70	Assembler for NEC Electronics’ 78K0 devices
DF0397.78K	V1.01	Device file for uPD78F0397 device

Note: The version of the Applilet used produces code for the 78K0/Kx2 family of microcontrollers, in this case, the 78K0/KE2 (uPD78F0537_64). This device is very similar to the 78K0/LG2 device (uPD78F0397) used in the DemoKit-LG2.

3.2 Hardware Tools

Table 7. Hardware Tools Used for Application Note

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

4. Appendix B – Software Listings

Note: Although the files buzzer.h and buzzer.c do not appear in the demonstration program, they illustrate support for buzzer output.

4.1 Main.c

```
/*
******************************************************************************
**
** This device driver was created by Applilet for the 78K0/KB2, 78K0/KC2,
** 78K0/KD2, 78K0/KE2 and 78K0/KF2 8-Bit Single-Chip Microcontrollers.
**
** Copyright(C) NEC Electronics Corporation 2002 - 2005
** All rights reserved by NEC Electronics Corporation.
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : main.c
** Abstract : This file implements main function
** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler: NEC/CC78K0
**
******************************************************************************
*/
/*
******************************************************************************
** Include files
******************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.h"
#include "port.h"
#include "timer.h"
/*
******************************************************************************
** MacroDefine
******************************************************************************
*/
/* define one or the other to select tone generation type */
#define TONEGEN_TM01       1
#define      TONEGEN_BUZ         0

/*----------------------------------------------------------------------------
** Tone( toneno ) function
** parameter toneno = 1 - 4
** start tone generation with selected tone
**----------------------------------------------------------------------------
*/
#if (TONEGEN_TM01 == 1)
/* tone 1 = 3520 Hz, from original 142 millisecond squarewave */
#define TONE_1      TM_TM01_INTERVALVALUE
/* tone 2 = 1760 Hz, with twice the count value */
#define TONE_2      ((TONE_1 * 2) + 1)
/* tone 3 = 880 Hz, twice again the count value */
```

```
#define TONE_3      ((TONE_2 * 2) + 1)
/* tone 4 = 440 Hz, twice again the count value */
#define TONE_4      ((TONE_3 * 2) + 1)

/* table of tone values to use */
USHORT ToneTab[4] = {
       TONE_1,
       TONE_2,
       TONE_3,
       TONE_4
};

void Tone(UCHAR toneno)
{
USHORT tone[2];     /* variable array to hold tone setting value */

       /* check for proper parameter */
       if ((toneno < 1) || (toneno > 4))
              return;

       /* stop tone output in case it is still going on */
       TM01_Stop();

       /* set tone[0] to appropriate value from table */
       tone[0] = ToneTab[toneno - 1];

       /* set the tone count value in the CR001 compare register */
       TM01_ChangeTimerCondition(&tone[0],1);
       TM01_Start(); /* start square wave generation */
}
#endif
#if (TONEGEN_BUZ == 1)
/* tone 1 = 7812.5 Hz, CKS.6,5 = 00 to select fPRS/1024 */
#define TONE_1      0x00
/* tone 1 = 7812.5 Hz, CKS.6,5 = 01 to select fPRS/2048 */
#define TONE_2      0x20
/* tone 1 = 7812.5 Hz, CKS.6,5 = 10 to select fPRS/4096 */
#define TONE_3      0x40
/* tone 1 = 7812.5 Hz, CKS.6,5 = 11 to select fPRS/8192 */
#define TONE_4      0x60
/* mask to clear CKS bits 6,5 */
#define TONE_MASK 0x9F

/* table of tone values to use */
UCHAR ToneTab[4] = {
       TONE_1,
       TONE_2,
       TONE_3,
       TONE_4
};

void Tone(UCHAR toneno)
{
UCHAR cks_val;

       /* check for proper parameter */
       if ((toneno < 1) || (toneno > 4))
              return;

       /* stop tone output in case it is still going on */
       BUZ_Stop();
```

```
        /* get current CKS value, mask out buzzer frequncy bits */
        cks_val = CKS & TONE_MASK;

        /* set buzzer frequency bits from table entry selected */
        cks_val = cks_val | ToneTab[toneno - 1];

        /* update CKS register to set buzzer frequncy */
        CKS = cks_val;

        /* start the buzzer output */
        BUZ_Start();
}
#endif


/*
**-----------------------------------------------------------------------------
**
** Abstract:
**      main function
**
** Parameters:
**      None
**
** Returns:
**      None
**
**-----------------------------------------------------------------------------
*/
void main( void )
{
        IMS = MEMORY_IMS_SET;
        IXS = MEMORY_IXS_SET;
        /* TODO. add user code */
        sw_in = 0;    /* clear switch value */

        while(1){
                if (sw_in != 0) {
                        if (sw_in == SWITCH_1) {
                                /* set the tone associated with this switch */
                                Tone(1);
                                TM00_Start(); /* start 500 msec timer */
                        }
                        if (sw_in == SWITCH_2) {
                                /* set the tone associated with this switch */
                                Tone(2);
                                TM00_Start(); /* start 500 msec timer */
                        }
                        if (sw_in == SWITCH_3) {
                                /* set the tone associated with this switch */
                                Tone(3);
                                TM00_Start(); /* start 500 msec timer */
                        }
                        if (sw_in == SWITCH_4) {
                                /* set the tone associated with this switch */
                                Tone(4);
                                TM00_Start(); /* start 500 msec timer */
                        }
                        /* clear switch input */
                        sw_in = 0;
                } /* end if sw_in != 0 */
        } /* end while (1) loop */
}       /* end main() */
```

4.2 Macrodriver.h

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

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

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

#define      ON     1
#define      OFF    0

#define      TRUE   1
#define      FALSE  0

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

#define SET   1
#define CLEAR 0

#define MD_STATUS          unsigned short
```

```
#define MD_STATUSBASE            0x0

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

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

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

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

enum INTLevel { Highest, Lowest };


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


#endif
```

4.3 System.h

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

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

void Clock_Init( void );

#endif
```

4.4 Systeminit.c

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

```
*******************************************************************************
** Include files
*******************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "int.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();
      /* INT initiate */
      INT_Init();
      /* TM00 initiate */
      TM00_Init();
      /* TM01 initiate */
      TM01_Init();
      /* TM51 initiate */
      TM51_Init();
}

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

4.5 System.c

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

/*
**------------------------------------------------------------------------------
**
** Abstract:
**     Init the Clock Generator and Oscillation stabilization time.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**------------------------------------------------------------------------------
*/
void Clock_Init( void )
{
      ClrIORBit(MCM, 0x05);                  /* High-Internal-OSC operate for CPU */

      SetIORBit(MCM, 0x01);                  /* peripheral hardware clock:frh */
```

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

4.6 Int.h

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

#ifndef      _MDINT_
#define      _MDINT_
/*
*******************************************************************************
** MacroDefine
*******************************************************************************
*/
#define      EGP_INT      0x0
#define      EGN_INT      0x0
#define      PU7_KR 0xc0
#define      PM7_KR 0xc0
#define      KRM_KR 0xc0

enum ExternalINT {
	EX_INTP0, EX_INTP1, EX_INTP2, EX_INTP3,
	EX_INTP4, EX_INTP5, EX_INTP6, EX_INTP7
	};

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

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

/* added definitions of switch inputs */
#define SWITCH_1    0x01
#define SWITCH_2    0x02
#define SWITCH_3    0x04
#define SWITCH_4    0x08

/* added declaration of switch variable */
extern unsigned char sw_in;

#endif
```

4.7 Int.c

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

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

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

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

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

4.8 Int_user.h

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

#pragma      interrupt    INTKR  MD_INTKR

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

```
#include "int.h"
/* added include of timer.h to access timer functions */
#include "timer.h"

/*
******************************************************************************
** MacroDefine
******************************************************************************
*/
/* added definition of sw_in variable to hold switch data */
unsigned char sw_in;

/* added function to scan switch matrix
** Parameters: none
** Return: unsigned char with
** bit 0 set if sw 1 down
** bit 1 set if sw 2 down
** bit 2 set if sw 3 down
** bit 3 set if sw 3 down
*/
unsigned char scan_sw(void)
{
unsigned char value;
unsigned char uc;

       PM0.5 = 1;                         /* make P05 an input to turn drive off; P04
is still driving low */
       NOP();                             /* short wait before reading inputs to allow
pull-up to bring line high */
       NOP();
       value = (P7 >> 6) & 0x03;  /* bits 1 and 0 = P77, P76 with P04 low */
                                                 /* bit 0 = 0 if sw 1 down, bit 1 = 0
if sw 2 down, otherwise 1 */

       PM0.5 = 0;                         /* make P05 an output again to drive low */
       PM0.4 = 1;                         /* make P04 an input to turn drive off */
       NOP();                             /* short wait */
       NOP();
       value = value | ((P7 >> 4) & 0x0C);     /* bits 3 and 2 = P77, P76 with P05 low */
                                                 /* bit 2 = 0 if sw 3 down, bit 3 = 0
if sw 4 down, otherwise 1 */

       PM0.4 = 0;                         /* make P04 an output again, both output
lines driving low */

       KRIF = 0;                          /* clear key return interrupt flag which may
have been triggered */
       return (~value & 0x0F);          /* return bits 3-0 = 1 if switch down, = 0 if up */
}

/*
**---------------------------------------------------------------------------
**
** Abstract:
**     INTKR Interrupt service routine.
**
** Parameters:
**     None
**
** Returns:
**     None
**
```

```
**-------------------------------------------------------------------------------
*/
__interrupt void MD_INTKR( void )
{
unsigned char sw_first, sw_second;

       sw_first = scan_sw();      /* get first switch input */
 TM51_Start();                    /* start 5 ms. timer */
 while(!TMIF51);                  /* wait for Timer51 Interrupt */
 TM51_Stop();              /* stop the timer */
 TMIF51=0;                        /* clear flag */
 sw_second = scan_sw();    /* read switches second time */
 if (sw_first == sw_second)
 sw_in = sw_first;  /* switches are stable, report value */
 else
 sw_in = 0;                /* otherwise no input */
}
```

4.9 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 0x2
#define      TM_TM00_INTERVALVALUE      0x3d08
#define      TM_TM00_SQUAREWIDTH 0x3d08
#define      TM_TM00_PPGCYCLE    0x3d08
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE       0x3d08
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
```

```
#define      TM_TM01_INTERVALVALUE     0x46f
#define      TM_TM01_SQUAREWIDTH 0x46f
#define      TM_TM01_PPGCYCLE    0x46f
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE      0x46f
#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     0x9b
#define      TM_TM51_SQUAREWIDTH 0x9b
#define      TM_TM51_PWMACTIVEVALUE    0x9b
#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);
void TM01_Init(void);
void TM51_Init(void);

/*timer start*/
void TM00_Start(void);
void TM01_Start(void);
void TM51_Start(void);

/*timer stop*/
void TM00_Stop(void);
void TM01_Stop(void);
void TM51_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TM01_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TM51_ChangeTimerCondition(UCHAR value);
__interrupt void MD_INTTM000(void);

#endif       /* _MDTIMER_*/
```

4.10 Timer.c

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

```
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : timer.c
** Abstract : This file implements a device driver for the timer module
** APIlib: NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :  uPD78F0537
**
** Compiler: NEC/CC78K0
**
********************************************************************************
*/

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


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

/*TM01 pulse width measure*/

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

```
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM00_Start()
{
       TMC00 = 0x0c;                      /* interval timer start */
       ClrIORBit(MK0H, 0x40);                   /* INTTM000 enable */
}

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

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

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function can initialize TM01_module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM01_Init( )
{
       TMC01=0x00;
                                        /* internal count clock */
       PRM01 |= TM_TM01_CLOCK;
       /* TM01 squarewave output */
       ClrIORBit(CRC01,0x01);
       CR001 = TM_TM01_SQUAREWIDTH;
       CR011 = 0xffff;

       ClrIORBit(P0,0x40);              /* TO01(p06) as output */
       ClrIORBit(PM0,0x40);
       TOC01=0x7;                       /* init low */
}
/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function start the TM01 counter.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM01_Start(void)
{
       TMC01 = 0x0c;             /* squarewave 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 TM51_module.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void TM51_Init( void )
{
       ClrIORBit(TMC51, 0x80);
       TCL51 = TM_TM51_CLOCK;                 /* countclock=fx/256 */
       /* 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);
}

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

4.11 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"


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

/* Timer00, Timer01 pulse width measure */
/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     TM00 INTTM000 interrupt service routine
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
      /* when TM00 times out, stop TM01 to turn off sound */
      TM01_Stop();
      /* and stop TM00 - only want one interrupt */
      TM00_Stop();
}
```

4.12 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      0xcf
#if 0  /* change definition of default pull-ups for port P0 (all off) */
#define       PORT_PU0      0x0
#else  /* set bits 4 and 5 to 1, so P04 and P05 will have pull-up resistors when inputs */
#define       PORT_PU0      0x30
#endif
#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      0x0
#define       PORT_P3       0x0
#define       PORT_PM4      0xff
#define       PORT_PU4      0x0
#define       PORT_P4       0x0
#define       PORT_PM5      0xff
#define       PORT_PU5      0x0
#define       PORT_P5       0x0
#define       PORT_PM6      0xff
#define       PORT_P6       0x0
#define       PORT_PM7      0xff
#define       PORT_PU7      0x0
#define       PORT_P7       0x0
#define       PORT_PM12     0xff
#define       PORT_PU12     0x0
#define       PORT_P12      0x0
#define       PORT_P13      0x0
#define       PORT_PM14     0xff
#define       PORT_PU14     0x0
```

```
#define      PORT_P14      0x0
#define      PORT_ADPC     0x0

void PORT_Init( void );

#endif
```

4.13 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;
/* remove ports which do not exist for uPD78F0397 */
/*     P4 = PORT_P4; */
/*     P5 = PORT_P5; */
/*     P6 = PORT_P6; */
       P7 = PORT_P7;
       P12 = PORT_P12;
       P13 = PORT_P13;
/*     P14 = PORT_P14; */

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

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

4.14 Option.inc

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

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

;
;*****************************************************************************
;** MacroDefine
;*****************************************************************************
;
OPTION_BYTE   EQU    00H
POC81  EQU    00H
POC82  EQU    00H
POC83  EQU    00H
CG_ONCHIP     EQU    02H
CG_SECURITY0  EQU    0ffH
CG_SECURITY1  EQU    0ffH
CG_SECURITY2  EQU    0ffH
CG_SECURITY3  EQU    0ffH
CG_SECURITY4  EQU    0ffH
CG_SECURITY5  EQU    0ffH
CG_SECURITY6  EQU    0ffH
CG_SECURITY7  EQU    0ffH
CG_SECURITY8  EQU    0ffH
CG_SECURITY9  EQU    0ffH
```

4.15 Option.asm

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

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

```
       ONC_SET CSEG AT 84H
ONCHIP:      DB     CG_ONCHIP

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

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

#ifndef      _MDBUZ_
#define      _MDBUZ_
/*
*******************************************************************************
** MacroDefine
*******************************************************************************
*/
void BUZ_Init( void );
void BUZ_Start( void );
void BUZ_Stop( void );

#endif
```

4.17 Buzzer.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 : buzzer.c
** Abstract : This file implements device driver for buzzer module.
** APIlib :   NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
** Device :   uPD78F0537
**
** Compiler : NEC/CC78K0
**
*******************************************************************************
*/

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

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function initializes the buzzer output controller.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void BUZ_Init( void )
{
      BZOE = 0;                  /* stop BUZ */

      ClrIORBit(P14, 0x02);
      ClrIORBit(PM14, 0x02);            /* set p141 BUZ output mode */

      ClrIORBit(CKS, 0x60);             /* BUZ output clock: fprs/2^10 */
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function enable buzzer output operation.
```

```
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void BUZ_Start( void )
{
      BZOE = 1;                 /* start BUZ */
}

/*
**-----------------------------------------------------------------------------
**
** Abstract:
**     This function disable buzzer output operation.
**
** Parameters:
**     None
**
** Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void BUZ_Stop( void )
{
      BZOE = 0;                 /* stop BUZ */
}
```