

1 Example Laboratory Report for a Project Requiring a Microprocessor

CDR Charles B. Cameron

12 February 2002

1.1 Purpose

The purpose of this report is to give students an example of a practical design problem for EE461 Microcomputer-Based Digital Design in which a microprocessor is included. The report shows how to design a circuit containing the microprocessor and describes the program within it.

The objective of the circuit is threefold:

1. Generate a 1-Hz output.
2. Use interrupts to make the frequency accurate.
3. Demonstrate how to use semaphores so the interrupt service routine can communicate with a foreground program with great reliability.

Because the program entails the use of real-time interrupts, it is difficult to present information on how to use the microprocessor's simulation program to verify correct operation. Nonetheless, that difficult question is also discussed in this report.

1.2 Equipment

Agilent 54622D Mixed Signal Oscilloscope

Digi-Design Prototyping Board

1.3 Design

1.3.1 Circuit Schematic

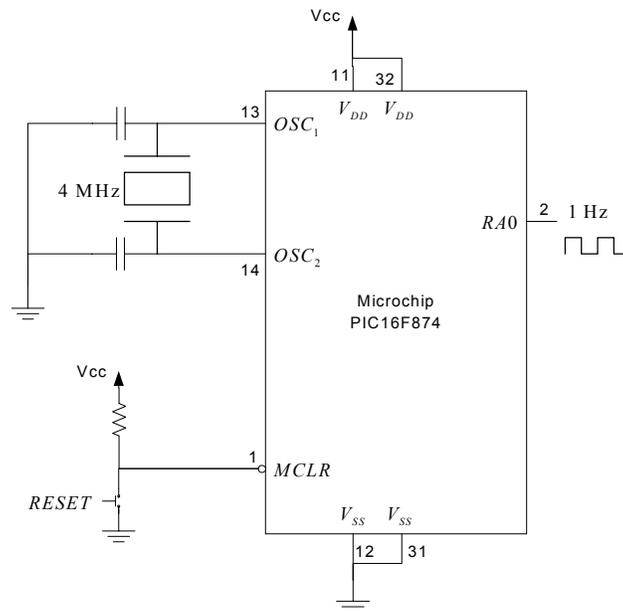


Figure 1: Schematic of connections to the PIC16F874

Several features of the circuit schematic diagram in Figure 1 are worth mentioning.

The oscillator inputs OSC_1 and OSC_2 are connected in the HS mode, a mode which must be specified when the microprocessor program is downloaded to the chip. The crystal selected has a frequency of 4 MHz. Since the PIC16F874 divides this clock rate by four internally, the result is that one instruction cycle takes one microsecond.

The PIC16F874 has two connections for both power and ground. The \overline{MCLR} input is an active-LOW signal and must be held high for normal operation. We have connected it to a switch to permit a manual reset when desired.

Bit 0 of Port A (signal $RA0$) is used to output a 1-Hz signal from the circuit.

1.3.2 Program

Appendix A contains a complete listing of the PIC16F874 program which produces the required output signal.

The included file **p16f874.inc** includes definitions of symbols not recognized inherently by the assembler.

When an interrupt occurs, it is essential that the current context be saved. This means that the contents of the **Status** register be saved. Storage is reserved in the processor's internal RAM beginning at hexadecimal address 0x20 for this purpose.

Bits in the word **InterprocessFlags** permits the interrupt service routine (ISR) to communicate with the foreground program. The ISR has sole permission to alter the **TimerInt** bit in this word. The foreground program has sole permission to alter the **TimerAck** bit. When the ISR

wants to notify the foreground program that one half second has elapsed, it asserts the **TimerInt** bit. The foreground program eventually notices this and asserts the **TimerAck** bit in acknowledgement. The next time the ISR gets control, it notices the acknowledgement and withdraws (de-asserts) the **TimerInt** bit. Soon the foreground program notices this and withdraws (de-asserts) the **TimerAck** bit, completing the exchange of information.

The **Light** bit of Port A is used to generate a 1-Hz signal. This is bit 0 of the port. A mask with a 1 in this position is also created to simplify accessing this bit later in the program.

Together, **PulseCount1** and **PulseCount0** comprise a 16-bit counter. The ISR will update this counter 500 times and then notify the foreground program as described above. This means that the ISR notifies the foreground program twice each second. All the foreground program needs to do to generate the output signal is toggle it each time it receives such a notification and provide an acknowledgement to the ISR.

When power is applied to the system or after a **Reset** signal has occurred, the program starts execution at location 0. The instruction stored there clears the **PCLATH** register, causing all subsequent instructions to be fetched from page 0 of the Flash program memory. (If the program were large enough we might have to manipulate the contents of this register later but as it happens this program is not very big at all.)

The interrupt service routine must begin at location 4 in the PIC16F874 architecture so the main program skips over it. Its description is presented in section 1.3.8. At label **main** the main program resumes.

1.3.3 Initialization of Port A

All the bits of Port A are initialized as output pins. However, the only bit RA0 is used. It carries the 1-Hz signal outside the chip.

1.3.4 Initialization of Timer 2

Timer 2 is then initialized to create a 1-ms interval between successive interrupts. Subsequently we can count off 500 of these and know that a halfsecond has elapsed. By choosing a prescaler value of 4, the value 250 in register **PR2**, and a postscaler value of 1 we obtain $4 \times 250 \times 1 = 1000$ 1- μ s instruction cycles between successive interrupts. Once configured, interrupts are enabled and then the timer is started.

1.3.5 Other Initialization Steps

The **InterprocessFlags** are set to zero, signifying that the ISR has not yet notified the foreground program that 500 ms have elapsed and the foreground program has not acknowledged such a notification.

The 16-bit pulse counter is initialized. Its operation is discussed in section 1.3.6.

Finally, the **Light** bit of Port A is extinguished. It therefore is off to begin with.

1.3.6 Initializing the Pulse Counter

Initializing the Pulse Counter entails putting the value 499 into the 16-bit register. The value 499 is appropriate because one 1-ms interval will elapse for each value of the counter from 499 down to 0, inclusive, making 500 intervals or 500 ms in all.

1.3.7 Main Program

The main or foreground program continually checks to see if the ISR has announced completion of a 500-ms interval. If so, the main program toggles the value of the **Light** bit of Port A. Otherwise it simply keeps looking.

The rest of the main program is devoted to taking care of its end of the communication with the ISR. When it discovers the **TimerInt** bit of the **InterprocessFlags** has been asserted, it asserts the **TimerAck** bit of the same word. The next step in the protocol is to await withdrawal of the **TimerInt** signal. When this happens it withdraws the **TimerAck** bit, completing the exchange of information.

1.3.8 Interrupt Service Routine

1.3.8.1 Recognizing the Interrupt

Since the PIC16F874 only has one interrupt vector, the first task upon entry into the ISR is to figure out which interrupt took place. Although this program has only a single enable interrupt and we could therefore bypass this step, the program shows how to recognize a particular interrupt and then invoke the subroutine **Timer2** to handle it.

Before reaching this part of the code the ISR saves the values of the **W** register and of the **STATUS** register. Subsequently it restores these values. The scheme used is available in Microchip's documentation. Its use of the **SWAPF** instruction is a little obscure. It is used because it does not affect the condition codes in the **STATUS** register whereas the various move instructions do affect them.

1.3.8.2 Handling the Timer Interrupt

Upon entry into the ISR there are two possibilities: either the ISR is waiting for acknowledgement from the foreground program of an earlier notification of the completion of a 500-ms interval or it is not. In either case the ISR must properly count the timer interrupt which caused it to get control in the first place. After doing this, however, it must handle the two situations appropriately.

If it is not awaiting an acknowledgement then it is free to set the **TimerInt** bit of the **InterprocessFlags** register if, in fact, 500-ms have elapsed.

Otherwise, it should refrain from issuing another notification until the acknowledgement has been withdrawn.

We expect the protocol to be completed in well under 500 ms. In general, if this were not a valid assumption, we could alter the ISR to count the number of notifications it wanted to generate but was inhibited from generating. When the (slow) foreground program finally withdrew the acknowledgement indication, the ISR could pass that count instead of just passing a single bit. In our case, it is not necessary.

1.4 Experimental Observations

During the debugging of the program the PIC16F874 simulator was used to great advantage. The principle difficulty in using it was that the program takes much longer to execute than it does in the target hardware. In order to reduce this time, we single-stepped through the program

and changed the values of Timer 2 with the debugger in order to hasten the execution. For example, when Timer 2 needed to count from 0 to 255 before an interrupt would be generated, we simply altered the value manually, causing the ISR to be entered earlier than normal. When the ISR was deciding whether all 500 1-ms intervals had elapsed, we often just changed the value of the 16-bit counter **PulseCounter** to achieve this goal early. This allowed us to check the logic associated with communications between the ISR and the foreground program in a reasonable amount of time.

Once this debugging had been satisfactorily completed, we downloaded the program into a chip and it produced a nearly correct result. However, the measured period of the signal was 1.005 s, not 1.000 s. While this error was small, it still seemed excessive. We discovered that 501 ms, not 500 ms, was occurring between successive notifications by the ISR to the foreground program. Once this was corrected, we got the results shown in Figure 2.

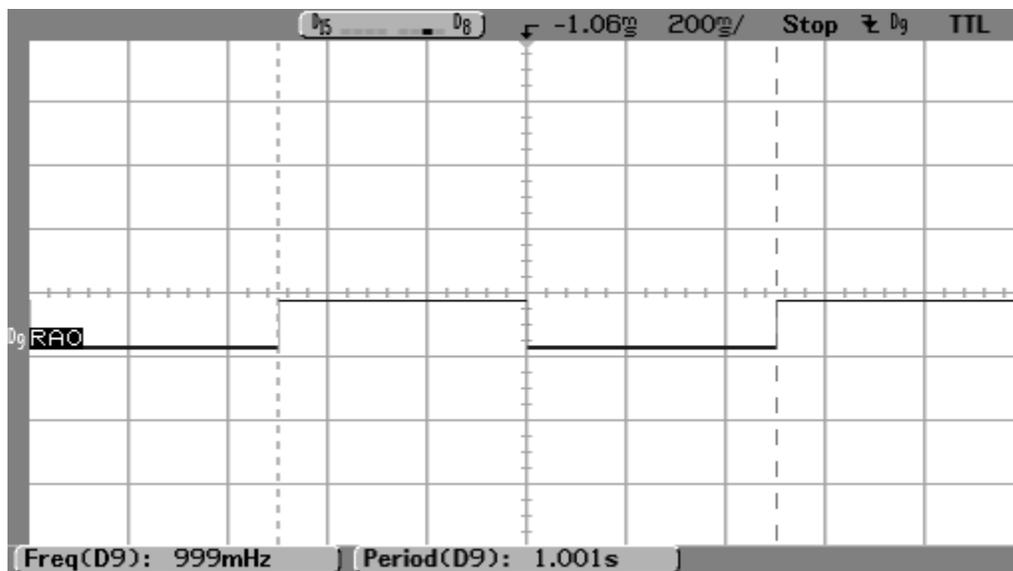


Figure 2: Oscilloscope Display of the Output of the PIC16F874

Figure 2 shows an Agilent 54622D oscilloscope display of the final results of the design. Bit **RA0** is bit 0 of Port A and it corresponds to the **Light** bit of that port, as it is referred to in the program.

The display reveals that the period of the signal is 1.001 s, very close to the desired 1.000 s.

1.5 Conclusions

The design was successful in creating a 1-Hz signal to within 1 part in 1000. Whether this error is real or an artifact of the Agilent 54622D oscilloscope itself is not clear and might be worthy of further investigation. Doing so could be as simple as decreasing the time scale. Possibly the oscilloscope is not capable of better than 1 part in 1000 of accuracy but this, too, could be checked if the discrepancy needed to be explained better.

The protocol for communication between an ISR and the foreground program is very robust and while not strictly necessary in this easy project it would be very useful in a more complicated situation. It is necessary in general because the foreground can never be sure just when the ISR

will execute and the ISR can never be sure just where in its cycle the foreground program was suspended before the ISR gained control. By restricting write-access to the flags to either just the ISR or just the foreground program, the possibility of one process altering data already set by another process is eliminated.

The PIC16F874 does have a read-modify-write capability which also can be used to eliminate the problem of multiple access to the same memory location. That capability was ignored in this project.

A. Program Listing

```

LOC OBJECT CODE      LINE SOURCE TEXT
VALUE

00001 ; Filename:      intexamp.asm
00002 ; Author:        Charles B. Cameron, CDR, USN
00003 ; Date:          5 February 2002
00004 ; Description:   A program demonstrating how to use the timer
00005 ;                interrupt so that a foreground program can achieve
00006 ;                accurate timing without burdening the interrupt
00007 ;                routine.
00008 ;
00009 ;*****
00010 ;
00011 ;   Files required:
00012 ;
00013 ;               p16f874.inc
00014 ;
00015 ;*****
00016 ;
00017 ;   Notes:
00018 ;   This program generates an interrupt every 1 ms.
00019 ;   It's assumed the processor is driven with a 4.000 MHz crystal
00020 ;   oscillator and is to operate in HS mode.
00021 ;
00022 ;   The interrupt service routine (ISR) counts 500 interrupts (1 s)
00023 ;   and then sets a semaphore. The foreground routine monitors the
00024 ;   semaphore and, when it sees it, toggles bit 0 of Port A. It
00025 ;   then resets a different semaphore to inform the ISR that it got
00026 ;   the message. At this point, the ISR resets its semaphore. Then
00027 ;   the foreground program can reset its semaphore and the cycle
00028 ;   continues indefinitely.
00029 ;
00030 ;*****
00031
00032
00033     list      p=16f874          ; list directive to define processor
00034     #include <p16f874.inc>      ; processor specific variable definitions
00001     LIST
00002 ; P16F874.INC Standard Header File, Version 1.00   Microchip Technology, Inc.
00034     LIST
00035
2007  3F32 00036     __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_ENABLE_ON & _LVP_OFF & _CP
D_OFF
00037
00038 ; Turns code protection off, watch dog timer off,
00039 ; brown-out reset disabled, power-up timer enabled, HS oscillator mode selected,
00040 ; flash program memory write disabled, low-voltage in-circuit serial programming
00041 ; disabled, and data EE memory code protection off.
00042
00043 ;***** VARIABLE DEFINITIONS *****
0000020 00044 w_temp      EQU      0x20          ; variable used for context saving
00000021 00045 status_temp   EQU      0x21          ; variable used for context saving
00046
00000022 00047 InterprocessFlags equ 0x22          ; flags for interprocess communication
00000000 00048 TimerInt      equ      0            ; Bit 0 = 1 when the ISR has counted enough interrupts.
00049 ;                = 0 otherwise.

```

```

LOC OBJECT CODE      LINE SOURCE TEXT
VALUE
00000001            00050 TimerAck      equ    1          ; Bit 1 = 1 when the foreground process acknowledges
                                00051                      ; the notification.
                                00052                      ; = 0 otherwise.
00000000            00053 Light        equ    0          ; Points to bit 0 of Port A. Can be used to
                                00054                      ; operate a light
00000001            00055 LightMask     equ    B'00000001' ; A mask with a 1 in the Light bit's position.
                                00056
00000023            00057 PulseCount0    equ    0x23       ; LSB of count of the number of interrupts which have occurred.
00000024            00058 PulseCount1    equ    0x24       ; MSB of count of the number of interrupts which have occurred.
000000F3            00059 PulseCount0Init equ    D'500'-(D'256'+D'1') ; 500 x 1 ms = 500 ms = 1/2 of a 1-Hz cycle.
00000001            00060 PulseCount1Init equ    1          ; = 1 x 256^1 + (500-(256+1) x 256^0 = 500-1 = 499
                                00061
                                00062
                                00063 ;*****
0000                00064                ORG    0x000          ; processor reset vector
0000 018A            00065                clrf   PCLATH        ; ensure page bits are cleared
0001 281D            00066                goto  main          ; go to beginning of program
                                00067
                                00068 ;*****
                                00069 ; The following instructions are a standard prolog for handling all interrupts.
                                00070 ;*****
0004                00071                ORG    0x004          ; interrupt vector location
0004 00A0            00072                movwf  w_temp        ; save off current W register contents
0005 0E03            00073                swapf  STATUS,W      ; move status register into W register
0006 0183            00074                clrf   STATUS        ; select Bank 0
0007 00A1            00075                movwf  status_temp   ; save off contents of STATUS register
                                00076
                                00077 ;*****
                                00078 ; Handle the individual interrupts
                                00079 ;*****
0008 188C            00080                btfsc  PIR1,TMR2IF   ; If this is a Timer 2 interrupt
0009 200F            00081                call   Timer2        ; then service it.
                                00082
                                00083 ;*****
                                00084 ; The following instructions are a standard epilog for handling all interrupts.
                                00085 ;*****
000A 0E21            00086                swapf  status_temp,w ; retrieve copy of STATUS register
000B 0083            00087                movwf  STATUS        ; restore pre-isr STATUS register contents
000C 0EA0            00088                swapf  w_temp,f      ; restore pre-isr W register contents
000D 0E20            00089                swapf  w_temp,w      ; restore pre-isr W register contents
000E 0009            00090                retfie              ; return from interrupt
                                00091
                                00092 ;*****
                                00093 ; Timer 2 Interrupt handler.
                                00094 ;*****
000F                00095 Timer2
                                00096 ; Are we waiting for the foreground process to acknowledge our signal?
000F 1822            00097                btfsc  InterprocessFlags,TimerInt
0010 2819            00098                goto  Timer2CheckAck
0011                00099 Timer2DecrementCount
                                00100 ; No, so decrement the count. If we reach the end, let the foreground routine know.
0011 204A            00101                call   CountTimer2Int
0012 1D03            00102                btfss  STATUS,Z

```

```

LOC OBJECT CODE      LINE SOURCE TEXT
VALUE
0013  2817           00103      goto   Timer2EndISR    ; We haven't reached zero so return.
                                00104 ; We have reached the end.
                                00105 ; Is the acknowledgement flag outstanding?
0014  18A2           00106      btfsc  InterprocessFlags,TimerAck
0015  2817           00107      goto   Timer2EndISR    ; Yes. Wait for it to be withdrawn.
                                00108 ; No, so let the foreground process know the desired
                                00109 ; number of interrupts has been counted.
0016  1422           00110      bsf    InterprocessFlags,TimerInt
                                00111 ; The ISR must reset the interrupt flag before returning.
0017           00112 Timer2EndISR
0017  108C           00113      BCF   PIR1,TMR2IF ; Clear Timer 2 interrupt flag and continue.
0018  0008           00114      return
0019           00115 Timer2CheckAck
                                00116 ; Has the foreground process acknowledged our signal?
0019  1CA2           00117      btfss  InterprocessFlags,TimerAck
001A  2811           00118      goto   Timer2DecrementCount ; No. Update count and continue.
                                00119
001B           00120 Timer2AckReceived ; Yes. Remove the notification semaphore.
001B  1022           00121      bcf    InterprocessFlags,TimerInt
001C  2811           00122      goto   Timer2DecrementCount
001D           00123 main
                                00124
                                00125 ; remaining code goes here
                                00126
                                00127 ; Configure Port A. Let all bits be outputs. We'll use Bit 0 to output a test signal.
001D  1283           00128      bcf    STATUS,RP0     ; Select Bank 0
001E  1303           00129      bcf    STATUS,RP1
001F  0185           00130      clrf   PORTA         ; Initialize Port A by clearing the output latches.
0020  1683           00131      bsf    STATUS,RP0     ; Select Bank 1
0021  3006           00132      movlw  B'00000110'   ; Let all pins be for digital use, not analog use.
Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
0022  009F           00133      movwf  ADCON1
0023  3000           00134      movlw  B'00000000'   ; Let all Port A pins be used for output.
Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
0024  0085           00135      movwf  TRISA
0025  1283           00136      bcf    STATUS,RP0     ; Revert to Bank 0
                                00137
                                00138 ; Set up Timer 2 to generate interrupts every 1 ms. Since we're assuming an instruction
                                00139 ; cycle consumes 1 us, we need to cause an interrupt every 1000 instruction cycles.
                                00140 ; We'll set the prescaler to 4, the PR2 register to 250, and the postscaler to 1. This
                                00141 ; will generate interrupts every 4 x 250 x 1 = 1000 instruction cycles.
                                00142
                                00143 ; *****
                                00144 ; START OF CODE to initialize Timer 2
                                00145 ; *****
0026  018B           00146      CLRF  INTCON         ; Disable interrupts
0027  0191           00147      CLRF  TMR2          ; Clear Timer2 register
0028  1683           00148      BSF   STATUS, RP0    ; Bank1
0029  170B           00149      bsf   INTCON,PEIE    ; Enable peripheral interrupts
Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
002A  018C           00150      CLRF  PIE1          ; Mask all peripheral interrupts except
Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
002B  148C           00151      bsf   PIE1,TMR2IE    ; the timer 2 interrupts.

```

```

LOC OBJECT CODE      LINE SOURCE TEXT
VALUE
002C 1283            00152      BCF      STATUS, RP0      ; Bank0
002D 018C            00153      CLRWF   PIR1             ; Clear peripheral interrupts Flags
002E 3001            00154      movlw   B'00000001'     ; Set Postscale = 1, Prescale = 4, Timer 2 = off.
002F 0092            00155      movwf   T2CON
0030 1683            00156      BSF     STATUS, RP0     ; Bank1
0031 30F9            00157      movlw   D'250'-1       ; Set the PR2 register for Timer 2 to divide by 250.
Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
0032 0092            00158      movwf   PR2
0033 1283            00159      BCF     STATUS, RP0     ; Bank0
0034 178B            00160      bsf     INTCON,GIE      ; Global interrupt enable.
0035 1512            00161      BSF     T2CON,TMR2ON    ; Timer2 starts to increment
00162
00163 ; *****
00164 ; END OF CODE to initialize Timer 2
00165 ; *****
00166 ; Initialize interprocess communications flags
0036 01A2            00167      clrf   InterprocessFlags ; Flags are 0 when not in use.
00168 ; Initialize the interrupt counter so that it will count up.
0037 2045            00169      call  InitPulseCount
00170 ; Turn off Light bit of Port A
0038 1005            00171      bcf    PORTA,Light
00172
00173 ; *****
00174 ;
00175
0039                00176      loop
00177 ; The function of this program is to toggle an output bit PORTA(Light)
00178 ; every 1/2 second. The toggling is trivial. Most of the work here has
00179 ; to do with a proper handshake between the interrupt service routine and
00180 ; this foreground program. The ISR will notify us when 1/2-second's worth
00181 ; of interrupts have been counted. When this happens, it will set a
00182 ; notification flag. Once we've noticed it, we'll set an acknowledgement flag.
00183 ; When the ISR has noticed this, it will withdraw the notification flag,
00184 ; whereupon we withdraw the acknowledgement flag. The ISR has exclusive
00185 ; write-access to the notification flag. The foreground process has exclusive
00186 ; write-access to the acknowledgement flag.
00187
00188 ; Have we already acknowledged a notification?
0039 18A2            00189      btfsc  InterprocessFlags,TimerAck
003A 2841            00190      goto   Acknowledged      ; Yes.
003B 1C22            00191      btfss  InterprocessFlags,TimerInt ; No. Is there a new one?
003C 2839            00192      goto   loop              ; No. Keep waiting.
003D                00193      NextHalfSecond          ; Yes
003D 14A2            00194      bsf    InterprocessFlags,TimerAck ; Acknowledge it.
003E 3001            00195      movlw  LightMask        ; This puts a 1 in the Light bit of the W register
003F 0685            00196      xorwf  PORTA,LightMask  ; Toggle Light bit of Port A.
0040 2839            00197      goto   loop              ; and resume waiting.
00198 ; We've already acknowledged the most recent notification.
00199 ; We're waiting for the notification to be withdrawn by the ISR.
00200 Acknowledged
0041                00201      btfsc  InterprocessFlags,TimerInt ; Is an interrupt notification outstanding?
0042 2839            00202      goto   loop              ; Yes. Wait until it's withdrawn.
0043 10A2            00203      bcf    InterprocessFlags,TimerAck ; Not any more. Withdraw the acknowledgement.

```


SYMBOL TABLE

LABEL	VALUE
ACKDT	00000005
ACKEN	00000004
ACKSTAT	00000006
ADCON0	0000001F
ADCON1	0000009F
ADCS0	00000006
ADCS1	00000007
ADDEN	00000003
ADFM	00000007
ADIE	00000006
ADIF	00000006
ADON	00000000
ADRESH	0000001E
ADRESL	0000009E
Acknowledged	00000041
BCLIE	00000003
BCLIF	00000003
BF	00000000
BRGH	00000002
C	00000000
CCP1CON	00000017
CCP1IE	00000002
CCP1IF	00000002
CCP1M0	00000000
CCP1M1	00000001
CCP1M2	00000002
CCP1M3	00000003
CCP1X	00000005
CCP1Y	00000004
CCP2CON	0000001D
CCP2IE	00000000
CCP2IF	00000000
CCP2M0	00000000
CCP2M1	00000001
CCP2M2	00000002
CCP2M3	00000003
CCP2X	00000005
CCP2Y	00000004
CCPR1H	00000016
CCPR1L	00000015
CCPR2H	0000001C
CCPR2L	0000001B
CHS0	00000003
CHS1	00000004
CHS2	00000005
CKE	00000006
CKP	00000004
CREN	00000004
CSRC	00000007
CountTimer2Int	0000004A
CountTimer2IntBorrow0	00000050
CountTimer2IntBorrow1	00000058
D	00000005

SYMBOL TABLE

LABEL	VALUE
DATA_ADDRESS	00000005
DC	00000001
D_A	00000005
EEADR	0000010D
EEADRH	0000010F
EECON1	0000018C
EECON2	0000018D
EEDATA	0000010C
EEDATH	0000010E
EEIE	00000004
EEIF	00000004
EEPGD	00000007
F	00000001
FERR	00000002
FSR	00000004
GCEN	00000007
GIE	00000007
GO	00000002
GO_DONE	00000002
I2C_DATA	00000005
I2C_READ	00000002
I2C_START	00000003
I2C_STOP	00000004
IBF	00000007
IBOV	00000005
INDF	00000000
INTCON	0000000B
INTE	00000004
INTEDG	00000006
INTF	00000001
IRP	00000007
InitPulseCount	00000045
InterprocessFlags	00000022
Light	00000000
LightMask	00000001
NOT_A	00000005
NOT_ADDRESS	00000005
NOT_BO	00000000
NOT_BOR	00000000
NOT_DONE	00000002
NOT_PD	00000003
NOT_POR	00000001
NOT_RBPU	00000007
NOT_RC8	00000006
NOT_T1SYNC	00000002
NOT_TO	00000004
NOT_TX8	00000006
NOT_W	00000002
NOT_WRITE	00000002
NextHalfSecond	0000003D
OBF	00000006
OERR	00000001
OPTION_REG	00000081

SYMBOL TABLE

LABEL	VALUE
P	00000004
PCFG0	00000000
PCFG1	00000001
PCFG2	00000002
PCFG3	00000003
PCL	00000002
PCLATH	0000000A
PCON	0000008E
PEIE	00000006
PEN	00000002
PIE1	0000008C
PIE2	0000008D
PIR1	0000000C
PIR2	0000000D
PORTA	00000005
PORTB	00000006
PORTC	00000007
PORTD	00000008
PORTE	00000009
PR2	00000092
PS0	00000000
PS1	00000001
PS2	00000002
PSA	00000003
PSPIE	00000007
PSPIF	00000007
PSPMODE	00000004
PulseCount0	00000023
PulseCount0Init	000000F3
PulseCount1	00000024
PulseCount1Init	00000001
R	00000002
RBIE	00000003
RBF	00000000
RC8_9	00000006
RC9	00000006
RCD8	00000000
RCEN	00000003
RCIE	00000005
RCIF	00000005
RCREG	0000001A
RCSTA	00000018
RD	00000000
READ_WRITE	00000002
RP0	00000005
RP1	00000006
RSEN	00000001
RX9	00000006
RX9D	00000000
R_W	00000002
S	00000003
SEN	00000000
SMP	00000007

SYMBOL TABLE

LABEL	VALUE
SPBRG	00000099
SPEN	00000007
SREN	00000005
SSPADD	00000093
SSPBUF	00000013
SSPCON	00000014
SSPCON2	00000091
SSPEN	00000005
SSPIE	00000003
SSPIF	00000003
SSPM0	00000000
SSPM1	00000001
SSPM2	00000002
SSPM3	00000003
SSPOV	00000006
SSPSTAT	00000094
STATUS	00000003
SYNC	00000004
T0CS	00000005
T0IE	00000005
T0IF	00000002
T0SE	00000004
T1CKPS0	00000004
T1CKPS1	00000005
T1CON	00000010
T1INSYNC	00000002
T1OSCEN	00000003
T1SYNC	00000002
T2CKPS0	00000000
T2CKPS1	00000001
T2CON	00000012
TMR0	00000001
TMR1CS	00000001
TMR1H	0000000F
TMR1IE	00000000
TMR1IF	00000000
TMR1L	0000000E
TMR1ON	00000000
TMR2	00000011
TMR2IE	00000001
TMR2IF	00000001
TMR2ON	00000002
TOUTPS0	00000003
TOUTPS1	00000004
TOUTPS2	00000005
TOUTPS3	00000006
TRISA	00000085
TRISB	00000086
TRISC	00000087
TRISD	00000088
TRISE	00000089
TRISE0	00000000
TRISE1	00000001

SYMBOL TABLE

LABEL	VALUE
TRISE2	00000002
TRMT	00000001
TX8_9	00000006
TX9	00000006
TX9D	00000000
TXD8	00000000
TXEN	00000005
TXIE	00000004
TXIF	00000004
TXREG	00000019
TXSTA	00000098
Timer2	0000000F
Timer2AckReceived	0000001B
Timer2CheckAck	00000019
Timer2DecrementCount	00000011
Timer2EndISR	00000017
TimerAck	00000001
TimerInt	00000000
UA	00000001
W	00000000
WCOL	00000007
WR	00000001
WREN	00000002
WRERR	00000003
Z	00000002
_BODEN_OFF	00003FBF
_BODEN_ON	00003FFF
_CPD_OFF	00003FFF
_CPD_ON	00003EFF
_CP_ALL	00000FCF
_CP_HALF	00001FDF
_CP_OFF	00003FFF
_CP_UPPER_256	00002FEF
_DEBUG_OFF	00003FFF
_DEBUG_ON	000037FF
_HS_OSC	00003FFE
_LP_OSC	00003FFC
_LVP_OFF	00003F7F
_LVP_ON	00003FFF
_PWRTE_OFF	00003FFF
_PWRTE_ON	00003FF7
_RC_OSC	00003FFF
_WDT_OFF	00003FFB
_WDT_ON	00003FFF
_WRT_ENABLE_OFF	00003DFF
_WRT_ENABLE_ON	00003FFF
_XT_OSC	00003FFD
_16F874	00000001
loop	00000039
main	0000001D
status_temp	00000021
w_temp	00000020

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```
0000 : XX-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXX-----
2000 : -----X-----
```

All other memory blocks unused.

Program Memory Words Used: 89
Program Memory Words Free: 4007

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 5 reported, 0 suppressed