Chapter 9. Timer Modules and Digital Clock Application

In 16F877, there are three timer modules: Timer0, Timer1, and Timer2 modules. The Timer0 module is a readable/writable 8-bit timer/counter consisting of one 8-bit register, TMR0. It triggers an interrupt when it overflows from FFh to 00h.

The Timer1 module is a readable/ writable 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L). The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The Timer1 Interrupt is generated on overflow.

The Timer2 is an 8-bit timer with a prescaler, a postscaler, and a period register. Using the prescaler and postscaler at their maximum settings, the overflow time is the same as a 16-bit timer. Timer2 is the PWM time-base when the CCP module(s) is used in the PWM mode. Detailed description and application of each timer, except Timer2 module, follow.

1. Timer 0

Timer0 module can work as a timer and a counter, however, in this section of Timer0, we use it as a timer only. In Timer1 module, we use it, instead, as a counter. So, for counter purpose, see the section for Timer1 module.

Timer mode is selected by clearing the T0CS bit (OPTION_REG<5>). In timer mode, the Timer0 module will increment every instruction cycle (without prescaler). Prescaler concept comes from the too-fast instruction cycle of the microcontroller. Think about the Timer0 register, TMR0. If the content is incremented by one every instruction (i.e., 0.2 µs with 20 MHz crystal oscillator), it takes, from 00h to FFh ,only 255x0.2µs=51µs. Then, how many overflow would we need, if we want to have an exact 1 second time delay? It would be over 19500 overflows. A mere 1ms delay would require about 20 overflows. Prescaler then is to give multiple instructions cycles for the increment of TMR0 register. Prescaler value of 1:4 would take 4 instruction cycles to increment TMR0 by 1. On the other hand, prescaler value of 1:256 requires 256 instruction cycles for the increment. With prescaler value of 1:256, one over flow would take 255x256x0.2µs=13056µs. Therefore, with 1:256, it would take only 76 overflows to have an exact 1 second timing. The prescaler is not readable or writable. Instead, The prescaler assignment is controlled in software by the PSA control bit (OPTION_REG<3>). Clearing the PSA bit will assign the prescaler to the Timer0 module.

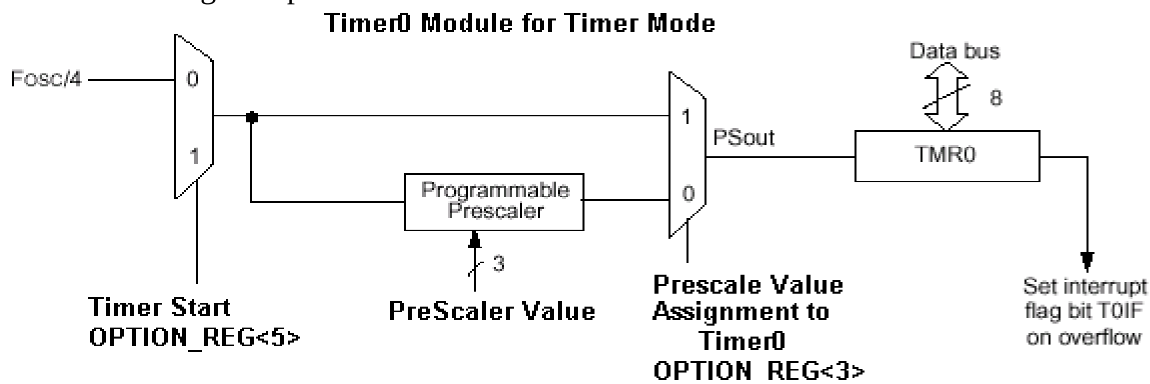


Fig. 70 Timer0 Module for Timer Mode

Timer0 starts or stops by the T0CS bit of OPTION_REG. Once it is started, the incremental signal comes to the TMR0 register based on the value selected for a prescaler. When TMR0 register is overflow, the T0IF flag is set to indicate the overflow. There are two ways to monitor the overflow event of TMR0: polling the T0IF flag and Triggering the Timer0 interrupt. In our example, we explore both the methods.

As you notice, we already talked about one register heavily, OPTION_REG register, while explaining the Timer0 module. The main control action of OPTION_REG register is to assign a prescaler value to Timer0 and start/stop the timer. Clearing T0CS bit starts the timer increment based on the prescaler value, assigned by clearing PSA bit and selected by the PS2:PS0 bits.

OPTION_REG (81h) For Timer Operation

$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

T0CS: TMR0 Clock
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock

PSA: Prescaler Assignment
1 = Prescaler is to WDT
0 = Prescaler is to the Timer0

PS2:PS0: Prescaler Rate Select

TMR0 Rate			
1:2	0	0	0
1:4	0	0	1
1:8	0	1	0
1:16	0	1	1
1:32	1	0	0
1:64	1	0	1
1:128	1	1	0
1:256	1	1	1

The only other file register for the Timer0 module operation is INTCON register. INTCON register allows, in principle, interrupt for all interrupt enabled devices and modules. For the polling method, we may be able to enable the global interrupt by setting the GIE bit, but disable the T0IE bit of Timer0 module interrupt. Therefore, to use the interrupt method for Timer0 application, we have set both the bits: GIE and T0IE. If interrupt method is not used, just clearing GIE bit would do. In polling method, the pin T0IF bit must be monitored for the overflow of TMR0. In interrupt method, this is not necessary. However, for both the method, once a overflow event occurs, the T0IF must be cleared by software, i.e., in the code.

INTCON REGISTER (0Bh, 8Bh, 10Bh, 18Bh) for TIMER0 Operation

GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RB1F

GIE: Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

T0IE: TMR0 Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TNR0 interrupt

T0IF: TMR0 Interrupt Flag bit
1 = TMR0 register has overflowed
(must be cleared in software)
0 = TMR0 register did not overflow

2. Timer 0 Application 1 - LED Blinking

Since we discussed about Timer0 module and necessary special function registers, it is about the time to apply this module. We will discuss two simple example cases of LED On and Off program. In our previous example of LED, we could build a time delay solely based on the number of instruction cycles for a given routine. In this section, we apply Timer0 module for the same purpose. To do this, we apply two different approaches as announced earlier: polling approach and interrupt approach.

Timer0 Application with Polling Approach
The polling approach is to monitor the T0IF bit of INTCON register for an overflow event in TMR0. For a desired delay, we would come up with how many overflows are necessary based upon the prescaler value. Here is a general procedure for the polling approach.

1. Assign the prescaler to Timer0 by clearing PSA bit (OPTION_REG<3>).
2. Select the desired prescale value by the 3 bits of OPTION_REG. (OPTION_REG<2:0>)
3.Clear TMR0 register and clear T0IF bit (INTCON<2>).
4.Turn on the timer by clearing T0CS bit (OPTION_REG<5>).
5.Poll T0IF for the timer overflow. The timer overflows when the value of TMR0 increments from 0xFF to 0x00. This sets T0IF.
6. If T0IF is set, clear it.

Then, how do we get 1 second time delay? As we briefly discussed above, with 0.2μs of one instruction cycle time, we need 76 overflows of TMR0 when 1:256 prescaler value is selected. In the sample program, we will turn on an LED for 1 second while turning off the other LED, and vice versa, using the timer. Let's build the 1 second delay routine. The strategy is to decrease a temporary counting register COUNT from the magic number 76 every time the TMR0 overflow occurs. The subroutine expires when the COUNT reduces to zero, which will turn into one second lapse of time. Before returning to the main program, we have to clear the T0IF bit so that the TMR0 is again incremented by one.

```
;DELAY SUBROUTINE for 1 Second delay
DELAY1s
      banksel     count
      movlw       0x4c              ;Count=76 for 1 second to expire
      movwf       count

over  btfss       INTCON,     T0IF        ;Tmr0 overflow?
      goto        over
      bcf         INTCON, T0IF            ;reset/clear when done
      decfsz      count
      goto        over
      return
```

Two LEDs are connected to RD0 and RD1, respectively.

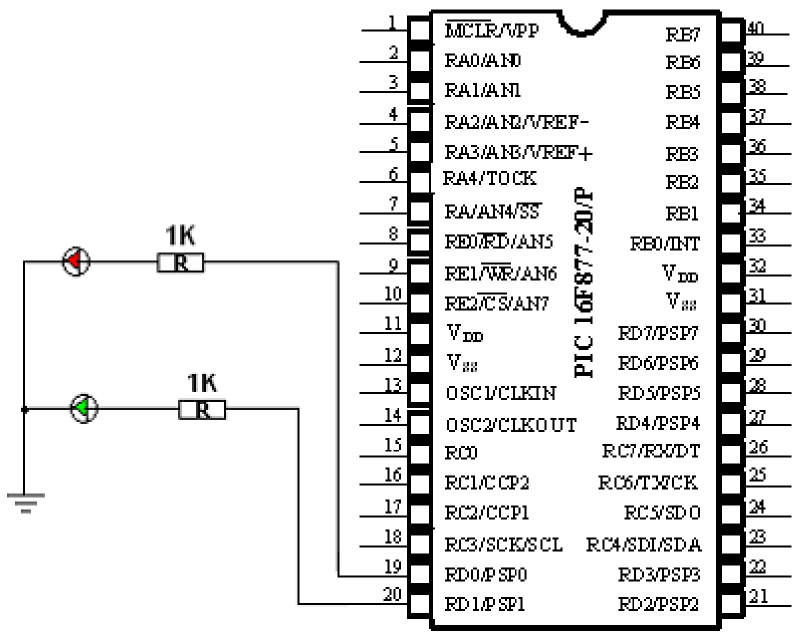


Fig. 71 PIC 16F877 connection to two LEDs

The code listed below is the full program except the 1 second time delay we already discussed.

```
;tmr0poll.asm
;
;This program uses TMR0 module with software polling
;to give exact 1 s delay of LED On and Off
;
        list P = 16F877

STATUS            EQU   0x03
TMR0              EQU   0x01        ;Timer0 register
INTCON            EQU   0x0B        ;
OPTION_REG        EQU   0x81        ;Option Register
```

```
T0IF            EQU   0x02
PORTD           EQU   0x08
TRISD           EQU   0x88
LED1            EQU   0x01        ;LED1 is connected to PORTD<1>
LED0            EQU   0x00        ;and PORTD<0>


            CBLOCK      0x20              ; RAM AREA for USE at address 20h
            count
            ENDC              ;end of ram block
;
;
;==========================================================
      org         0x0000
      goto        START
;=======================================================
      org         0x05
START
      banksel     INTCON
      clrf        INTCON            ;int disabled
      clrf        TMR0
      banksel     TRISD
      clrf        TRISD             ;PORTD<7-0>=outputs
      movlw       0xC7              ;11000111
      banksel     OPTION_REG        ;pre-scaler at 1:256
      movwf       OPTION_REG        ;11000111
      banksel     TMR0              ;Timer0 Starting
      clrf        TMR0              ;TMR0=0

;Determine the time count
monitor
      bsf         PORTD,LED1              ;led on 1 second
      bcf         PORTD,LED0
      call        delay1s                 ;1 second time delay by TMR0
      bcf         PORTD,LED1              ;led off 1 second
      bsf         PORTD,LED0
      call        delay1s
      goto        monitor                 ;Keeping on
;DELAY SUBROUTINE for 1 Second delay
;HERE
;
      END
```

Timer Application with Timer0 Interrupt

The second approach is to use the Timer0 interrupt. Even though we have not discussed much on interrupt, time to time, this subject will pop up, and we will discuss the subject as need basis. The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h. This overflow sets bit T0IF (INTCON<2>). The interrupt can be triggered by setting bit T0IE (INTCON<5>). Bit T0IF must be cleared in software by the Timer0 module interrupt service routine before re-enabling this interrupt.

The Global Interrupt Enable bit, GIE (INTCON<7>), enables (if set) all un-masked interrupts or disables (if cleared) all interrupts. Individual interrupts can be disabled through their corresponding enable bits in the INTCON register. The GIE bit is cleared on reset. The “return from interrupt” instruction, RETFIE, exits the interrupt routine as well as sets the GIE bit, which allows any pending interrupt to execute.

When an interrupt is responded to, the GIE bit is cleared to disable any further interrupt, the return address is pushed into the stack and the PC(Program Counter) is loaded with 0004h. In other words, an interrupt event occurs, the execution of a main program is suspended and the execution starts from the instruction originating at 0004h. Therefore, any routine residing from the 0004h to handle interrupt is usually called an interrupt handler or interrupt service routine. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. Generally the interrupt flag bit(s) must be cleared in software before re-enabling the global interrupt to avoid recursive interrupts.

Interrupt latency is defined as the time from the interrupt event (the interrupt flag bit gets set) to the time that the instruction at address 0004h starts execution (when that interrupt is enabled). For synchronous interrupts (typically internal), the latency is 3 instruction cycles. For asynchronous interrupts (typically external), the interrupt latency will be 3 - 3.75 instruction cycles. The exact latency depends upon when the interrupt event occurs in relation to the instruction cycle. In most application, the interrupt latency does not give much delay. Moreover, we have no control over this. Accept!

So, for Timer0 application, we have to have the interrupt handler residing at 0004h. This handler will decide what we do (or what we want the 16F877 controller to do) when the Timer0 interrupt event occurs by the TMR0 overflow. What we do is, whenever there is interrupt (this case only from the Timer0 module of TMR0 overflow), that we increase the COUNT. That is all. The handler does not care what the current value of COUNT is. The clearing of COUNT and checking the COUNT is the job of 1 second delay subroutine.

```
;Interrupt Handler for Timer0 interrupt
      ORG         0x0004            ;Interrupt Vector address
      incf        COUNT             ;increase COUNT
      bcf         INTCON, T0IF      ;clear the interrupt flag for
                                    ;another interrupt
      retfie                        ;return from Interrupt
```

Since the `COUNT` is accessed by any part of the code, the 1 second time delay subroutine must check the value of `COUNT` starting from 0. When the `COUNT` becomes 76 (or 4Ch), the subroutine expires and the 1 second time delay is achieved. The subroutine does not have to take care of clearing T0IF; it's done by the interrupt handler. When the `COUNT` becomes 76 and the subroutine expires, the `COUNT` must be cleared for another 1 second counting.

```
;subroutine delay1s
;
delay1s
      banksel     COUNT
      btfss       COUNT, 0x06       ;check if COUNT increased to 0x4c
                                    ;01001100 bit 6
      goto        delay1s
```

```
int1s btfss       COUNT, 0x03       ;bit 3
      goto        int1s
int1s2
      btfss       COUNT, 0x02       ;bit 2
      goto        int1s2
                                    ;now 1 sec expired
      clrf        COUNT             ;COUNT=0
      return
```

The example code, without including the subroutine, is listed below.

```
;tmr0int.asm
;
;This program uses TMR0 module with interrupt enabled
;to give exact 1 s delay
;
        list P = 16F877

STATUS            EQU   0x03
TMR0              EQU   0x01        ;Timer0 module
INTCON            EQU   0x0B        ;Intcon
OPTION_REG        EQU   0x81        ;Option Register
PORTD             EQU   0x08
TRISD             EQU   0x88
LED1              EQU   0x01        ;LED is connected to PORTD<1>
LED0              EQU   0x00
T0IF              EQU   0x02        ;tmr0 overflow flag
T0IE              EQU   0x05        ;Tmr0 interrupt enable/disable
ZERO              EQU   0x02        ;Zero flag on STATUS (1: zero)
GIE               EQU   0x07        ;Global Interrupt

            CBLOCK      0x20              ; RAM AREA for USE at address 20h
            count
            ENDC              ;end of ram block
;
;
;==========================================================
      org         0x0000
      goto        START
;=======================================================
;Interrupt Handler
      org         0x0004            ;Interrupt Vector
      incf        COUNT       ;increase COUNT
      bcf         INTCON, T0IF      ;clear the overflow flag
      retfie                  ;return from Interrupt


START clrf        COUNT             ;starting from COUNT=0
      banksel     INTCON
      bsf         INTCON, GIE       ;Global Interrupt Enable
      bsf         INTCON, T0IE      ;tmr0 interrupt enabled
      clrf        TMR0
```

```
      banksel     TRISD
      clrf        TRISD             ;PORTD<7-0>=outputs
      movlw       0xC7              ;pre-scaler at 256
      banksel     OPTION_REG
      movwf       OPTION_REG        ;Timer0 starts


; Is count decreased to 0?  Then 1 second passed.
; timecount is for how many seconds to pass.
;Determine the time count
ONOFF
      banksel     PORTD
      bsf         PORTD, LED1       ;LED1 ON
      bcf         PORTD, LED0
      call        delay1s
      banksel     PORTD
      bcf         PORTD, LED1       ;LED1 off
      bsf         PORTD, LED0
      call        delay1s
      goto        ONOFF             ;repeat

;subroutine delay1s
;--------------------------------------------------
      END
```

After running the program, you may be tempted to apply it to a digital clock. Several versions of digital clock (or just a timer watch) are discussed before the final version, displayed on an LCD module.

3. Timer0 Application 2 –DIGITAL CLOCK

In the application of Timer0 module, we will explore the world of digital clock. First two versions are aimed to display the time on a PC monitor; one (CLOCK1) as a timer watch and the other (CLOCK2) as a digital clock with time setting allowed using a keyboard. The second two versions are displayed on a LCD module; one (CLOCK3) as a timer watch and the other (CLOCK4) as a digital clock with time setting using four buttons. In CLOCK4, another interrupt event, RB0/INT external interrupt, is utilized. All through the version, 1 second time delay is implemented using the polling approach.

CLOCK1-Display on PC monitor
This version of digital clock is a timer watch displayed in the format of HH:MM:SS for Hour, Minute, and Second display. The timer starts from 00:00:00 and ticks as an actual timer watch. Let's discuss the strategy. As in the LED On/Off program, when the COUNT reaches at 76, the Second must be increased by one. Then, the number indicating the current Second, in hex number, must be converted to a 2-digit decimal number. These decimal digits will be displayed occupying the two slots assigned for each time unit.

So we first need a general routine which convert a 1-byte hex number to a 2-digit decimal number. In other words, a single bye hex number, say, 16h which is 22 in decimal must be converted to two 8-byte number in decimal number system.

16h: 0001 0110 ---> 0000 0010 (Upper Byte) and 0000 0010 (Lower Byte)

For Hour, since we can have from 00 to 23, the maximum hex number for the time unit is HH= 17h=0001 0111. If put the upper nibble to `hh1hex` (a variable in the assembly code) and the lower nibble to `hh0hex`, we would have:
HH=00010111 ---->hh1dex=00000001 and hh0hex=00000111

If the bit0 of hh1dex is 1, it corresponds to 16. Therefore, the upper decimal digit would be increased by 1, and the lower decimal digit must be increased by 6.

Then, the hh0dex must be examined with the additional increment of 6. In this example, the new hh0hex becomes 00001101= 0Dh. Then, what would be the maximum value of hh0hex? Since the maximum value hh0hex can get is 00001111=0Fh, it could reach above 20 but not above 30. Therefore, we have to check if hh0hex is greater than 20. In the example it's not above 20. So we check if the value is above 10, then. Since 0Dh is bigger than 9we have to subtract 10 from 0D, while adding the carry to the upper digit, hh1dec. In other words, when hh0hex is bigger than 19we increase hh1dec by two and subtract 20 from hh0hex. The resultant hh0hex becomes hh0dec. If hh0hex is not bigger than 19 but bigger than 9, then we increase hh1dec by 1 and subtract 10 from hh0hex. This hh0hex becomes hh0dec, the lower digit of the decimal number.

OK. Let's do the math again for a hex number to a 2-digit decimal number conversion. This algorithm is the basis for a hex number, increased by the 1 second time delay, to 2-digit decimal number display.

Example 1: HH=13h=19d=0001 0011.

(1) `hh1hex` = 0000 0001 (upper nibble)
(2) `hh0hex` = 0000 0011 (lower nibble)
(3) Since the Bit0 of `hh1hex` is 1 (i.e., 16): increase `hh1dec` by 1 (`hh1dec`=1 now) and increase `hh0hex` by 6. `hh0hex`=0000 1001 now.
(4) Since `hh0hex` is not greater than 9, (it is 9), `hh0hex` becomes `hh1dec`. So `hh1dec` = 9 now.
(5) Finally, the 2 digits of decimal number is: 1 (by `hh1dec`) 9 (by `hh0dec`)
(6) Pint `hh1dec` followed by `hh0dec`, 19, to indicate the 19th hour

Example 2: MM (for Minute) = 3Bh=59d = =0011 1011

(1) `mm1hex` = 0000 0011 (upper nibble)
(2) `mm0hex` = 0000 1011 (lower nibble)
(3) Since Bit 0 of `mm1hex` is 1 (i.e. $16x2^0$=16d), increase `mm1dec` by 1 and `mm0hex` by 6. So, currently, `mm1dec`=1, and the new value of `mm0hex` = 0000 1011 + 0000 0110 = 0001 0001 = 17d
(4) Since Bit 1 of `mm1hex` is 1 (i.e., $16x2^1$=32d) increase `mm1dec` by 3 and `mm0hex` by 2. Therefore, the current value of `mm1dec` = 4 and the new value of `mm0hex` is 19d.
(5) Now checking `mm0hex` indicates that it is smaller than 20 and bigger than 9. So it would increase `mm1dec` by 1 and the resultant `mm0hex` after being

subtracted by 10 is 9. Finally, `mm1dec`=5 and `mm0dec` =9.
(6) Display the two digits, 5 and 9, to indicate the 59th minute.

Example 3: SS (for Second) = 1Fh=0001 1111 = 31d

(a) `ss1hex` = 0000 0001 (upper nibble)
(b) `ss0hex` = 0000 1111 (lower nibble)
(c) The bit0 of `ss1hex` is 1, therefore, $16 \times 2^0 = 16$, increase `ss1dec` by 1 and `ss0hex` by 6. So the current value of `ss1dec` =1 and the new value of `ss0hex` is 15d+ 6d= 21d.
(c) Since `hh0hex` is bigger than 19, increase `ss1dec` by 2 to 3 and subtract 20 from `hh0dex`, which results in 1d as `ss0dec`.
(d) Therefore, the final values for `ss1dec` and `ss0dec` are 3 and 1, respectively.
(e) Display `ss1dec` followed by `ss0dec` to indicate the 31st second.

Since the maximum decimal number is 59, and it's hex equivalent is 3Bh, there is no need to check the 2nd or higher bit of `hh1hex`, `mm1hex`, or `ss1hex`. In other words, all we have to do is the check the 0th and 1st bits of the upper nibble. So the following is the subroutine to convert a 1-byte hex number to a 2 digit decimal number.

```
;===h2d2====
;1 byte hex to 2 digit DECIMAL number
;for SS second (MM minute, or HH hour)
;The hex number is stored in hms before calling this subroutine
h2d2
;convert 1-byte hex number to 2 digit decimal number
      movf        hms,0             ;W<--hms
      andlw       0x0F              ;lower nibble
      movwf       hms0hex           ;hms0hex
      movf        hms,0
      movwf       hmstemp
      swapf       hmstemp,0
      andlw       0x0F              ;upper nibble
      movwf       hms1hex
;
      clrf        hms1dec
      clrf        hms0dec
      btfss       hms1hex,0x01      ;Bit1 check   (32)
      goto        b0check
      incf        hms1dec           ;hms1dec = hms1dec + 3
      incf        hms1dec
      incf        hms1dec           ;
      incf        hms0hex           ;hms0hex = hms0hex +2
      incf        hms0hex
b0check
      btfss       hms1hex,0x00      ;Bit0 check  (16)
      goto        hms0check
      incf        hms1dec           ;hms1dec=hms1dec + 1
      incf        hms0hex
      incf        hms0hex
      incf        hms0hex
```

```
      incf        hms0hex
      incf        hms0hex
      incf        hms0hex          ;hms0hex = hms0hex + 6
hms0check
      bcf         HILO20,0x00      ;index for >19 condition
      movf        hms0hex,0        ;check if it's bigger than 20(d)
      call        TWENTY
      btfss       HILO20,0x00
      goto        hms0check2
      movlw       0x14             ;if >19, subtract 20
      subwf       hms0hex
      movf        hms0hex,0
      movwf       hms0dec          ;then hms1dec=hms1dec+2
      incf        hms1dec
      incf        hms1dec          ;two decimal digits
      return
hms0check2
      bcf         HILO10,0x00      ;if <20, the check if >9
      movf        hms0hex,0        ;then check >10
      call        TEN
      btfss       HILO10,0x00
      goto        less             ;less than <10
      movlw       0x0A
      subwf       hms0hex          ;if >9
      movf        hms0hex,0        ;subtract 10
      movwf       hms0dec
      incf        hms1dec          ;hms1dec=hms1dec+1
      return

less  movf        hms0hex,0        ;if <9 then
      movwf       hms0dec          ;keep it to ss0dec
      return
```

The subroutine for `TEN` (checking if a number is greater than or equal to 10) has been discussed before. The two subroutines, `TEN` and `TWENTY` (checking if a number is greater than or equal to 20), are listed below. For the new subroutine, `TWENTY`, read the comment lines very carefully to understand the strategy.

```
;subroutine to check >=10 or <10 ==================
; >=10  ---> HILO10=1
;<10 --->HILO10=0
;     4 3210
;9    0 1001
;10   0 1010
;11   0 1011
;12   0 1100
;13   0 1101
;14   0 1110
;15   0 1111
;16   1 0000
TEN
      banksel     HILO10
      clrf        HILO10
      movwf       TENtemp
      btfss       TENtemp, 0x04    ;4th bit
```

```
      goto        thirdbit
      bsf         HILO10, 0x00
      return
thirdbit
      btfss       TENtemp, 0x03     ;3rd bit
      return
      btfss       TENtemp, 0x02
      goto        nextbit
      bsf         HILO10,0x00
      return
nextbit
      btfss       TENtemp,0x01
      return
      bsf         HILO10, 0x00
      return
;=========================================
;subroutine to check >=20 or <10 ==================
; >=20  ---> HILO20=1
;<20 --->HILO20 =0
;20d = 0001 0100   b4& b2=1
;21    0001 0101
;22    0001 0110
TWENTY
      banksel     HILO20
      clrf        HILO20
      movwf       Twentytemp
      btfss       Twentytemp, 0x04  ;4th bit
      return
      btfss       Twentytemp, 0x02  ;2nd bit
      return
      bsf         HILO20,0x00
      return
```

Now our discussion must go to increasing the Second, and if Second reaches 60 that value must be changed to 00 while increasing the Minute by 1. Similar measure has to be applied to Minute and to Hour. When Hour becomes 24, then it should clear every time unit so that it restarts from 00:00:00. Therefore, after we call 1 second time delay (which is exactly the same routine we used for the LED On/Off using the polling approach) we increase Second (represented by SS in the code) by one. Then we have to check if SS is 60. 60 in decimal is 3C in hexadecimal and 00111100 in binary.

To make sure the content of SS is exactly 00111100, the easiest way to do so is to apply XOR operation with SS. The result of XOR operation of SS with 00111100 is zero only when the content of SS is 00111100. All other values will produce at least one set bit, thus making the result non-zero. The zero or non-zero result can be checked by the ZERO flag of the STATUS register. The tactic applies to find the content of Minute (represented by MM) for 60. A similar measure can solve for Hour (represented by HH) for 24. Examine closely the following code for the main timer watch program.

```
      call        delay1s           ;1 sec elapsed
      incf        SS
      movf        SS,0
      clrf        STATUS
```

```
      xorlw       B'00111100'         ;if SS=60(d) or 3C or 0011 1100
      btfss       STATUS, ZERO
      goto        again               ;if <60 continue

      clrf        SS                  ;if SS=60, then SS=0
      incf        MM                  ;MM=MM+1
      movf        MM,0
      clrf        STATUS
      xorlw       B'00111100'
      btfss       STATUS,ZERO
      goto        again               ;<60, then continue

      clrf        MM                  ;if MM=60, then MM=0
      incf        HH                  ;HH=HH+1
      movf        HH,0
      clrf        STATUS
                                      ;check 24hour 24d = 00011000
      xorlw       B'00011000'
      btfss       STATUS,ZERO
      goto        again
      clrf        STATUS              ;if HH=24
      call        clear               ;clear all the time units (HH=MM=SS=00)
      goto        again
```

The following example code contains all the necessary components including all the subroutines. A complete listing is necessary this time to show the algorithmic process for the very first step for a digital clock. The code will display the time in HH:MM:SS format starting from 00:00:00 like a timer watch. Read comments very carefully to better understand the code.

```
;clock1.asm
;(timer watch)
;This program uses TMR0 module with interrupt enabled
;to give exact 1 s delay for
;HH:MM:SS format
;Displayed on a PC monitor
;
        list P = 16F877

STATUS            EQU   0x03
CARRY             EQU   0x00
TMR0              EQU   0x01          ;Timer0 module
INTCON            EQU   0x0B          ;Intcon
OPTION_REG        EQU   0x81          ;Option Register
T0IF              EQU   0x02          ;tmr0 overflow flag
T0IE              EQU   0x05          ;Tmr0 interrupt enable/disable
ZERO              EQU   0x02          ;Zero flag on STATUS (1: zero)
GIE               EQU   0x07          ;Global Interrupt

TXSTA             EQU   0x98          ;TX status and control
RCSTA             EQU   0x18          ;RX status and control
SPBRG             EQU   0x19          ;USART TX Register
RCREG             EQU   0x1A          ;USART RX Register
PIR1              EQU   0x0C          ;USART RX/TX buffer status (empty or
full)
```

```
RCIF            EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF            EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE          EQU   0x20        ;TXSTA=00100000 : 8-bit, Async
RXMODE          EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port,
                                  ;enable RX
BAUD            EQU   0x0F        ;0x0F (19200), 0x1F (9600)

      CBLOCK    0x20              ; RAM AREA for USE at address 20h
            ASCIIreg
            count
            HHset
            MMset
            SSset

            Hms         ;general variables for HH, MM, and SS
            hms1hex
            hms0hex
            hms1dec
            hms0dec
            hmstemp

            HH
            HHtemp
            HH1
            HH0
            HH1hex
            HH0hex
            hh1dec
            hh0dec
            MM
            MMtemp
            MM1
            MM0
            mm1hex
            mm0hex
            mm1dec
            mm0dec
            SS
            SStemp
            SS1
            SS0
            ss1hex
            ss0hex
            ss1dec
            ss0dec
            HILO10
            HILO20
            TENtemp
            TWENTYtemp
        ENDC                ;end of ram block
;
;==========================================================
      org         0x0000
      GOTO        START
;=======================================================
      org         0x05
START
```

```
      banksel     COUNT
      clrf        COUNT       ;starting from COUNT=0
      banksel     INTCON
      bcf         INTCON      ; Interrupt Disabled
      clrf        TMR0
      movlw       0xC7        ;pre-scaler at 255
      banksel     OPTION_REG
      movwf       OPTION_REG


      call  Async_mode        ;For display to PC monitor
;
      call  clear             ;clear every file register (HH,MM,SS all 0)

again
      movf        SS,0
      movwf       hms
      call        h2d2        ;conversion of SS into 2 –digit decimal number
      movf        hms1dec,0   ;ss1dec & ss0dec
      movwf       ss1dec
      movf        hms0dec,0
      movwf       ss0dec

      movf        MM,0
      movwf       hms
      call        h2d2        ;conversion of MM to mm1dec & mm0dec
      movf        hms1dec,0
      movwf       mm1dec
      movf        hms0dec,0
      movwf       mm0dec

      movf        HH,0        ;conversion of HH to hh1dec & hh0dec
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       hh1dec
      movf        hms0dec,0
      movwf       hh0dec

      call        clockdisplay      ;display them in HH:MM:SS format

      call        delay1s           ;clock ticking here for 1 sec
      incf        SS                ;increase SS
      movf        SS,0
      clrf        STATUS
      xorlw       B'00111100'       ;if SS=60(d) or 3C or 0011 1100
      btfss       STATUS, ZERO
      goto        again             ;if SS<60 do the conversion and display

      clrf        SS                ;if SS=60, SS=0, and MM=MM+1
      incf        MM
      movf        MM,0
      clrf        STATUS
      xorlw       B'00111100'       ;
      btfss       STATUS,ZERO
      goto        again             ;if MM<0, do the conversion and display
```

```
      clrf        MM                  ;if MM=60, MM=0, and HH=HH+1
      incf        HH
      movf        HH,0
      clrf        STATUS
                        ;check 24hour 24d = 00011000
      xorlw       B'00011000'
      btfss       STATUS,ZERO         ;if HH<23, do the conversion and display
      goto        again
      clrf        STATUS
      call        clear               ;if HH=24, HH=MM=SS=0, start again
      goto        again

;SUBROUTINES
;===h2d2====
;1 byte hex to 2 digit DECIMAL number
; for SS second (MM minute, or HH hour)
h2d2
;convert 1-byte hex number to 2 digit decimal number
      movf        hms,0         ;W<--hms
      andlw       0x0F          ;lower nibble
      movwf       hms0hex             ;hms0hex
      movf        hms,0
      movwf       hmstemp
      swapf       hmstemp,0
      andlw       0x0F          ;upper nibble
      movwf       hms1hex
;
      clrf        hms1dec
      clrf        hms0dec
      btfss       hms1hex,0x01        ;B1 check
      goto        b0check
      incf        hms1dec
      incf        hms1dec
      incf        hms1dec             ;32(d)
      incf        hms0hex
      incf        hms0hex
b0check
      btfss       hms1hex,0x00        ;B0 check
      goto        hms0check
      incf        hms1dec             ;16(d)
      incf        hms0hex
      incf        hms0hex
      incf        hms0hex
      incf        hms0hex
      incf        hms0hex
      incf        hms0hex
hms0check
      bcf         HILO20,0x00
      movf        hms0hex,0           ;check if it's bigger than 20(d)
      call        TWENTY
      btfss       HILO20,0x00
      goto        hms0check2
      movlw       0x14
      subwf       hms0hex
      movf        hms0hex,0
      movwf       hms0dec
      incf        hms1dec
```

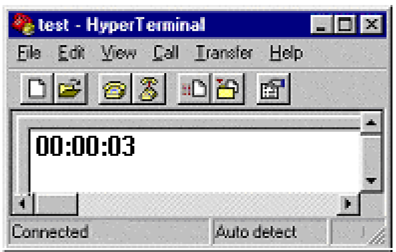
```
      incf        hms1dec                 ;two decimal digits
      bcf         HILO20,0x00
      return
hms0check2
      bcf         HILO10,0x00
      movf        hms0hex,0         ;then check >10
      call        TEN
      btfss       HILO10,0x00
      goto        less              ;less than <10
      movlw       0x0A
      subwf       hms0hex
      movf        hms0hex,0
      movwf       hms0dec
      incf        hms1dec
      return

less  movf        hms0hex,0
      movwf       hms0dec                 ;so keep it to ss0dec
      return

;end of h2d2 subroutine
;
;DELAY SUBROUTINE for 1 Second delay
;
DELAY1s
      banksel     count
      movlw       0x4c                    ;Count=76 for 1 second to expire
      movwf       count

over  btfss       INTCON,     T0IF        ;Tmr0 overflow?
      goto        over
      bcf         INTCON, T0IF            ;reset
      decfsz      count
      goto        over
      return

;---------------------------------------------------
;RX TX Initialization with Asyc Mode
;Async_mode Subroutine
Async_mode
      banksel     SPBRG
      movlw       baud        ;B'00001111' (19200)
      movwf       SPBRG
      banksel     TXSTA
      movlw       TXMODE            ;B'00100000' Async Mode
      movwf       TXSTA
      banksel     RCSTA
      movlw       RXMODE            ;B'10010000' Enable Port
      movwf       RCSTA
      return
;RS232 TX subroutine ============
TXPOLL
      banksel     PIR1
      btfss       PIR1, TXIF  ; Check if TX buffer is empty
      goto        TXPOLL
      banksel     TXREG
      movwf       TXREG       ; Place the character to TX buffer
```

```
      return
;-------------------------
RXPOLL
      banksel     PIR1
      btfss       PIR1, RCIF  ;RX Buffer Full?  (i.e. Data Received?)
      goto        RXPOLL
      banksel     RCREG
      movf        RCREG,0           ;received data  to W
      return
;
;To send CR  ===============
CR
      movlw       H'0d'       ;CR
      call        TXPOLL
      return
;To send CR and LF ===============
CRLF
      movlw       H'0d'       ;CR
      call        TXPOLL
      movlw       H'0a'       ;LF
      call        TXPOLL
      return
;subroutine to check >=10 or <10 ==================
; >=10  ---> HILO10=1
;<10 --->HILO10=0
TEN
      banksel     HILO10
      clrf        HILO10
      movwf       TENtemp
      btfss       TENtemp, 0x04     ;4th bit
      goto        thirdbit
      bsf         HILO10, 0x00
      return
thirdbit
      btfss       TENtemp, 0x03     ;3rd bit
      return
      btfss       TENtemp, 0x02
      goto        nextbit
      bsf         HILO10,0x00
      return
nextbit
      btfss       TENtemp,0x01
      return
      bsf         HILO10, 0x00
      return
;subroutine to check >=20 or <10 ==================
; >=20  ---> HILO20=1
;<20 --->HILO20 =0
;20d = 0001 0100   b4& b2=1
TWENTY
      banksel     HILO20
      clrf        HILO20
      movwf       Twentytemp
      btfss       Twentytemp, 0x04  ;4th bit
      return
      btfss       Twentytemp, 0x02  ;2nd bit
      return
```

```
      bsf         HILO20,0x00
      return
;
;subroutine  CLOCKDISPLAY
clockdisplay
      banksel     hh1dec
      movlw       0x30        ;To all digits add 30h to convert to ASCII
      addwf       hh1dec
      addwf       hh0dec
      addwf       mm1dec
      addwf       mm0dec
      addwf       ss1dec
      addwf       ss0dec

      movf        hh1dec,0
      call        TXPOLL
      movf        hh0dec,0
      call        TXPOLL
      movlw       ':'
      call        TXPOLL            ;:

      movf        mm1dec,0
      call        TXPOLL
      movf        mm0dec,0
      call        TXPOLL
      movlw       ':'
      call        TXPOLL            ;:

      movf        ss1dec,0
      call        TXPOLL
      movf        ss0dec,0
      call        TXPOLL
      call        CR
      return
;========================================
;clock clear-reset subroutine
clear
      clrf        STATUS
      banksel     SS
      movlw       0x00  ;W=0
      clrf        HH
      clrf        MM
      clrf        SS

      clrf        hh1hex
      clrf        hh0hex
      clrf        hh1dec
      clrf        hh0dec

      clrf        mm1hex
      clrf        mm0hex
      clrf        mm1dec
      clrf        mm0dec

      clrf        ss1hex
      clrf        ss0hex
      clrf        ss1dec
```

```
      clrf        ss0dec
      clrf        hms
      return

      END
;END of the code
```

When you run the code, you should see a screen shown below on your monitor.



<u>CLOCK2 - Time Setting with PC Monitor Display</u>
Now let's make the timer watch as an actual digital clock displayed on the same monitor. To do this we have to provide one important feature: Time setting. Allowing a user (or you) to set the time before the clock starts involves more things than one can imagine. First, we have to receive keyed-in numbers for Hour, Minute, and, Second, respectively. Since the numbers entered are in decimal, they should be converted to hexadecimal numbers. These hex numbers are then supplied to the conversion subroutine to convert back to 2-digit decimal numbers for clock display. Why can't we use the keyed-in decimal numbers directly to display the time? Why do we have to reconvert the converted hex number from a decimal number to a decimal number for clock display?

Think about the following situation. For simplicity of argument, consider only the time unit of Second. In other words, only Second is allowed to be adjusted by a user. If you type 45 using your keyboard for Second as the starting time for your digital clock. Each digit could become the first and second digit for Second: `ss1dec` and `ss0dec` as used in the above timer watch program. Then, clock starts from there. So the next clock display after 1 second time delay, hopefully, would be 00:00:46.

However this wishful thinking does not work. It's because after 1 second time delay, SS (the representative variable for Second) would be increased by 1. However, the SS does not contain the would-be starting value of 45, since we directly have the `ss1dec` and `ss0dec` from the number 45. So, we have to convert to SS from `ss1dec` and `ss0dec` for the starting value. That's why we plan to convert the keyed-in decimal numbers to an 8-byte hex number (say, SS, in this case). Conversion from SS to `ss1dec` and `ss0dec` is already covered by using the `h2d2` subroutine.
Therefore the additional parts we have to have to the previous code of timer watch are as

follows:
a. Reading keyed-in decimal number for Hour, Minute, and Second.
b. Conversion of the keyed-in decimal numbers to 1-byte hex numbers (to `HH`, `MM`, and `SS`)
c. Starting the clock using them as starting values.

We need a detailed discussion on the first two parts. The format we want to use for time setting is that we type HH: as a prompt for a user to set the Hour. At the next line, we prompt MM: for the Minute. And at the third line would prompt SS: for the Second. Then at the fourth line, the clock with the set values would start.
Reading the keyed-in decimal numbers is rather an easy task. The serial reception we once studied can be easily applied to receive any keyed-in characters. The following is the subroutine for keyed-in reading for time setting, `timeset`. It does not involve much complexity.

```
;subroutine
;time set prompt and reception
timeset
      call        CRLF        ;move to the next line as the starter
      movlw       'H'
      call        TXPOLL
      movlw       'H'
      call        TXPOLL
      movlw       ':'
      call        TXPOLL      ;HH: as typed

      call        RXPOLL      ;read the first digit, hh1dex
      call        TXPOLL      ;echo the keyed-in number
                              ;subwf   f - W --->d
      movwf       hh1hex
      movlw       0x30
      subwf       hh1hex      ;convert from ASCII to hex number

      call        RXPOLL      ;read the second digit, hh0hex
      call        TXPOLL      ;echo
      movwf       hh0hex
      movlw       0x30        ;hh0hex=hh0hex-30h
      subwf       hh0hex      ;conversion to hex from ASCII

      call        CRLF        ;move to the next line

      movlw       'M'
      call        TXPOLL
      movlw       'M'
      call        TXPOLL
      movlw       ':'
      call        TXPOLL      ;MM: prompted
      call        RXPOLL      ;read the first digit mm1hex
      call        TXPOLL      ;echo
      movwf       mm1hex
      movlw       0x30
      subwf       mm1hex      ;ASCII to HEX

      call        RXPOLL      ;read the second digit, mm0hex
      call        TXPOLL      ;echo
      movwf       mm0hex
```

```
      movlw       0x30        ;ASCII --> HEX
      subwf       mm0hex

      call        CRLF        ;move to the next line

      movlw       'S'
      call        TXPOLL
      movlw       'S'
      call        TXPOLL
      movlw       ':'
      call        TXPOLL      ;SS: prompted
      call        RXPOLL      ;ss1hex
      call        TXPOLL      ;echo
      movwf       ss1hex
      movlw       0x30
      subwf       ss1hex      ;To HEX from ASCII

      call        RXPOLL      ;ss0hex
      call        TXPOLL      ;echo
      movwf       ss0hex
      movlw       0x30
      subwf       ss0hex

      call        CRLF        ;move to the next line
      return

;RS232 TX and RX subroutines ============
TXPOLL
      banksel     PIR1
      btfss       PIR1, TXIF  ; Check if TX buffer is empty
      goto        TXPOLL
      banksel     TXREG
      movwf       TXREG       ; Place the character to TX buffer
      return
;------------------------
RXPOLL
      banksel     PIR1
      btfss       PIR1, RCIF  ;RX Buffer Full?  (i.e. Data Received?)
      goto        RXPOLL
      banksel     RCREG
      movf        RCREG,0           ;received data  to W
      return
```

The next thing we will discuss is the conversion of the keyed-in decimal numbers to 1-byte hex numbers (to `HH`, `MM`, and `SS`). The objective of the discussion is how to convert the 2-digit decimal numbers, for example `hh1hex` and `hh0hex`, to the 1-byte hex number `HH`.

Let's start with an example for `HH` (and `hh1hex` and `hh0hex`). Since the maximum number we get from the upper (or 10) digit `hh1hex` is 2, i.e., 0000 0010, therefore 0000 0010 should be interpreted as 20d (or 14h) while 0000 0011 as 10d (or 0Ah). The sum of this interpreted number and the lower (or unit) digit `hh0hex` would make `HH`, the hex number equivalent.

We can get a general interpretation rule of the upper digit as follows: $\sum_{n=0}^{7} k_n \cdot 2^n \cdot 10$, where k_n is the binary value of the n^{th} bit of the digit. Of course, since we are dealing with a digital clock,

For `MM` (and `mm1hex` and `mm0hex`) and `SS` (and `ss1hex` and `ss0hex`), since the maximum number for the upper digit `mm1hex` (or `ss1hex`) is 5, i.e., 0000 0101, the number *n* goes only to 2 from the formula.

By the way, a number 0000 0101, using the formula above, is interpreted to:

$$\sum_{n=0}^{2} k_n \cdot 2^n \cdot 10 = 1 \cdot 2^0 \cdot 10 + 0 \cdot 2^1 \cdot 10 + 1 \cdot 2^2 \cdot 10 = 50$$

Then, how do we apply this formula for upper digit in the 17F877 coding? Directly applying the formula to a code is too luxurious to the microcontroller. However, we can indirectly apply the formula by testing k_n, the n^{th} bit of the digit and multiplying by ($10 x 2^n$). The following subroutine, `d22h`, is to apply the formula to convert a 2-digit decimal number into a 1-byte hex number. After examining the subroutine, try to make the subroutine simpler by making a part of the code as another subroutine, and apply the same procedure to Hour, Minute, and Second processing.

```
;subroutine
;conversion of decimal two digits to 1-byte hex number
d22h
;HOUR FIRST
      movlw       0x00
      btfss       hh1hex,0x01       ;bit1 check for HOUR
      goto        hnext1
      addlw       0x14              ;if bit1=1, +20
hnext1
      btfss       hh1hex,0x00       ;bit0 check
      goto        hnext2
      addlw       0x0A              ;if bit0=1, +10
hnext2
      movwf       HH
      movf        hh0hex,0          ;+hh0hex the lower digit
      addwf       HH                ;total sum
;end of HH calculation

;MINUTE NEXT
      movlw       0x00
      btfss       mm1hex,0x00       ;bit0 check MINUTE
      goto        mnext1
      addlw       0x0A              ;+10
mnext1
      btfss       mm1hex,0x01       ;bit1 check
      goto        mnext2
      addlw       0x14              ;+20
mnext2
      btfss       mm1hex, 0x02      ;bit2 check
      goto        mnext3
      addlw       0x28              ;+40
mnext3
      movwf       MM
      movf        mm0hex, 0
```

```
      addwf       MM                  ;total sum in hex

;For SECOND
      movlw       0x00
      btfss       ss1hex,0x00         ;bit0 check for SECOND
      goto        snext1
      addlw       0x0A                ;+10
snext1
      btfss       ss1hex,0x01         ;bit1 check
      goto        snext2
      addlw       0x14                ;+20
snext2
      btfss       ss1hex, 0x02        ;bit2 check
      goto        snext3
      addlw       0x28                ;+40
snext3
      movwf       SS
      movf        ss0hex, 0
      addwf       SS                  ;total sum in hex

      return
```

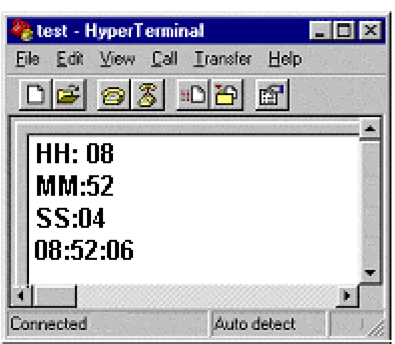
The following code is the main part of the CLOCK2 program. No subroutine is listed. Also, the block of variables (registers) defined from the address 20h is also omitted. The `CBLOCK...ENDC` part is the same as the one we used in CLOCK1 program.

```
; clock2.asm
;
;Clock program
;Time setting allowed
;Display format of HH:MM:SS
;Displayed on a PC monitor
;
        list P = 16F877

STATUS            EQU   0x03
CARRY             EQU   0x00
TMR0              EQU   0x01          ;Timer0 module
INTCON            EQU   0x0B          ;Intcon
OPTION_REG        EQU   0x81          ;Option Register
T0IF              EQU   0x02          ;tmr0 overflow flag
T0IE              EQU   0x05          ;Tmr0 interrupt enable/disable
ZERO              EQU   0x02          ;Zero flag on STATUS (1: zero)
GIE               EQU   0x07          ;Global Interrupt

TXSTA             EQU   0x98          ;TX status and control
RCSTA             EQU   0x18          ;RX status and control
SPBRG             EQU   0x99          ;Baud Rate assignment
TXREG             EQU   0x19          ;USART TX Register
RCREG             EQU   0x1A          ;USART RX Register
PIR1              EQU   0x0C          ;USART RX/TX buffer status (empty or
full)
RCIF              EQU   0x05          ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF              EQU   0x04          ;PIR1<4>: TX Buffer 1-empty 0-full
```

```
TXMODE          EQU   0x20       ;TXSTA=00100000 : 8-bit, Async
RXMODE          EQU   0x90       ;RCSTA=10010000 : 8-bit, enable port,
                                 ; enable RX
BAUD            EQU   0x0F       ;0x0F (19200), 0x1F (9600)

      CBLOCK    0x20             ; RAM AREA for USE at address 20h
;NOTE THAT THIS PORTION MUST BE COPIED FROM CLOCK1.ASM CODE
;FOR A SUCCESSFUL  COMPILING
        ENDC                ;end of ram block
;
;
;
;==========================================================
      org       0x0000
      GOTO      START
      org       0x05
;=======================================================

START
      banksel   INTCON
      clrf      INTCON           ;int disabled
      clrf      TMR0
      banksel   OPTION_REG       ;pre-scaler at 256
      movwf     OPTION_REG       ;11000111
      banksel   TMR0
      clrf      TMR0

      call      Async_mode             ;RX-232
;
      call      clear                  ;clear every file register
begin
;display clock reset prompt
      call      timeset          ;time adjustment
;
;conversion of decimal two digits to 1-byte hex number
      call      d22h
;
again
      movf      SS,0
      movwf     hms
      call      h2d2
      movf      hms1dec,0
      movwf     ss1dec
      movf      hms0dec,0
      movwf     ss0dec

      movf      MM,0
      movwf     hms
      call      h2d2
      movf      hms1dec,0
      movwf     mm1dec
      movf      hms0dec,0
      movwf     mm0dec

      movf      HH,0
      movwf     hms
      call      h2d2
```

```
        movf        hms1dec,0
        movwf       hh1dec
        movf        hms0dec,0
        movwf       hh0dec

        call        clockdisplay
        call        delay1s
        incf        SS
        movf        SS,0
        clrf        STATUS
        xorlw       B'00111100' ;if SS=60(d) or 3C or 0011 1100
        btfss       STATUS, ZERO
        goto        again

        clrf        SS
        incf        MM
        movf        MM,0
        clrf        STATUS
        xorlw       B'00111100'
        btfss       STATUS,ZERO
        goto        again

        clrf        MM
        incf        HH
        movf        HH,0
        clrf        STATUS
                          ;check 24hour 24d = 00011000
        xorlw       B'00011000'
        btfss       STATUS,ZERO
        goto        again
        clrf        STATUS
        call        clear
        goto        again

;SUBROUTINES HERE
;
        END
```

When we run the CLOCK2 program, after setting the time, for example, HH=08, MM=52, SS=04, we would see the following screen on the monitor.



<u>CLOCK3 - LCD Display Version</u>
The next version is closer to a digital clock, or rather a timer watch displayed on a LCD module.

We use the 20x4 LCD module we already used for the previous example programming. For this timer watch example, we will stick to 4-bit interface configuration. If you lost most of the gains on LCD, go back to the proper section and code for better understand this section.

The final result of CLCOK3 on LCD is to display HH:MM:SS format display without time setting features. Therefore, it would start from 00:00:00 at the second line of the LCD screen. The first line of the LCD would display 'PIC CLOCK' as a logo.

Since we already have necessary subroutines, the primary task is to send the calculated digits of time units to LCD not to the PC monitor. Therefore, we have to change the subroutine `clockdisplay` which is for PC monitor to `clockLCDdisplay` for LCD. Basically this change comprises most of the changes we need for displaying on LCD. All the other subroutines are the same as we used from CLOCK1 and CLOCK2. Remember the two subroutines we developed for LCD: instruction write for 4-bit interface (`instw4`) and data write for 4-bit interface (`dataw4`).

```
;subroutin CLOCKLCDDISPLAY
clockLCDdisplay
      banksel     hh1dec
      movlw       0x30
      addwf       hh1dec            ;ASCII conversion
      addwf       hh0dec
      addwf       mm1dec
      addwf       mm0dec
      addwf       ss1dec
      addwf       ss0dec
      movf        hh1dec,0
      call        dataw4            ;hh1dec write to LCD
      movf        hh0dec,0
      call        dataw4            ;hh0dec write to LCD
      movlw       ':'
      call        dataw4            ;: follows
      movf        mm1dec,0
      call        dataw4
      movf        mm0dec, 0
      call        dataw4
      movlw       ':'
      call        dataw4
      movf        ss1dec,0
      call        dataw4
      movf        ss0dec,0
      call        dataw4
      return
```

The example code listed below comes with only main part: subroutines are omitted since we already discussed them before. As before, the `CBLOCK..ENDC` part is also omitted since it is the same block we used for CLOCK1.

```
;clock3.asm
;
;DIGITAL CLOCK ON LCD
; NO BUTTONS FOR TIME SETTING
```

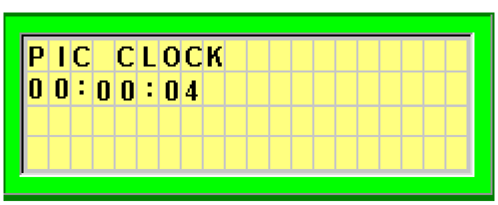
```
;20x4 LCD module
;by Truly (HD44780 compatible)
;
; 4-bit interfacing
;
; Pin Connection from LCD to 16F877
; LCD (pin#)      16F877 (pin#)
;DB7 (14) -----RB7(40)
;DB6 (13) ----RB6(39)
;DB5 (12) ----RB5(38)
;DB4 (11) ----RB4(37)
;DB3 (10)
;DB2 (9)
;DB1 (8)
;DB0 (7)
;E (6) ------RB2(35)
;RW (5) -----RB3(36)
;RS (4) -----RB1(24)
;Vo (3) -----GND
;Vdd (2) ----+5V
;Vss (1) -----GND
;
;Example clcok display:
;     PIC CLOCK  (1st line)
;     HH:MM:SS   (2nd line)
;

      list P = 16F877

STATUS      EQU   0x03
PORTB       EQU   0x06
TRISB       EQU   0x86
RS          EQU   0x01  ;RB1
E           EQU   0x02  ;RB2
RW          EQU   0x03  ;RB3
CARRY       EQU   0x00
TMR0        EQU   0x01        ;Timer0 module
INTCON      EQU   0x0B        ;Intcon
OPTION_REG  EQU   0x81        ;Option Register
T0IF        EQU   0x02        ;tmr0 overflow flag
T0IE        EQU   0x05        ;Tmr0 interrupt enable/disable
ZERO        EQU   0x02        ;Zero flag on STATUS (1: zero)
GIE         EQU   0x07        ;Global Interrupt
;RAM

      CBLOCK      0x20
;NOTE INCLUDE THE VARIABLES (FILE REGISTERS) HERE
;
      ENDC

;program should start from 0005h
;0004h is allocated to interrupt handler

      org         0x0000
      goto        START
```

```
      org         0x05
Start
      BANKSEL     TRISB
; 1 for input, 0 for output

      movlw       0x00
      movwf       TRISB             ;All output

;LCD routine starts
      call        delay10ms
      call        delay10ms         ;LCD warm-up

      banksel     PORTB
      bcf         PORTB, RW         ;RW set LOW here
                                    ;give LCD module to reset automatically

;For TMR0
      banksel     INTCON
      clrf        INTCON            ;int disabled
      clrf        TMR0
      movlw       0xC7
      banksel     OPTION_REG        ;pre-scaler at 256
      movwf       OPTION_REG        ;11000111
      banksel     TMR0
      clrf        TMR0

;END FOR TMR0

;4-BIT INTERFACING
;
;Function for 4-bit (only one write must be done)
;In other words, send only the high nibble
;IMPORTANT
      movlw       0x28
      call        hnibble4
;Function for 4-bit, 2-line display, and 5x8 dot matrix
      movlw       0x28
      call        instw4
;Display On, CUrsor On, No blinking
      movlw       0x0E        ;0F would blink
      call        instw4
;DDRAM address increment by one & cursor shift to right
      movlw       0x06
      call        instw4
;DISPLAY CLEAR

      movlw       0x01
      call        instw4

;Set DDRAM ADDRES
      movlw       0x80        ;00
      call        instw4
;WRITE DATA in the 1st position of line 1
      movlw       'P'         ;P
      call        dataw4
      movlw       'I'         ;I
      call        dataw4
```

```
      movlw       'C'         ;C
      call        dataw4
      movlw       ' '         ;space
      call        dataw4
      movlw       'C'
      call        dataw4
      movlw       'L'
      call        dataw4
      movlw       'O'
      call        dataw4
      movlw       'C'
      call        dataw4
      movlw       'K'
      call        dataw4
;
      call        clear       ;HH=MM=SS=0
                              ;hh1dec=hh0dec=0
                              ;mm1dec=mm0dec=0
                              ;ss1dec=ss0dec=0
AGAIN
;CLOCK DISPLAY
;Set DDRAM address for the 1st position of line 2 (40h)

      movlw       0xC0              ;B'11000000'
      call        instw4            ;RS=0

;CLOCK DISPLAY PART
;Conversion of a hex to a 2-digit decimal number

      movf        SS,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       ss1dec
      movf        hms0dec,0
      movwf       ss0dec

      movf        MM,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       mm1dec
      movf        hms0dec,0
      movwf       mm0dec

      movf        HH,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       hh1dec
      movf        hms0dec,0
      movwf       hh0dec

;Displaying them on LCD
      call        clockLCDdisplay
;1 sec delay
      call        delay1s
```

```
        incf        SS
        movf        SS,0
        clrf        STATUS
        xorlw       B'00111100' ;if SS=60(d) or 3C or 0011 1100
        btfss       STATUS, ZERO
        goto        again

        clrf        SS
        incf        MM
        movf        MM,0
        clrf        STATUS
        xorlw       B'00111100'
        btfss       STATUS,ZERO
        goto        again

        clrf        MM
        incf        HH
        movf        HH,0
        clrf        STATUS
                          ;check 24hour 24d = 00011000
        xorlw       B'00011000'
        btfss       STATUS,ZERO
        goto        again
        clrf        STATUS
        call        clear
        goto        again


;====SUBROUTINES =====
;HERE
;====================
        END
```

If we compile the full code of CLOCK3 and run it, then we would see the following display.



CLOCK4 - LCD Display with Time Setting

This is the eventual version of our digital clock. We display the time on the LCD and provide the feature of time setting. For the time setting feature, we have four buttons: TIME button for the time setting session, HOUR button for Hour setting, MIN button for Minute setting, and CLOCK button to start the clock. The TIME button would stop the clocking procedure and accepts the HOUR and MIN keys to set the time. Since we cannot always wait for the TIME button pressed, we would better have some type of interruption feature of 16F877.

As discussed early in this chapter, interrupt is a useful feature that allows the main program can proceed without keeping eye on the event. Since the button triggered signal comes from outside (external) of 16F877, we consider the RB0/INT interrupt. As the name implies, the RB0 pin (PORTB<0>) has a dual use: regular I/O pin as RB0 and external interrupt (INT) source. This interrupt can be enabled by setting the INT enable bit INTE (INTCON<4>).

External interrupt on the RB0/INT pin is edge triggered, either rising, if INTEDG bit (OPTION_REG<6>) is set, or falling, if the INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, flag bit INTF (INTCON<1>) is set. Flag bit INTF must be cleared in software (i.e., in the code) in the interrupt service routine before re-enabling this interrupt.

The interrupt handler then should do a lot of work: (i) reading the HOUR and MIN buttons, (ii) increasing the corresponding hex numbers for Hour and Minute, and (iii) reading CLOCK button to expire the interrupt handler.

The main routine is not much different from CLOCK3: it displays the contents of `HH`, `MM`, and `SS` (after hex to decimal conversion) no matter what the contents are. The only change includes the necessary accommodation for PORTB for buttons and one LED attached at PORTD for indication purpose. This LED will be turned on as far as the interrupt handler is being processed. The CLOCK button would turn off the LED and clock starts. The circuit diagram for CLOCK4 is illustrated below. The TIME button is connected to RB0/INT pin, and HOUR, MIN, and CLOCK buttons are connected to RD5, RD4, and RD2, respectively. The outputs from the buttons, when not pressed, are High, and when pressed, the outputs experience a High-to-Low transition. Therefore, the proper set-up for INTEDG is 'clear'.

Let's now discuss about the interrupt handler. As discussed, when the TIME button is pressed the RB0/INT pin experiences the High-to-Low transition and this triggers the INT interrupt. Then the Program Counter (PC) is changed to 0004h where the interrupt handler is residing. A TIME button would clear the contents of the time units, and fill them with new values according to the HOUR and MIN buttons. One click of HOUR or MIN would increase the value by 1 and we display the content on LCD.

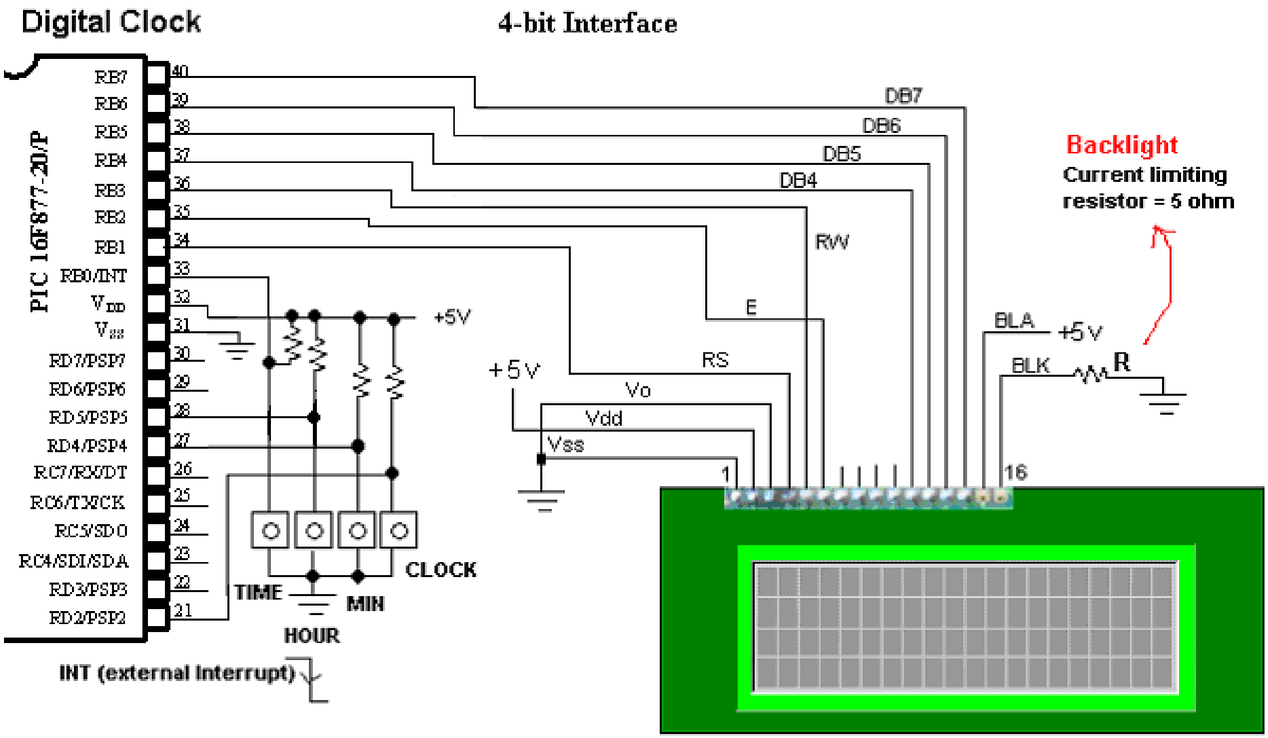


Fig. 72 Interrupt Handler

Let's consider how many different tasks are involved in the interrupt handler. First, we have to detect the button pressing of HOUR or MIN. Then, as they are pressed, we have to display the settings as they are changed. Detecting button presses is not difficult; it only needs a delicate adjustment in time delays in button polling. This will be detailed while explaining the listed code. So read the comment line very carefully for the most sensitive and reliable button reading.

So, our main topic is to remembering the set time by the buttons and displaying them as they are changed, all inside the interrupt handler. So when a keyed-in from say, `HH`, is detected, the content of `HH` is increased by 1. Then, we check if `HH` is 24. If it is 24, we have to change it to 0. For `MM`, if the content is 60, we have to clear the value. After this adjustment, we display the content in decimal format. This is done by calling the hex-to-2 digit decimal conversion subroutine, `h2d2`. Then, we move the cursor of the LCD to the first column of line 2 and write them. The following list of the interrupt handler contains everything we discussed now.

```
;RB0/INT handler
      org         0x04                  ;the interrupt vector address
      banksel     TRISD
      movlw       B'11111100'
      movwf       TRISD                 ;Buttons and LEDs

;Set DDRAM address for the 1st position of line 2 (40h)

      movlw       0xC0                        ;B'11000000'
      call        instw4                      ;RS=0
      call        clear                       ;clear all the contents
```

```
      call        clockLCDdisplay         ;Display 00:00:00
                                          ;as the time setting starts
      banksel     PORTD
      bsf         PORTD,0x01              ;INT indicator on
      call        delay10ms
;CLOCK ADJUSTMENT ROUTINE
;Check for HOUR or MIN Button Pressed
      clrf        STATUS
      movlw       0x03
      movwf       Dtemp             ;this is to check HOUR and MIN buttons
                                    ;3 times at a time with 1 ms delay

HOURCHECK
      call        delay1ms    ;1ms delay is the best one
      banksel     PORTD
      btfss       PORTD, HOUR
      goto        HOURADJ           ;HOUR key is detected
      decfsz      Dtemp
      goto        HOURCHECK

      movlw       0x03
      movwf       Dtemp
      clrf        STATUS
MINCHECK
      call        delay1ms          ;1 ms delay is the next one
      btfss       PORTD, MIN
      goto        MINADJ            ;MIN key is detected
      decfsz      Dtemp
      goto        MINCHECK
ADJDONE
      btfsc       PORTD, CLOCK      ;Wait until the CLOCK
                                    ;start button is pressed
      goto        HOURCHECK         ;IF not, scan again for HOUR/MIN buttons
      bcf         INTCON, INTF      ;Clear the INTF flag
      banksel     PORTD
      bcf         PORTD, 0x01       ;INT indicator off
      retfie                        ;return from interrupt to main program

;hour adjustment
HOURADJ
      clrf        STATUS
      banksel     HH
      incf        HH
      movf        HH,0
      xorlw       B'00011000' ;24=00011000
      btfsc       STATUS,ZERO
      clrf        HH                ;if =24, clear HH
;IF HH=24 set to 0
;
      goto        prep

MINADJ
      clrf        STATUS
      banksel     MM
      incf        MM
;IF MM=60 set to 0
      movf        MM,0
```

```
      xorlw       B'00111100' ;60=00111100
      btfsc       STATUS,ZERO
      clrf        MM                  ;if =24, clear MM
      goto        prep
prep

      banksel     HH                  ;hex-to-decimal conversion
      movf        HH,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       hh1dec
      movf        hms0dec,0
      movwf       hh0dec

      movf        MM,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       mm1dec
      movf        hms0dec,0
      movwf       mm0dec

      movlw       0x00                ;for SS (no adjustment)
      movwf       ss1dec
      movwf       ss0dec

;Set DDRAM address for the 1st position of line 2 (40h)
      movlw       0xC0                      ;B'11000000'
      call        instw4                    ;RS=0
      call        clockLCDdisplay
      call        delay10ms
      goto        ADJDONE             ;scan again for another button press
;end of the interrupt handler
```

The interrupt handler actually takes most of the code of CLOCK4. The following code, with the interrupt handler, for the presentation of the coding structure, shows the CLOCK4 program in all except subroutines and `CBLOCK...ENDC` block.

```
;clock4.asm
;
;DIGITAL CLOCK ON LCD ------the last version
;with Buttons
;
;20x4 LCD module
;by Truly (HD44780 compatible)
;
; 4-bit interfacing
;
; Pin Connection from LCD to 16F877
; LCD (pin#)      16F877 (pin#)
;DB7 (14) -----RB7(40)
;DB6 (13) ----RB6(39)
;DB5 (12) ----RB5(38)
;DB4 (11) ----RB4(37)
;DB3 (10)
```

```
;DB2 (9)
;DB1 (8)
;DB0 (7)
;E (6) ------RB2(35)
;RW (5) -----RB3(36)
;RS (4) -----RB1(24)
;Vo (3) -----GND
;Vdd (2) ----+5V
;Vss (1) -----GND
;
;BUTTONS
;RB0---External INT---TIME SET button (Return to 00:00:00 and ready for
change)
;RD5 --- HOUR button  (increase one at a button)
;RD4 --- MIN button
;RD2 --- CLOCK Button  (Start the clock)
;
;NOTE: RB0 is normal HIGH, and it goes to LOW when the TIME button is
pressed.
;  Therefore (1) INTEDG (OPTION_REG<6>) must be cleared.
;            (2) GIE (Global interrupt) of INTCON must be set
;            (3) INTE (INTCON<4>) must be set to enable INT interrupt
;            (4) Once triggerred, INTF (INTCON<1>) would be set; this
;                must be cleared by software.
;
;Example display:
;     PIC CLOCK
;     HH:MM:SS
;

      list P = 16F877

STATUS      EQU   0x03
PORTB       EQU   0x06
TRISB       EQU   0x86
PORTD       EQU   0x08
TRISD       EQU   0x88
RS          EQU   0x01  ;RB1
E           EQU   0x02  ;RB2
RW          EQU   0x03  ;RB3
CARRY       EQU   0x00
TMR0        EQU   0x01        ;Timer0 module
INTCON      EQU   0x0B        ;Intcon
OPTION_REG  EQU   0x81        ;Option Register
INTEDG      EQU   0x06        ;RB0/INT egde selection (1: rising; 0:falling)

INTE        EQU   0x04        ;RB0/INT enable
INTF        EQU   0x01        ;RB0/INT flag
T0IF        EQU   0x02        ;tmr0 overflow flag
T0IE        EQU   0x05        ;Tmr0 interrupt enable/disable
ZERO        EQU   0x02        ;Zero flag on STATUS (1: zero)
GIE         EQU   0x07        ;Global Interrupt
CLOCK       EQU   0x02        ;CLOCK START BUtton
HOUR        EQU   0x05        ;HOUR adj
MIN         EQU   0x04        ;MINUTE adj
;RAM
```

```
      CBLOCK      0x20
;NOTE INCLUDE THE SAME BLOCK, TO THIS PLACE, USED FOR CLOCK3
;ALONG WITH THE LINE BELOW

            Dtemp
      ENDC

;program should start from 0005h
;0004h is allocated to interrupt handler

      org         0x0000
      goto        START


      org         0x04
;RB0/INT handler
      banksel     TRISD
      movlw       B'11111100'
      movwf       TRISD

;Set DDRAM address for the 1st position of line 2 (40h)

      movlw       0xC0              ;B'11000000'
      call        instw4                  ;RS=0
      call        clear             ;clear all the contents
      call        clockLCDdisplay
      banksel     PORTD
      bsf         PORTD,0x01  ;INT indicator on
      call        delay10ms
;CLOCK ADJUSTMENT ROUTINE
;Check for HOUR or MIN Button Pressed
      clrf        STATUS
      movlw       0x03
      movwf       Dtemp

HOURCHECK
      call        delay1ms    ;1ms delay is the best one
      banksel     PORTD
      btfss       PORTD, HOUR
      goto        HOURADJ
      decfsz      Dtemp
      goto        HOURCHECK

      movlw       0x03
      movwf       Dtemp
      clrf        STATUS
MINCHECK
      call        delay1ms    ;1 ms delay is the bext one
      btfss       PORTD, MIN
      goto        MINADJ
      decfsz      Dtemp
      goto        MINCHECK
ADJDONE
      btfsc       PORTD, CLOCK
;Wait until the CLOCK start button is pressed
      goto        HOURCHECK
      bcf         INTCON, INTF
```

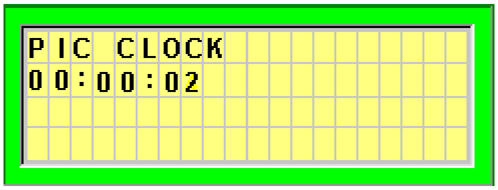
```
      banksel     PORTD
      bcf         PORTD, 0x01       ;INT indicator off
      retfie                        ;return to main program

;hour adjustment
HOURADJ
      clrf        STATUS
      banksel     HH
      incf        HH
      movf        HH,0
      xorlw       B'00011000' ;24=00011000
      btfsc       STATUS,ZERO
      clrf        HH
;IF HH=24 set to 0
;
      goto        prep

MINADJ
      clrf        STATUS
      banksel     MM
      incf        MM
;IF MM=60 set to 0
      movf        MM,0
      xorlw       B'00111100' ;60=00111100
      btfsc       STATUS,ZERO
      clrf        MM
      goto        prep
prep
      banksel     HH
      movf        HH,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       hh1dec
      movf        hms0dec,0
      movwf       hh0dec

      movf        MM,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       mm1dec
      movf        hms0dec,0
      movwf       mm0dec

      movlw       0x00  ;for SS
      movwf       ss1dec
      movwf       ss0dec

;Set DDRAM address for the 1st position of line 2 (40h)
      movlw       0xC0              ;B'11000000'
      call        instw4                  ;RS=0
      call        clockLCDdisplay
      call        delay10ms
      goto        ADJDONE

; END of INT handler
```

```
Start
      BANKSEL     TRISB
; 1 for input, 0 for output

      movlw       0x01
      movwf       TRISB       ;All output except RB0/INT



      banksel     TRISD
      movlw       B'11111100' ;PORTD all inputs except the last two
      movwf       TRISD

      banksel     PORTD
      bcf         PORTD,0x01
      bcf         PORTD, 0x00 ;OFf the LEDs

;LCD routine starts
      call        delay10ms
      call        delay10ms

      banksel     PORTB
      bcf         PORTB, RW   ;RW set LOW here

                          ;give LCD module to reset automatically

;For RB0/INT
      banksel     INTCON
      clrf        INTCON            ;int disabled
      bsf         INTCON, GIE ;interrupt enabled
      bsf         INTCON, INTE      ;RB0/INT enable
;FOR TMR0
      clrf        TMR0
      movlw       0xC7
      banksel     OPTION_REG  ;pre-scaler at 255
      movwf       OPTION_REG  ;10000111  (with INTEDG=0)
      banksel     TMR0
      clrf        TMR0

;END FOR TMR0

;THE ONLY CHANGE IN 4-BIT INTERFACING
;EXCEPT 2 SUBROUTINES
;
;Function for 4-bit (only one write must be done)
;In other words, send only the high nibble
;IMPORTANT
LCDINIT
      movlw       0x28
      call        hnibble4
;Fundtion for 4-bit, 2-line display, and 5x8 dot matrix
      movlw 0x28
      call        instw4
;Display On, CUrsor On, No blinking
      movlw       0x0E        ;0F would blink
```

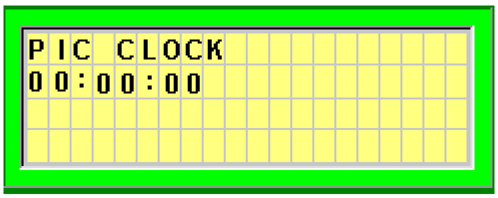
```
      call        instw4
;DDRAM address increment by one & cursor shift to right
      movlw       0x06
      call        instw4

LCDREADY
;DISPLAY CLEAR

      movlw       0x01
      call        instw4

;Set DDRAM ADDRES
      movlw       0x80        ;00
      call  instw4
;WRITE DATA in the 1st position of line 1
      movlw       0x50        ;P
      call        dataw4

      movlw       0x49        ;I
      call        dataw4

      movlw       0x43        ;C
      call        dataw4
      movlw       ' '
      call        dataw4
      movlw       'C'
      call        dataw4
      movlw       'L'
      call        dataw4
      movlw       'O'
      call        ataw4
      movlw       'C'
      call        dataw4
      movlw       'K'
      call        dataw4

;
      call        clear
AGAIN
;CLOCK DISPLAY
;Set DDRAM address for the 1st position of line 2 (40h)

      movlw       0xC0              ;B'11000000'
      call        instw4                  ;RS=0

;CLOCK DISPLAY PART

      movf        SS,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       ss1dec
      movf        hms0dec,0
      movwf       ss0dec

      movf        MM,0
      movwf       hms
```

```
      call        h2d2
      movf        hms1dec,0
      movwf       mm1dec
      movf        hms0dec,0
      movwf       mm0dec

      movf        HH,0
      movwf       hms
      call        h2d2
      movf        hms1dec,0
      movwf       hh1dec
      movf        hms0dec,0
      movwf       hh0dec

      call        clockLCDdisplay
      call        delay1s
      incf        SS
      movf        SS,0
      clrf        STATUS
      xorlw       B'00111100' ;if SS=60(d) or 3C or 0011 1100
      btfss       STATUS, ZERO
      goto        again

      clrf        SS
      incf        MM
      movf        MM,0
      clrf        STATUS
      xorlw       B'00111100'
      btfss       STATUS,ZERO
      goto        again

      clrf        MM
      incf        HH
      movf        HH,0
      clrf        STATUS
                        ;check 24hour 24d = 00011000
      xorlw       B'00011000'
      btfss       STATUS,ZERO
      goto        again
      clrf        STATUS
      call        clear
      goto        again

;SUBROUTINES
;HERE
      END
;end of program
```

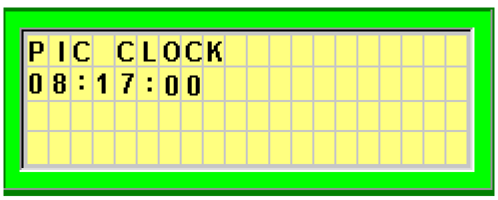
When you compile the full code and run it, the first LCD display would be like this: the clock starts from 00:00:00.



When you press the TIME button, the LCD would go back to 00:00:00. And the clock does not tick, instead, it waits for HOUR, MIN, or CLOCK button.



If you press the buttons of HOUR and MIN, the numbers for HH and MM would increase.



When you finally press the CLOCK button, the digital clock starts to tick from the set time.

If you leave your clock run for a day or so, you may notice that your clock is slightly slower than your watch. The reason is that LCD display consumes a lot of time, a few tens of milli-seconds. Therefore, to make your digital clock reasonably accurate, we reduce down the number of overflows (remember 76) to make an exact 1 second delay. It is very hard to consider all the delay factors in the program and find the exact number of the overflow count, however, just one or two trial and error hopefully gives us the best number. So we change the 1 second time delay to accommodate the delay involved in LCD display, as follows.

```
;DELAY SUBROUTINE for 1 Second delay
;
DELAY1s
      banksel     count
      movlw       0x3C                  ;Count=76 for 1 second to expire
                                        ;lowered to 60 to
                                        ;accommodate LCD delays
      movwf       count

over  btfss       INTCON,     T0IF        ;Tmr0 overflow?
      goto        over
      bcf         INTCON, T0IF            ;reset
      decfsz      count
      goto        over
return
```

4. TIMER 1 and Application to Color Sensing

Timer1 Module

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The Timer1 Interrupt, if enabled, is generated on overflow which is latched in the TMR1IF (PIR1<0>) interrupt flag bit. This interrupt can be enabled/disabled by setting/clearing the TMR1IE (PIE1<0>) interrupt enable bit. Timer1 can operate in one of three modes as a synchronous timer, a synchronous counter, or an asynchronous counter.

This section discusses only of the synchronous counter feature of Timer1 module, counting the pulses entered to either RC0/T1OSI (Pin#15) or RC1/T1OSO (Pin#16) pin. For further and other applications, please refer to the Microchip 16F877 data sheet. The operation of Timer1 is controlled by T1CON register.

T1CON: Timer1 Control Register (10h) for Synchronous Counter Mode

--	--	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON

T1CKPS1:T1CKPS0:
Timer1 Input Clock Prescale
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

T1OSCEN: Timer1 Oscillator Enable bit
1 = **External Clock Pin is RC1/T1OSI**
0 = **External Clock Pin is RC0/T1OSO**

$\overline{\text{T1SYNC}}$: Timer1 External Clock
1 = Do not synchronize external clock input
0 = Synchronize external clock input

TMR1CS: Timer1 Clock Source Select bit
1 = External clock (on the rising edge)
0 = Internal clock (Fosc/4)

TMR1ON: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Since we are reading external clock (or pulse) and we assume that it is not that fast, we normally set the prescaler 1:1 ratio. In other words, we do not delay the sampling of the external pulse, but treat the external clock as it is to count number of pulses per given period.

In the counter mode, there are two pins we can use to apply the external clock pulse: RC0/T1OSO and RC1/T1OSI. Selection of one of them is controlled by the T1OSCEN bit. Setting the bit selects RC1/T1OSO and clearing it does for RC0/T1OSI. Since our counter mode is synchronous, we clear the T1SYNC bit. For TMR1CS bit, we set it for external clock counting. Finally, we set the TMR1ON bit to start the Timer1 module. Counting of the rising edge of the external clock pulse would increase the TMR1 registers (TMR1H and TMR1L) by one. When the content crosses from FFFFh to 0000h, the Timer1 interrupt bit TMR1IF would be set, if interrupt is enabled. Usually, when we count number of pulses within a period, we disable the interrupt, and after the lapse of the time, we stop the timer and read the content of TMR1 register. The initialization of T1CON for counting external clock pulses entered to the pin #15 RC0/T1OSO would be: 00000010. When we start the counting, we set the TMR1ON, bit0 of the T1CON.

Timer1 Counter Application to Color Sensor

Our application of Timer1 module as a counter is to color sensing using Texas Advanced Optoelectronic Solutions (TAOS)'s TCS230 Programmable Color Light-to-Frequency Counter. The TCS230 combines configurable silicon photodiodes and a current-to-frequency converter on single monolithic CMOS integrated circuit.

The output is a square wave (50% duty cycle) with frequency directly proportional to light intensity (irradiance). The full-scale output frequency can be scaled by one of three preset values via two control input pins. Digital inputs and digital output allow direct interface to a microcontroller or other logic circuitry. Output enable (OE) places the output in the high-impedance state for multiple-unit sharing of a microcontroller input line. The light-to-frequency converter reads an 8 x 8 array of photodiodes. Sixteen photodiodes have blue filters, 16 photodiodes have green filters, 16 photodiodes have red filters, and 16 photodiodes are clear with no filters. All 16 photodiodes of the same color are connected in parallel and which type of photodiode the device uses during operation is pin-selectable. Photodiodes are 120 μm x 120 μm in size and are on 144-μm centers.

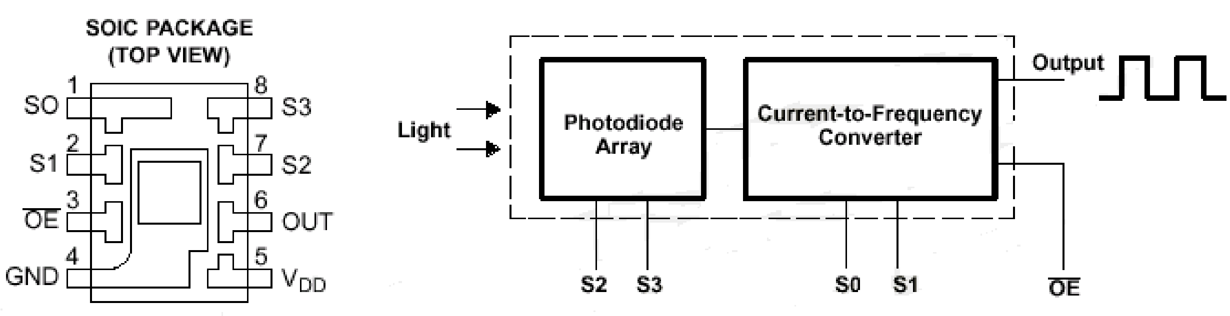


Fig. 73 Soic Package

VDD is for power supply voltage of +5V and ~OE should be Low to enable the color sensor. OUT pin is to generate frequency equivalent of color and luminance level. The frequency of the output can be programmed by S1 and S2 pins, from 100% to 20% to 2% to 0%. When 0% is selected with S1=L and S0=L, the color sensor is actually inactive. The typical full scale (100%) frequency is 600KHz. 20% of the frequency would then be 120KHz, and 2% would be 12KHz. If we have high rate clock pulse and need very accurate count, we may want to use the full frequency, however, in usual application 20% or 2% is just fine.

S0	S1	OUTPUT FREQUENCY SCALING (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

The pins of S2 and S3 determines which color filter we apply. The selection of S2=L and S3=L would focus on red color, while S2=H and S3=H focus on green color. The color determination by TCS230 needs a little experience. Under the same brightness, red color object would generate higher frequency with red filter, and relatively low frequency with green and blue filter. If we increase the brightness of the object, all the frequencies of the three filters would greatly increase. Therefore, the ratio not the frequency themselves is used to determine the true color of an object. Also, you may have to measure the frequency from OUT pin under your test condition. Brightness surrounding the sensor and the object along with the brightness of the LEDs for white light very much effect the nominal frequency of the sensor.

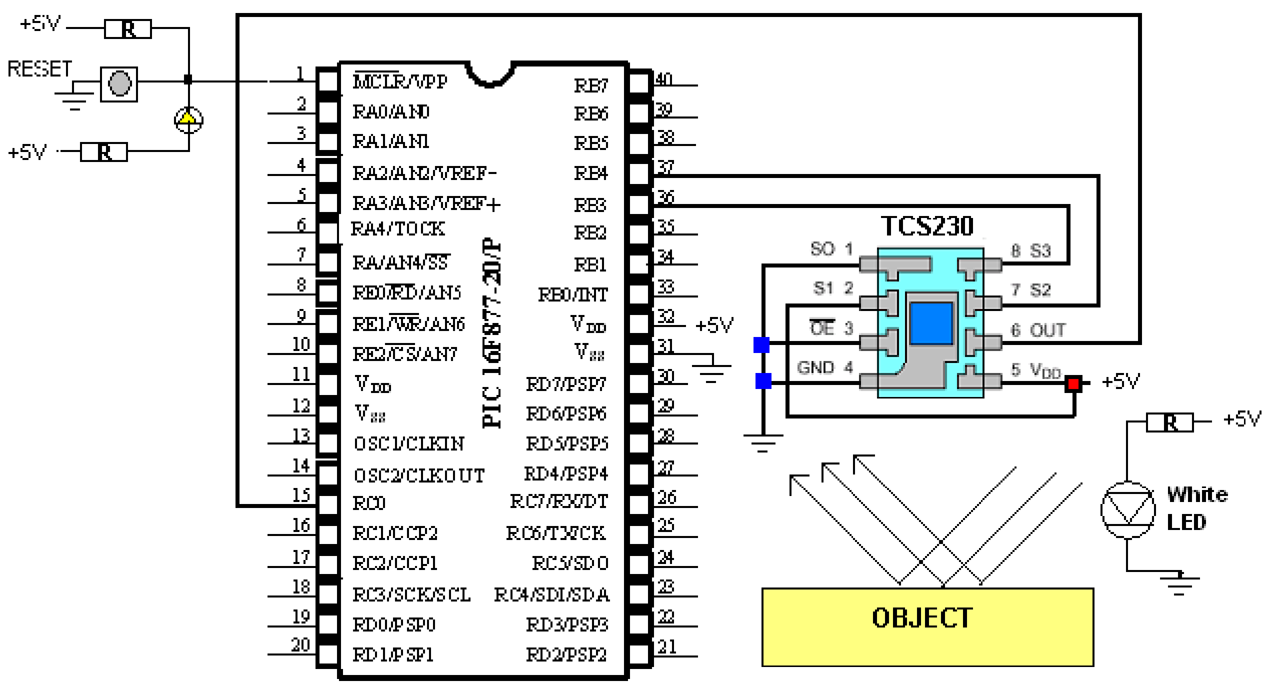


Fig. 74 PIC 16F877 connection to TC230

Since TCS230 is a very small surface mount device (SMD), without a surface mount adaptor such as Model 9165 , a Surfboard series from Capital Advanced Inc, it is almost impossible to implement the sensor.

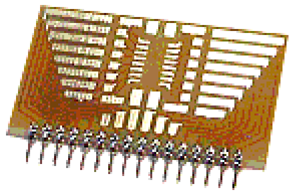


Fig. 75(a) Surfboard

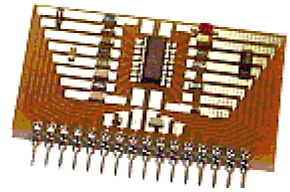
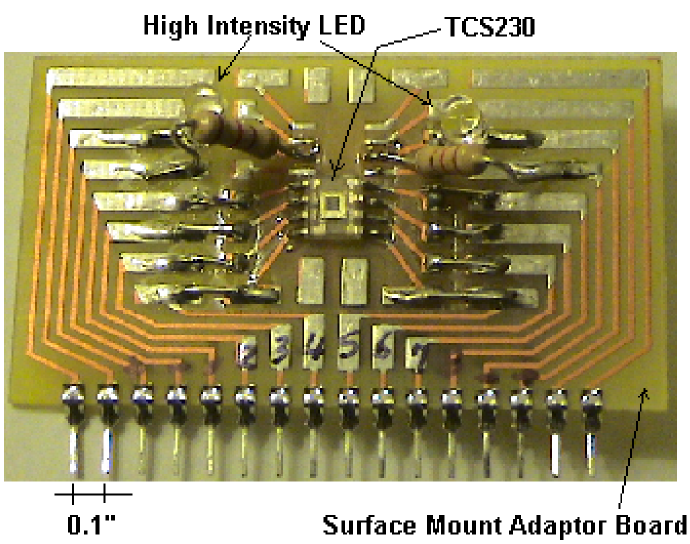


Fig. 75(b) Surfboard with TCS230 mounted on top

Also, providing a white light directly to the object is important, since the color we perceive is nothing but the reflected wave from the object. The following photo shows the author's implementation of a color sensor module with a TCS230, a 9165 Surfboard, and two high intensity white LEDs. Commercial version would have a focus lens on top of the TCS230 to have focused reflected wave from the object.

As illustrated, for 16F877 connection, we tied the ~OE to the ground so that TCS230 is always turn on. By making S0=0 and S1=1, we select 2% of full frequency, i.e., 12 KHz. However, under the author's test condition, the nominal frequency is only about 0.8 KHz for the "full frequency of 12KHZ" configuration. Further test shows that the maximum frequency is about 2.5 KHz. In other words, under the test condition, the maximum number of pulse count would be about 2500 per second. If we limit the counting period to only 100ms, the maximum number would only be 250, which is small enough to be filled only the lower TMR1 register (TMR1L).



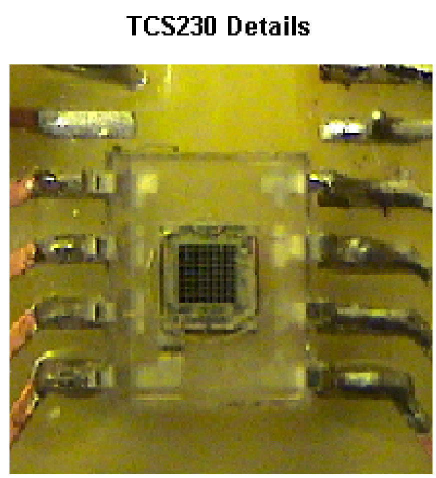


Fig. 76 Implementation of color sensor module

The color filter selection pins S2 and S3 are connected to RB5 and RB4, respectively. The OUT pin of TCS230 is connected to RC0/T1OSO pin of 16F877.

The following example code tries to read a frequency from an object for color determination, by reading 100ms for pulse count from a selected color filter configuration. The frequency counts for Red, Blue, and Green are to be displayed to a PC monitor, in a two-digit hex number format;

`Red1` & `Red0`, `Blue1` & `Blue0`, and `Green1` & `Green0`. The code does not try to determine the color, instead it just spews out the R, G, B, ratios in frequency counts. The color determination is left to the readers. The listing omits the subroutines, as usual.

The readers are encouraged to carefully follow the comments in the following code for better understanding of the program. Note that the `delay1s` subroutine used here does not utilize the Timer0 module; instead this is the first time delay subroutine we made using just numbers of instructions to make 1 second delay. To ease confusion, only delay1s subroutine is included in the subroutine section. All others are omitted.

```
;TCS230.asm
;
; This is to count 50% duty cycle pulses from TCS230 color sensor
; using tmr1 module
; of synchronous counter feature
;
; Output pulse  from TCS230 is connected to RC0 (TICK1)
; Color Filter Selection S2 and S3 are connected to RB5 and RB4 respectively
; S2 (RB5)  S3 (RB4)
; L               L     Red Filter
; L               H     Blue Filter
; H               H     Green Filter
; H               L     No Filter (Clear)
; Output Pulse Frequency Selection S0 and S1 are as follows (for 12 KHz
nominal)
; S0  S1
; L   H           (12 KHz)---actual value is much smaller in a test condition
;                 like 1 – 2KHz
;
        list P = 16F877

STATUS      EQU   0x03
PORTB       EQU   0x06
TRISB       EQU   0x86
PIE1        EQU   0x8C
PIR1        EQU   0x0C
T1CON       EQU   0x10
TMR1L       EQU   0x0E
TMR1H       EQU   0x0F
INTCON      EQU   0x8B
TMR1ON      EQU   0x00
S2          EQU   0x05
S3          EQU   0x04
ZERO        EQU   0x02        ;Z flag
TXSTA       EQU   0x98        ;TX status and control
RCSTA       EQU   0x18        ;RX status and control
SPBRG       EQU   0x99        ;Baud Rate assignment
TXREG       EQU   0x19        ;USART TX Register
RCREG       EQU   0x1A        ;USART RX Register
PIR1        EQU   0x0C        ;USART RX/TX buffer status (empty or full)
RCIF        EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF        EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE      EQU   0x20        ;TXSTA=00100000 : 8-bit, Async
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
```

```
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)

;
;RAM

      CBLOCK      0x20
            TEMP
            RedTEMP
            BlueTEMP
            GreenTEMP
            Red1
            Red0
            Blue1
            Blue0
            Green1
            Green0
            ASCIIreg
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount100ms
            Kount1s
            Kount10s
            Kount1m
      ENDC

;
;=========================================================
      org         0x0000
      GOTO        START
;===========      ===========================================
      org   0x05

START
      call        Async_mode

      BANKSEL     TRISB
      movlw       B'11000000'
      movwf       TRISB             ;PORTB setting for S2 and S3


;TMR1 Initialization
      banksel     T1CON
      clrf        T1CON


      banksel     INTCON
      clrf        INTCON            ;Disable interrupt

      banksel     PIE1
      clrf        PIE1              ;disable peripheral interrupt

      banksel     PIR1
      clrf        PIR1        ;clear peripheral interrupt flag

      banksel     T1CON
```

```
      movlw       '00000010'
      movwf       T1CON       ;1:1 prescaler
                              ;External Clock Source at RC0/T1OSO (pin #15)

                              ;TMR1 is OFF now

AGAIN
      banksel     PORTB
      bcf         PORTB, S2
      bcf         PORTB, S3         ;RED filter is set
      call        delay10ms         ;Wait for the setting is done
      banksel     TMR1H
      clrf        TMR1H
      clrf        TMR1L             ;Clear the counting regsiter
      bsf         T1CON, TMR1ON     ;Tmr1 now starts to increment
      call        delay100ms        ;Continue counting for 100ms
      banksel     T1CON
      bcf         T1CON, TMR1ON           ;TMR1 is OFF
      banksel     TMR1H
;     movf        TMR1H,0
;     movwf       T1HIGH
      movf        TMR1L,0           ;Get the RED count to W
      movwf       RedTEMP           ;Store the RED count to RedTEMP register
; RED is finished
;
      call  delay10ms               ;A short delay before Blue reading
; Go for Blue
      banksel     PORTB
      bcf         PORTB, S2
      bsf         PORTB, S3
      call        delay10ms
      banksel     TMR1H
      clrf        TMR1H
      clrf        TMR1L
      bsf         T1CON, TMR1ON     ;Tmr1 now starts to increment
      call        delay100ms        ;for 100ms

      banksel     T1CON

      bcf         T1CON, TMR1ON           ;TMR1 is OFF

      banksel     TMR1H
;     movf        TMR1H,0
;     movwf       T1HIGH
      movf        TMR1L,0
      movwf       BlueTEMP          ;Blue count
;
      call        delay10ms
; Go for Green
      banksel     PORTB
      bsf         PORTB, S2
      bsf         PORTB, S3
      call        delay10ms
      banksel     TMR1H
      clrf        TMR1H
      clrf        TMR1L
      bsf         T1CON, TMR1ON     ;Tmr1 now starts to increment
```

```
      call        delay100ms        ;for 100ms

      banksel     T1CON

      bcf         T1CON, TMR1ON           ;TMR1 is OFF

      banksel     TMR1H
;     movf        TMR1H,0
;     movwf       T1HIGH
      movf        TMR1L,0
      movwf       GreenTEMP         ;Green pulse count


;Display Preparation


;RED
      movf        RedTEMP,0
      movwf       TEMP
      swapf       TEMP,0      ;SWAP upper and lower nibbles --->W
      andlw       0x0F        ;Mask off upper nibble

      call        HTOA
      movwf       Red1

      movf        RedTEMP,0
      andlw       0x0F        ;mask of upper nibble
      call        HTOA
      movwf       Red0

;Blue
      movf        BlueTEMP,0
      movwf       TEMP
      swapf       TEMP,0    ;SWAP upper and lower nibbles --->W
      andlw       0x0F        ;Mask off upper nibble

      call        HTOA
      movwf       Blue1

      movf        BlueTEMP,0
      andlw       0x0F       ;mask of upper nibble
      call        HTOA
      movwf       Blue0

;Green
      movf        GreenTEMP,0
      movwf       TEMP
      swapf       TEMP,0    ;SWAP upper and lower nibbles --->W

      andlw       0x0F        ;Mask off upper nibble

      call        HTOA
      movwf       Green1

      movf        GreenTEMP,0
      andlw       0x0F       ;mask of upper nibble
      call        HTOA
```

```
      movwf       Green0



;display
;RED
      movlw       'R'
      call        TXPOLL
      movlw       ':'
      call        TXPOLL
      movf        Red1,0
      call        TXPOLL
      movf        Red0,0
      call        TXPOLL
      movlw       ' '
      call        TXPOLL

;BLUE
      movlw       'B'
      call        TXPOLL
      movlw       ':'
      call        TXPOLL
      movf        Blue1,0
      call        TXPOLL
      movf        Blue0,0
      call        TXPOLL
      movlw       ' '
      call        TXPOLL
;GREEN
      movlw       'G'
      call        TXPOLL
      movlw       ':'
      call        TXPOLL
      movf        Green1,0
      call        TXPOLL
      movf        Green0,0
      call        TXPOLL
      movlw       ' '
      call        TXPOLL
      call        CRLF

      call        delay1s            ;1 sec delay after R, G, B readings
      goto        AGAIN

;SUBROUTINE SECTION
;1 sec delay
;call 100 times of 10ms delay
Delay1s
      banksel     Kount1s
      movlw       H'64'
      movwf       Kount1s
R1s   call        Delay10ms
      decfsz      Kount1s
      goto        R1s
      return
;
;INCLUDE OTHER SUBROUTINES
```

```
; HERE
;

      END
;end of program
```

Your running the program would show the following or similar display.