CHAPTER 7 - TIMERS AND COUNTERS

INTRODUCTION

- Most applications will require some type of timing.
- This may be to determine the spacing of input signals or to properly sequence outputs.
- There are three methods that can be used. They are timing loops, internal timer modules, and external timing devices.
- There are three built in timer modules on the PIC16F876. They are Timer 0, Timer 1, and Timer 2.
- This chapter will discuss timing loops, Timer 2, and Timer 1. Timer 0 will be discussed in a later chapter. External timing devices will also be discussed later.
- A counter differs from a timer in that a counter is used to count pulses generated by an external device while a timer runs off the PIC's oscillator.
- This chapter will discuss the use of Timer 1 as a counter.

THE INSTRUCTION CYCLE

- Calculations for timing loops and for internal timer modules are based on the instruction cycle.
- One instruction cycle is the amount of time required to execute a non-branching instruction.
- This time depends on the PIC's oscillator frequency and is as follows.

  $T_i = 4 \cdot T_{osc} = 4 / f_{osc}$
- Example: With a 4MHz oscillator, $T_i = 4 \cdot T_{OSC} = 4 / 4MHz = 1\mu s$

TIMING LOOPS

Instruction Execution Time

- Non-branching instructions require $1 \cdot T_i$ to execute.
- Unconditional branches require $2 \cdot T_i$ to execute. This is because the ALU pre-fetches instructions. When a branch occurs, the pre-fetched instruction must be discarded and a new instruction loaded. This takes an extra instruction cycle.
- Conditional branches require $1 \cdot T_i$ if no branch and $2 \cdot T_i$ if branch occurs.
- Example: How long does it take to execute the following code if a 10MHz oscillator is used?

```
movf PORTB,W
movwf InputA
movwf InputB
swapf InputB
movlw H'0F'
andwf InputA,F
andwf InputB,F
return
```

  7 non-branching and 1 branching instruction $\rightarrow 9 \cdot T_i$

  $T_i = 4 \cdot T_{OSC} = 0.4\mu s$

  Therefore: Time = $3.6\mu s$

- Cannot always determine the exact execution time of a complete program because of the uncertainty of conditional branches. But the execution time of a simple piece of code can be determined.

Timing Loops

- Can use execution time to create intentional time delays.
- Not recommended when precision timing is required.
- For precision timing use the timer modules.

- Example 1: Determine the time delay for the following loop using f_{OSC} = 4MHz.

```
        movlw D'100'        ; 1
Delay   nop                 ; 1 × 100 = 100
        addlw -1            ; 1 × 100 = 100
        btfss STATUS,Z      ; (1 × 99) + 2 = 101
        goto Delay          ; 2 × 99 = 198
```

Total time delay = 500·T_i = 500µs.

- Example 2: Make a subroutine for a delay of 10ms using f_{OSC} = 4MHz.

Need 10,000·T_i. Will make use the previous example code. Execute the above loop 20 times.

```
Delay_10ms  movlw D'20'
            movwf DelayCount
DelayLoop1  movlw D'100'
DelayLoop2  nop
            addlw -1
            btfss STAUTS,Z
            goto DelayLoop2
            decfsz DelayCount,F
            goto DelayLoop1
            return
```

Total delay = 1 + 1 + (20 × 500) + [(19 × 1) + 2] + (19 × 2) + 2 = 10,063·T_i

Including the call for the subroutine this would give a delay of 10.065ms.

- Example 3: Use the above subroutine to get a ½ second delay.

1/2 second = 50 × 10ms.

```
            movlw D'50'
            movwf TimeCount
            call Delay_10ms
            decfsz TimeCount,F
            goto $-2
```

- The advantage of timing loops is that they are easy to write and simple to use.
- The disadvantages are that they are not precise and that you can not do anything else until the time expires.

TIMER 2

- Refer to the Timer 2 section of the data book for more details on Timer 2.
- The built-in timers, such as Timer 2, must be initialized for a specific time delay. The timers may then be started and stopped as desired.
- When the time has expired, a timer flag will be set to indicate such.
- The timers continue to run until stopped. They will set the timer flag every time the prescribed time has passed.
- You may execute other code while the timer is running. The timer does not need attention.

Timer 2 Signal Flow

- The instruction cycle clock is the input signal for Timer 2.
- Timer 2 consist of three counters: the prescaler, the period counter, and the postscaler. This is shown in Figure 7-1. A more complete diagram may be found in the data book.

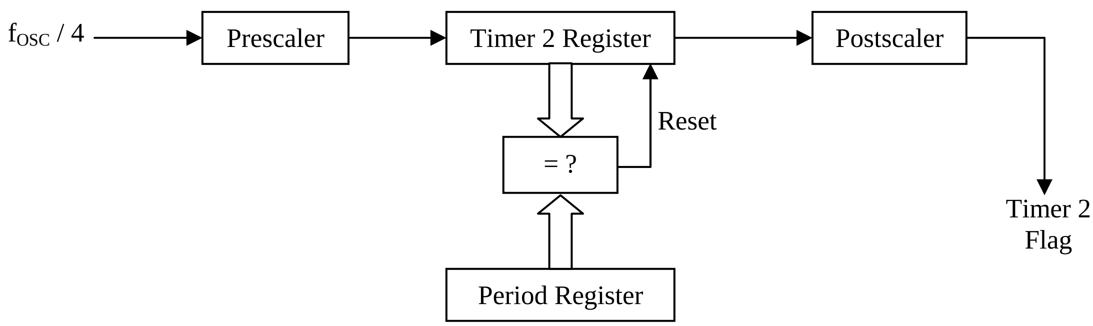

Figure 7-1

- The prescaler counts instruction cycles. When it rolls over, the timer 2 register is incremented.

- When the timer 2 register is equal to the prescribed period, it is cleared and the postscaler is incremented.
- When the postscaler rolls over, the timer 2 flag is set.
- The timer does not automatically stop. It will keep running until your program issues a command to stop it.

<u>Timer 2 Parameters</u>

- The total time delay is calculated as the product of the prescale, period, and postscale times the instruction cycle.

  $T_{delay} = T_i \times \text{Prescale} \times \text{Period} \times \text{Postscale}$

- The values for the prescaler, period, and postscaler must be set prior to starting the timer.
- Allowable values for these parameters are as follows.

  Prescaler: 1, 4, or 16

  Period: 1 to 256

  Postscaler: 1 to 16

- Example: Determine values for the timing parameters to give a 5ms delay for a 4MHz oscillator.

  $5ms = 5000\mu s = 5000 \cdot T_i$

  $5000 = 5 \times 10 \times 10 \times 10 = 5 \times 2 \times 5 \times 2 \times 5 \times 2 \times 5$

  Let Prescale = 4, Period = 250, and Postscale = 5

<u>Timer 2 Initialization</u>

- The prescaler and postscaler are set with the register *T2CON*. The period is set with the register *PR2*.

- Bits 6-3 of T2CON set the postscaler. (See the Timer 2 section of the data book.) The postscaler is equal to the binary equivalent of this 4-bit value plus 1.
- Bits 1 and 0 of T2CON set the prescaler. Refer to the data book for the correct patterns.
- The period will be the value of PR2 plus 1.
- Example: Initialize Timer 2 to give a 5ms delay using the parameters from the previous example.

```
InitTimer2
; Initialize Timer2 for 5ms delay at 4MHz
; Prescale = 4, Period = 250, Postscale = 5
      movlw B'00100001'
      movwf T2CON
      bsf STATUS,RP0
      movlw D'250'-1
      movwf PR2
      bcf STATUS,RP0
      bcf PIR1,TMR2IF
      clrf TMR2
      return
```

Timer 2 Flag

- The flag is a bit in the register *PIR1* (Peripheral Interrupt Register 1). The bit name is *TMR2IF* (Timer 2 Interrupt Flag).
- This bit will be set upon timeout.
- The bit remains set until cleared by software.
- The timer does not stop upon setting or clearing of the flag.

Starting and Stopping the Timer

- The timer is started by setting bit 2 of register T2CON. The bit name is *TMR2ON*.
- The timer is stopped by clearing the TMR2ON bit.
- The prescaler and postscaler counters are cleared by starting or stopping the timer.

- The TMR2 register may be cleared with the *clrf* command. TMR2 should be cleared before starting the timer to get an accurate time delay.

Example Delay Routine

- Assume that Timer2 has been initialized for a 5ms delay.
- Assume also that the timer has been started.
- ```
  Delay5ms
  ; 5ms delay using Timer2
        btfss PIR1,TMR2IF
        goto $-1
        bcf PIR1,TMR2IF
        return
  ```

Example Main Loop Timing

- Suppose you want your main loop to be executed every 5ms.
- Make use of the previous delay routine for this.
- ```
  Main
        call DoThis
        call DoThat
        call DoSomethingElse
        call Delay5ms
        goto Main
  ```
- As long as the other subroutines do not take more than a total of 5ms to execute, the main loop will be executed at precisely 5ms intervals.

Example Longer Delay

- Suppose you want a half-second delay.
- This is too long for one time-out of Timer2 .
- Make a loop to run the 5ms delay 100 times.
- ```
  HalfSecond
  ; 0.5s delay based on Timer2 set up for 5ms.
        movlw D'100'
        movwf TimeCount
        call Delay5ms
        decfsz TimeCount,F
  ```
  ```

```
goto $-2
return
```

TIMER 1

- Timer 1 is a 16-bit timer module.
- It uses a prescaler and a 16-bit period.
- Upon timeout, a flag will be set.
- Timer 1 is used with the CCP1 (Capture-Compare-PWM) module. CCP1 has many features that will be discussed in a later chapter. Only the Timer 1 reset feature will be discussed here.
- Refer to the Timer 1 and CCP1 sections of the data book for more details.

Timer 1 Signal Flow

- The instruction clock is the input for the timer.
- The timer has a prescaler and a 16-bit period. This is shown in Figure 7-2. A more detailed block diagram is shown in the data book.

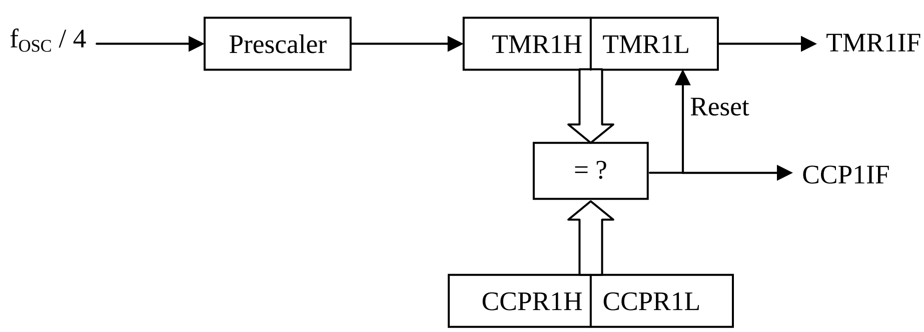

Figure 7-2

- The prescaler counts instruction cycles. When it rolls over, the 16-bit counter comprised of registers TMR1H and TMR1L is incremented.
- When the 16-bit value of TMR1H:TMR1L matches the 16-bit value of CCPR1H:CCPR1L, the 16-bit counter is cleared and a flag is set.

- The timer does not automatically stop. It will keep running until your program issues a command to stop it.
- Timer 1 may be used without the period registers. In this case the 16-bit counter will roll over and set a timer flag.

<u>Timer 1 Parameters</u>

- The time delay is the product of the prescale value and the period times T_i.
- The prescale may be 1, 2, 4, or 8.
- The period may be 0 to 65,535.
- If the CCP1 module is not used to set a period, then the timer will roll over before setting a flag. In this case, the period is effectively 65,536.
- Example: Determine values for the Timer 1 prescale and period to give a 100ms delay for a 4MHz oscillator.

  $100\text{ms} = 100{,}000 \cdot T_i$

  $100{,}000 = 4 \times 25000$

  Use 4 for the prescaler and 25000 for the period.

<u>Timer 1 Initialization</u>

- Register T1CON is used to set the prescale value.
- Registers CCPR1H and CCPR1L are used to set the period. The upper 8 bits must be placed in CCPR1H and the lower 8 bits in CCPR1L.
- Register CCP1CON is used to configure the CCP module as the Timer1 period. Move a value of H'0B' to CCP1CON for this purpose.
- Example: Configure Timer 1 to give a 100ms delay using the parameters of the previous example.

```
InitTMR1
; Initialize TMR1 for a 100ms delay with a 4MHz osc.
; Prescale = 4, Period = 25000
      movlw B'00100000'
      movwf T1CON
      movlw H'0B'
      movwf CCP1CON
      movlw High D'25000'
      movwf CCPR1H
      movlw Low D'25000'
      movwf CCPR1L
      bcf PIR1,CCP1IF
      return
```

- Notice the use of the directives *High* and *Low*. These tell the assembler to use the upper or lower 8 bits of the value given.

Timer 1 Flags

- If CCP1 is used to set the period for Timer 1, then the CCP1IF bit of the PIR1 register will be set upon timeout.
- If CCP1 is not used to set the period, then the TMR1IF bit of register PIR1 will be set upon roll over.
- The flags remain set until cleared by software.
- Setting or clearing of the flags does not stop the timer.

Starting and Stopping the Timer

- The timer is started by setting bit 0 of register T1CON. The bit name is *TMR1ON*.
- The timer is stopped by clearing the TMR1ON bit.

Example Delay Routine

- Assume that Timer1 has been initialized for a 100ms delay.
- Assume also that the timer has been started.

- ```
  Delay100ms
  ; 100ms delay using Timer1
        btfss PIR1,CCP1IF
        goto $-1
        bcf PIR1,CCP1IF
        return
  ```

Example Main Loop Timing

- Suppose you want your main loop to be executed every 100ms.
- Make use of the previous delay routine for this.
- ```
  Main
        call DoThis
        call DoThat
        call DoSomethingElse
        call Delay100ms
        goto Main
  ```
- As long as the other subroutines do not take more than a total of 100ms to execute, the main loop will be executed at precisely 100ms intervals.

TIMER 1 AS A COUNTER

- Rather than counting instruction cycles, Timer 1 may be configured to count external pulses.
- For example, you may wish to count pulses on a tachometer pick-up.

Counter Initialization

- Set the *TMR1CS* bit of register T1CON to make timer 1 a counter.
- Using TRISC assign RC0 as an input.
- Place the signal to be counted on RC0.
- The counter will then count rising edges of RC0.
- The prescaler may be used with the counter. If the prescaler is used, the 16-bit counter value will be incremented upon roll over of the prescaler.
- You may use the counter with or without CCP1 as a period register.

Reading the Count Value

- The counter could be used in a couple of ways.
- CCP1 can be used to provide a period. A flag will then be set to indicate that a certain number of pulses have occurred.
- The counter can also be used without the period. In this case read the 16-bit value of TMR1H:TMR1L at the beginning and again at the end. Take the difference to determine how many pulses have occurred.

Using a Crystal with the Counter

- Timer 1 may be used as a timer/counter with an external crystal.
- Place a crystal between RC0 and RC1. Both these pin must be assigned as inputs.
- The maximum allowable frequency is 200kHz. It is intended for use with 32.768kHz crystals.
- Set the *T1OSCEN* bit of T1CON to enable the oscillator.
- Set the prescaler and period for Timer 1 as described previously. (It may be used without the period also.)
- A flag will then be set at regular intervals.
- This timing mode is primarily intended to wake the PIC from sleep mode at regular intervals. Sleep mode is discussed in a later chapter.

EXTERNAL TIMER CHIPS

- External timer chips are available to provide such functions as date and time keeping.
- These are usually connected to the PIC with a serial interface.