Chapter

13

TIME-INTERVAL MEASUREMENTS

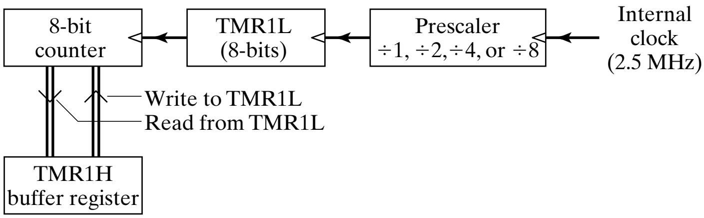
13.1 OVERVIEW

A task common to many microcontroller applications is the measurement of a time interval. This might be an internally generated interval such as the duration of a subroutine. It might be the interval between two edges of a waveform, or between the edge of a "triggering" signal and the delayed output edge of a responding device. The PIC18F452 offers superb capabilities for carrying out time-interval measurements. This chapter will be devoted to this widely used and fundamentally important topic.

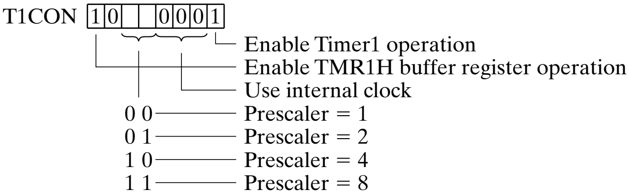
13.2 TIMER1 AND INTERNAL MEASUREMENTS

For internal time-interval measurements, the circuitry of Timer1, shown in Figure 13-1a, is quite similar to that of Timer0, shown in Figures 5-1 and 5-2. Not shown is the optional use of an external clock input, to be discussed in Section 20.10. Also not shown is Timer1's overflow flag and interrupt mechanism, which will be discussed in Section 13.4, to extend timing measurements to time intervals requiring 3 bytes for their expression. Finally, Timer1's interactions with its closely coupled CCP1 facility will be deferred to Section 13.5 for *external* time-interval measurements.

Reads from, and writes to, the 16-bit **TMR1H:TMR1L** register are supported by the same mechanism used to read from and write to **TMR0H:TMR0L**. As was discussed in Section 5.2, at the precise moment that the lower byte of the counter is read, the upper byte is copied into a buffer register. Thus, a subsequent read of **TMR1H** will yield a correct value corresponding to the earlier moment when **TMR1L** was read. This is true even if an intervening interrupt occurred, and the upper byte of the



(a) Counter configuration



(b) **T1CON** initialization

Prescaler	Timing Resolution	Maximum Measurement
÷ 1	0.4 μs	26 ms
÷ 2	0.8 μs	52 ms
÷ 4	1.6 μs	104 ms
÷ 8	3.2 μs	208 ms

(c) Role of prescaler

Figure 13-1 Timer1 for time-interval measurements.

counter (but not the buffer register) incremented between the read of **TMR1L** and **TMR1H**. Even without an intervening interrupt, a read of **TMR1L** when its value is 0xff followed by a read of **TMR1H** will yield the correct 16-bit value even though the upper byte of the counter (but, again, not the buffer register) incremented between the two reads.

The maximum interval to be measured can be extended in either of two ways. The simpler of the two is to initialize Timer1's control register, **T1CON**, to select an appropriate prescaler value, as specified in Figures 13-1b and c. Selecting a divider value of 1 will yield time-interval measurements of up to 26 milliseconds (with an internal clock of 2.5 MHz) having a measurement resolution of 0.4 microseconds, the internal clock period of the chip. Selecting a larger divider will yield a resolution of 3.2 microseconds for measurements up to 208 milliseconds.

In Section 13.4, Timer1 will be extended from its inherent 2-byte counter to a 3-byte counter by incrementing a 1-byte RAM variable each time the 2-byte counter rolls over from 0xffff to 0x0000. This will allow time-interval measurements to extend beyond 6 seconds, even with 0.4 microsecond resolution.

One measurement of interest is the amount of time it takes to execute a specific subroutine. A more useful measurement result, usually, is the *maximum* time it takes to execute the subroutine as its input parameters are varied.

Example 13-1 Microchip's **FXD0808U** subroutine will divide an 8-bit unsigned number by an 8-bit unsigned number. It will be used to convert a number as large as 255 into an ASCII string, using successive divides by 10. How long might an experiment take to check the published maximum execution time of this subroutine for *all* parameter values when it is used to divide a 1-byte number by 10?

Solution

The maximum execution time of the **FXD0808U** subroutine in this case can be found by trying all 256 cases and picking out the maximum. Even if each trial takes the published maximum of 31 cycles (i.e., 12.4 microseconds) for *all* parameter values, the maximum time will be found in 3.2 milliseconds.

Example 13-2 How long will it take to check the maximum execution time for *all* parameter variations?

Solution

Even if the time to try each divisor possibility on all 256 dividend values takes 3.2 milliseconds, the total time will be less than a second.

One of the crowning achievements of the engineering profession is its development of the very tools needed to carry out its various design activities. Two tools of help here are a **START** macro and a **STOP** macro that can be used in the code sequence

```
START
rcall  FXDØ8Ø8U
STOP
```

to measure the execution time to the code between two macros. The **START** macro, listed in Figure 13-2b, simply copies the value read from **TMR1H:TMR1L** to the 2-byte RAM variable **TIMEH:TIMEL**. The **STOP** macro of Figure 13-2c has a two-step job. First, it subtracts the value collected by the **START** macro from a new "snapshot" of the timer, taken at the moment that the

```
subwf  TMR1L,W
```

instruction is executed. Second, it subtracts a small correction, **Magic**. such that the sequence

```
START
STOP
```

produces a value of 0 in **TIMEH:TIMEL.**

Example 13-3 Determine the correction value, **Magic**.

Solution

One way to get this value is to run the **START/STOP** macros with no intervening code and with **Magic** = 0. The small resulting value in **TIMEL** is equal to the required value of **Magic**. That is, if the word **Magic** is replaced by this value, then the execution of **START** followed immediately by **STOP** will yield a result of **TIMEH:TIMEL** = 0.

Another way to determine the value of **Magic** is to count cycles from the read of **TMR1L** in the **START** macro to the read of **TMR1L** in the **STOP** macro. As shown in Figure 13-2d, this produces **Magic** = 5.

```
        TIMEL                   ;Lower byte of the time-interval measurement
        TIMEH                   ;Upper byte
```

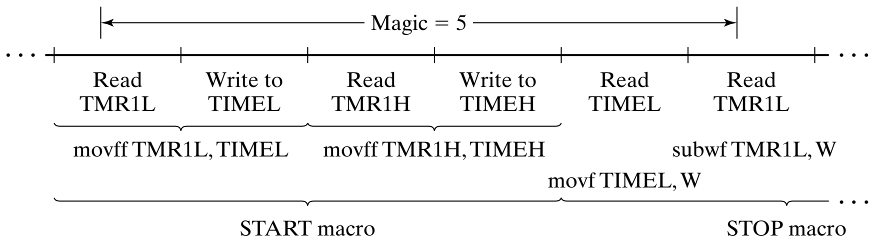
(a) Variables

```
START   macro
        movff  TMR1L,TIMEL
        movff  TMR1H,TIMEH
        endm
```

(b) **START** macro definition

```
STOP    macro
        movf  TIMEL,W           ;TIME = TMR1 - TIME
        subwf  TMR1L,W
        movwf  TIMEL
        movf  TIMEH,W
        subwfb  TMR1H,W
        movwf  TIMEH
        movlw  Magic            ;Make correction
        subwf  TIMEL,F
        btfss  STATUS,C         ;Skip if no borrow
        decf  TIMEH,F
        endm
```

(c) **STOP** macro definition



(a) Timing diagram for Example 13-3, to determine the value of **Magic**

Figure 13-2 START and **STOP** macro definitions

Measuring the maximum of successively collected time intervals requires that each new value be compared with the previous high value. The new value is discarded until it exceeds the previous high, in which case it becomes the new high. Figure 13-3 lists a **MAX** macro to form the maximum in **MAXH:MAXL**.

13.3 DISPLAYMAX SUBROUTINE

In this section, a **DisplayMax** subroutine will be developed. It will make use of two general-purpose utility subroutines to carry out its function:

- **CyclesToMicrosec**—This subroutine converts the number of instruction cycles in the 3-byte variable **AARGB0:AARGB1:AARGB2** into microseconds. It assumes that each cycle lasts 0.4 microseconds. For its use within the **DisplayMax** subroutine, the 2-byte **MAXH:MAXL** will be copied into **AARGB1:AARGB2**, the upper byte **AARGB0** will be cleared, and then the

```
        MAXL                    ;Lower byte of the maximum time-interval
        MAXH                    ;Upper byte
```

(a) Variables

```
        clrf  MAXL              ;Start maximum time interval at zero
        clrf  MAXH
```

(b) Initialization

```
MAX     macro
        movf  TIMEL,W           ;Form MAX - TIME
        subwf  MAXL,W
        movf  TIMEH,W
        subwfb  MAXH,W
        btfss  STATUS,C         ;If TIME > MAX, then update MAX with TIME
        movff  TIMEL,MAXL
        btfss  STATUS,C
        movff  TIMEH,MAXH
        endm
```

(c) The macro definition

Figure 13-3 MAX macro definition.

CyclesToMicrosec subroutine will be called. The number of microseconds will be returned in **AARGB0:AARGB1:AARGB2**.

- **DecimalDisplay**—This subroutine will take the 3-byte number in **AARGB0:AARGB1:AARGB2** and display its value (ranging from 0 to 6,710,886 microseconds) on the second line of the QwikFlash display.

The **CyclesToMicrosec** subroutine of Figure 13-4 carries out the conversion

$$\text{Microseconds} = (\text{Cycles}/10) \times 4$$

Microchip's **FXD2408U** subroutine will divide the number of cycles in **AARGB0:AARGB1:AARGB2** by the number 10 in **BARGB0**. It returns a quotient in **AARGB0:AARGB1:AARGB2** and a remainder with a value between 0 and 9 in **REMB0**. The quotient is then multiplied by 2 by shifting it left one place. Repeating this gives the needed

$$\text{Quotient} \times 4$$

However, the remainder, **REMB0**, from the division by 10 when multiplied by 4 also contributes to the total result. For example, if **REMB0** = 4, then

$$\textbf{REMB0} \times 4 = 16$$

This should be rounded to the nearest multiple of 10,

$$\textbf{REMB0} \times 4 = 20$$

and then the tens digit added to

$$\text{Quotient} \times 4$$

formed earlier. The **CyclesToMicrosec** subroutine treats the **REMB0** value as a BCD number and doubles it twice by adding it to itself twice, using BCD addition. Five is added to the result, using BCD addition, so that the tens digit will represent the correct rounded value. The units digit is cleared and the tens digit is swapped to the units digit position in **WREG**. This is added to **AARGB0:AARGB1:AARGB2** to produce the final result.

```
;;;;;;; CyclesToMicrosec subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  This subroutine converts AARGB0:AARGB1:AARGB2 from cycles to microseconds.
;            Microseconds = Ø.4 Cycles = (Cycles/1Ø)x4

CyclesToMicrosec
        MOVLF  1Ø,BARGBØ        ;Divide by 1Ø
        call  FXD24Ø8U
        bcf  STATUS,C           ;Multiply by two
        rlcf  AARGB2,F
        rlcf  AARGB1,F
        rlcf  AARGBØ,F
        rlcf  AARGB2,F          ;Do it again
        rlcf  AARGB1,F
        rlcf  AARGBØ,F
        movf  REMBØ,W           ;Get remainder and double it
        addwf  WREG,W           ; as a BCD number
        daw
        addwf  WREG,W           ;Double it again
        daw
        addlw  5                ;Round off
        daw
        andlw  Øxf Ø            ;Keep just tens digit
        swapf  WREG,W           ; and move it to the units position
        addwf  AARGB2,F         ; and add it to AARG
        clrf  WREG
        addwfc  AARGB1,F
        addwfc  AARGBØ,F
        return
```

Figure 13-4 CyclesToMicrosec subroutine.

The other general-purpose utility subroutine, **DecimalDisplay**, does a job similar to that of the **ByteDisplay** subroutine of Figure 7-18e, which writes a binary representation of the variable, **BYTE**, to the LCD. The **DecimalDisplay** subroutine writes a decimal representation of **AARGB0:AARGB1: AARGB2** to the second line of the display.

As shown in Figure 13-5, the **DecimalDisplay** subroutine divides the number by 10, converts the one-digit remainder to ASCII, and inserts it into the string at the right-most character position. This is repeated seven more times to fill the eight character positions of the string.

Next, leading zeros are blanked. The cursor-positioning code for the second line of the LCD is written to the beginning of the string and the ‹EOS› character (0x00) is tacked onto the end of the string. Finally, the string is sent to the display.

Given the **CyclesToMicrosec** and the **DecimalDisplay** subroutines as building blocks, the **DisplayMax** subroutine to display the value of **MAXH:MAXL** is easily obtained. It is shown in Figure 13-6.

13.4 EXTENDED INTERNAL MEASUREMENTS

The range of Timer1 can be extended by incrementing a 1-byte RAM variable, **TMR1X** (“Timer1 extension”), each time **TMR1H:TMR1L** overflows from 0xffff to 0x0000. At that moment, the **TMR1IF** flag will be set, as shown in Figure 13-7, and can be used to generate either a high-priority or a low-priority interrupt. If no other interrupt sources require fast service, then the simplicity of a single high-priority interrupt service routine affords the solution shown in Figure 13-8.

```
;;;;;;; DecimalDisplay subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Display whatever is in AARGBØ:AARGB1:AARGB2 as a decimal number on line 2
; of the LCD

DecimalDisplay
        lfsr  Ø,BYTESTR+8
        REPEAT_
          MOVLF  1Ø,BARGBØ       ;Divide AARG by ten
          call  FXD24Ø8U
          movf  REMBØ,W          ;Get digit
          iorlw  Øx3Ø            ;Convert to ASCII
          movwf  POSTDECØ        ; and move to string
          movf  FSRØL,W          ;Done?
          sublw  low BYTESTR
        UNTIL_  .Z.

        REPEAT_                  ;Blank leading zeros
          movlw  A'Ø'            ;ASCII code for zero
          subwf  PREINCØ,W       ;Leading zero?
          IF_  .Z.               ;If so, then blank it
            MOVLF  A' ',INDFØ
          ELSE_                  ;Otherwise, done with blanking
            BREAK_
          ENDIF_
          movf  FSRØL,W          ;In any case, stop at least-significant digit
          sublw  low BYTESTR+7
        UNTIL_  .Z.

        lfsr  Ø,BYTESTR          ;Set pointer to display string
        MOVLF  ØxcØ,BYTESTR      ;Add cursor-positioning code
        clrf  BYTESTR+9          ;and end-of-string terminator
        rcall  DisplayV
        return
```

Figure 13-5 DecimalDisplay subroutine.

```
;;;;;;; DisplayMax subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine takes MAXH:MAXL, converts it to microseconds, and displays
; it on the second line of the LCD.

DisplayMax
        movff  MAXL,AARGB2
        movff  MAXH,AARGB1
        clrf  AARGBØ
        rcall  CyclesToMicrosec
        rcall  DecimalDisplay
        return
```

Figure 13-6 DisplayMax subroutine.

Reading the 3-byte value of **TMR1X:TMR1H:TMR1L** requires care to obtain a correct reading under worst-case circumstances.

Example 13-4 Assuming that the 3 bytes are read in the order **TMR1L**, **TMR1H**, and *then* **TMR1X**, give an example of an invalid reading.

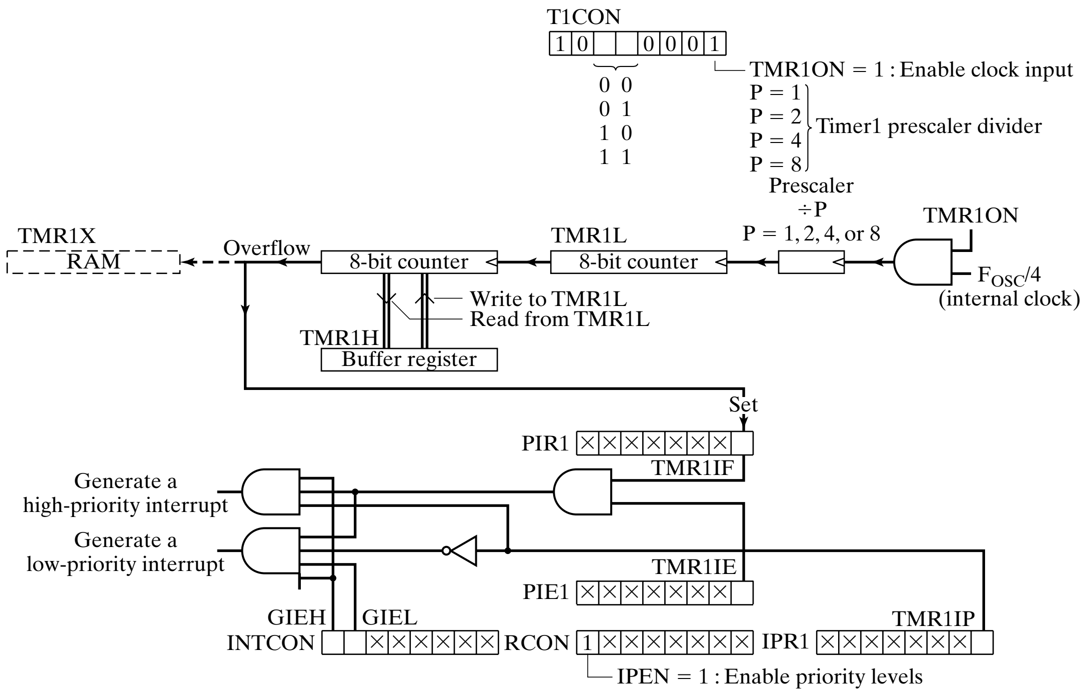


Figure 13-7 Timer1 for extended time-interval measurements.

```
TMR1X                   ;Extension of TMR1
```

(a) Variable

```
bsf  IPR1,TMR1IP        ;Assign high priority to TMR1 overflow interrupt
bcf  PIR1,TMR1IF        ;Clear flag
bsf  PIE1,TMR1IE        ;Enable TMR1 overflow interrupts
bsf  RCON,IPEN          ;Enable high/low interrupt structure
bsf  INTCON,GIEH        ;Enable high priority interrupts to CPU
return
```

(b) Last six instructions of the **Initial** subroutine, setting up the high-priority interrupt

```
org  0x0008             ;High priority interrupt vector
bcf  PIR1,TMR1IF        ;Clear flag
incf  TMR1X,F           ;and increment TMR1 extension
retfie  FAST
```

(c) High-priority interrupt service routine

Figure 13-8 Incrementing **TMR1X**, the extension of **TMR1H:TMR1L**.

Solution

If the three registers are read in this order, and produce a hex value of 35:ff:ff, the correct value is actually 34:ff:ff. The interrupt occurring as the hardware counter rolls over increments **TMR1X** from 0x34 to 0x35. Therefore, by the time **TMR1X** is read, the wrong value is read.

A solution to this ambiguity is to read the 3 bytes in the order **TMR1X**, **TMR1L**, **TMR1H**. If the most-significant bit (MSb) of **TMR1H** is 1, then the **TMR1X** value is valid because it was read sometime during the 32,768 counts before the overflow, when **TMR1H:TMR1L** was equal to B'1bbbbbbb bbbbbbbb'. On the other hand, if the MSb of **TMR1H** is 0, the value of **TMR1X** has the possibility of being invalid. This would be the case if the 3-byte hex number were read as 43:00:00 because the 0x43 was read first, before the hardware counter rolled over. The correct value of the counter at the instant that **TMR1L** was read is 44:00:00. A correct reading is assured when the MSb of **TMR1H** is 0 if **TMR1X** is simply read again. Observe that the instruction sequence

```
movf  TMR1H,W
movwf  TIMEH
```

will set the **STATUS** register's **N** bit if the MSb of **TMR1H** is set, while at the same time copying **TMR1H** to **TIMEH**. These considerations lead to the **STARTX** macro of Figure 13-9c, which copies the 3-byte **TMR1** (i.e., **TMR1X:TMR1H:TMR1L**) to the 3-byte RAM variable **TIME** (i.e., **TIMEX:TIMEH:TIMEL**).

The **STOPX** macro determines the number of cycles that have been executed since the **STARTX** macro was executed, putting this value into the 3-byte variable, **TIME**. It first reads **TMR1** into **TMR1BUF**. Then it subtracts **TIME** (collected by the **STARTX** macro) from **TMR1BUF**, putting the result into **TIME**. Finally, it subtracts from **TIME** the "Magic number," 9, so that the back-to-back execution of

```
STARTX
STOPX
```

will produce **TIME** = 0.

The **MAX3** macro ratchets the 3-byte **MAX** variable (i.e., **MAXX:MAXH:MAXL**) up to the maximum value of **TIME**. When a new value of **TIME** is formed, it is compared with the previous highest value, located in **MAX**. If the new value of **TIME** is larger, this new value replaces the previous value of **MAX**. Thus, the sequence

```
STARTX
‹code whose maximum duration is to be determined›
STOPX
MAX3
```

forms the maximum duration in **MAXX:MAXH:MAXL**.

13.5 CCP1 AND EXTERNAL MEASUREMENTS

The PIC18F452 includes a capture/compare/pulse-width-modulation facility called CCP1 that can be closely coupled to Timer1 to measure time intervals between signal edges occurring on the RC2/CCP1 pin. Figure 13-10 illustrates the connection and its setup. With **T1CON** initialized to B'10000001' and with **CCP1CON** initialized to B'00000101', both prescalers will be bypassed. The **CCP1IF** flag in the **PIR1** register will be set when a rising edge occurs on the CCP1 input pin. In addition, **TMR1H:TMR1L** will be copied to **CCPR1H:CCPR1L** at that precise moment.

In the case of an internal time-interval measurement, the code to be executed to make the measurement is executed automatically at the beginning of each measurement (with the **START** macro) and at the end of each measurement (with the **STOP** and **MAX** macros). The CPU has all the time it needs to do the task being measured.

```
TMR1X                   ;Extension of TMR1
TMR1LBUF                ;Temporary buffer for TMR1L
TMR1HBUF                ;Temporary buffer for TMR1H
TMR1XBUF                ;Temporary buffer for TMR1X
TIMEL                   ;Lower byte of the time-interval measurement
TIMEH                   ;Upper byte
TIMEX                   ;Extension byte
MAXL                    ;Lower byte of maximum measurement
MAXH                    ;Upper byte
MAXX                    ;Extension byte
```

(a) Variables

```
clrf  MAXL              ;Start maximum time interval at zero
clrf  MAXH
clrf  MAXX
```

(b) Initialization

```
STARTX  macro                   ;Save TMR1 in TIME
        movff  TMR1X,TIMEX
        movff  TMR1L,TIMEL
        movf  TMR1H,W           ;Copy TMR1H to TIMEH and copy bit 7 to N
        movwf  TIMEH
        btfss  STATUS,N         ;Does TMR1 = B'Øbbbbbbb bbbbbbbb'?
        movff  TMR1X,TIMEX      ;If so, then reread TMR1X
        endm
```

(c) **STARTX**

```
STOPX   macro                   ;Form TIME = TMR1 - TIME
        movff  TMR1X,TMR1XBUF   ;Form valid reading in TMR1BUF
        movff  TMR1L,TMR1LBUF
        movf  TMR1H,W
        movwf  TMR1HBUF
        btfss  STATUS,N         ;Does TMR1 = B'Øbbbbbbb bbbbbbbb'?
        movff  TMR1X,TMR1XBUF   ;If so, then reread TMR1X

        movf  TIMEL,W           ;Form TIME = TMR1BUF - TIME
        subwf  TMR1LBUF,W
        movwf  TIMEL
        movf  TIMEH,W
        subwfb  TMR1HBUF,W
        movwf  TIMEH
        movf  TIMEX,W
        subwfb  TMR1XBUF,W
        movwf  TIMEX

        movlw  9                ;Magic = 9; Make correction
        subwf  TIMEL,F
        btfss  STATUS,C
        decf  TIMEH,F
        btfss  STATUS,C
        decf  TIMEX,F
        endm
```

(d) **STOPX**

```
MAX3    macro
        movf  TIMEL,W           ;Form MAX - TIME for three-byte numbers
        subwf  MAXL,W
        movf  TIMEH,W
        subwfb  MAXH,W
        movf  TIMEX,W
        subwfb  MAXX,W          ;C=Ø if TIME > MAX
        btfss  STATUS,C         ;Replace MAX with TIME if C=Ø
        movff  TIMEL,MAXL
        btfss  STATUS,C
        movff  TIMEH,MAXH
        btfss  STATUS,C
        movff  TIMEX,MAXX
        endm
```

(e) **MAX3**

Figure 13-9 STARTX, **STOPX**, and **MAX3** macros.

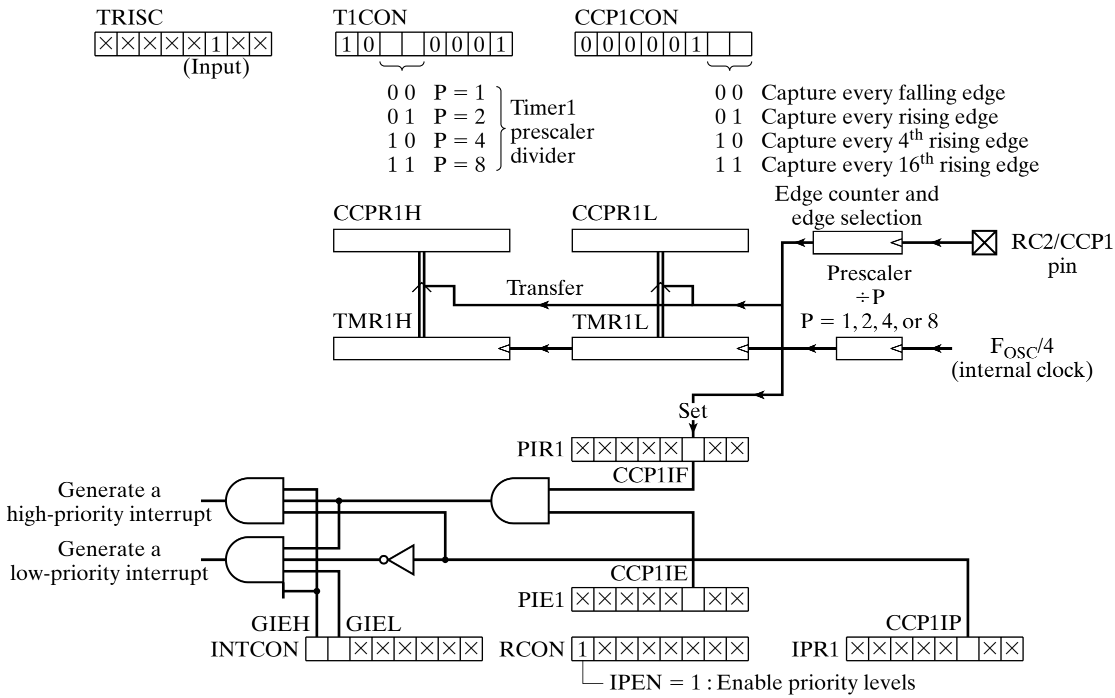


Figure 13-10 CCP1/Timer1 capture mode.

To achieve this same functionality for external time-interval measurements, both the start edge and the stop edge must generate an interrupt. Consider the measurement of a positive pulse (i.e., rising edge to falling edge). Within the CCP1 interrupt handler, if bit 0 of **CCP1CON** is set, then a rising-edge interrupt has occurred and the 2-byte **CCPR1** value can be copied into **TIME**. If it is cleared, then **TIME** can be replaced by **CCPR1** − **TIME**. The **MAX** macro of Figure 13-3 can then be invoked to ratchet up the maximum time interval whenever the latest measurement exceeds the previous maximum. Finally, the CCP1 interrupt handler can toggle bit 0 of **CCP1CON** in preparation for the next edge, clear the **CCP1IF** flag, and return.

Within the mainline loop, the display of the maximum value can be updated every second by counting loop times. Every 100th time around the mainline loop, **MAXH:MAXL** can be read by the **DisplayMax** subroutine of Figure 13-6 and displayed.

Example 13-5 Does the reading of **MAXH:MAXL** in this case constitute a critical region that should be protected by disabling interrupts, reading **MAXH:MAXL**, and then reenabling interrupts?

Solution

The reading does constitute a critical region. Between the reading of **MAXL** and the reading of **MAXH** in the **DisplayMax** subroutine, a CCP1 interrupt might change the value read. The result would be **MAXH(new):MAXL(old)**. If the old value was 00:fe and the new value is

01:02, then the value read would be 01:fe. It would be read and displayed, probably invalidating the on-going measurement.

Example 13-6 What determines the minimum pulse width of the pulse to be measured in this way?

Solution

In response to the leading edge of the pulse, the CPU must get to the CCP1 interrupt handler. If CCP1 is the only high-priority interrupt, then in the worst case, it is put off by the longest critical region in the mainline code. Within the handler, if bit 1 of **CCP1CON** equals 1, then this is the rising (i.e., leading) edge of the pulse. **CCPR1H:CCPR1L** must be copied to **TIMEH:TIMEL**, the bit 1 of **CCP1CON** toggled, and the **CCP1IF** flag bit cleared. At this point, even as the

```
retfie  FAST
```

instruction is being executed, the falling edge of the pulse can occur and its time will be successfully captured.

Since the time to respond to the trailing edge of the pulse takes somewhat longer, the minimum interval between pulses must be somewhat longer than the minimum pulse width asked for in this example.

13.6 CCP1 AND INTERNAL MEASUREMENTS

Internal time-interval measurements have already been examined in great detail. However, the use of the CCP1/Timer1 combination offers an interesting alternative. In support of this alternative, the RC2/CCP1 pin is initialized as an output, but with nothing connected to it. Then the **START** and **STOP** macros are redefined as

```
START  macro
       bsf  PORTC,RC2
       endm

 STOP  macro
       bcf  PORTC,RC2
       endm
```

The execution of the **START** macro will cause the output pin to go high and will trigger a CCP1 capture. The execution of the **STOP** macro will complete the measurement.

13.7 EXTENDED EXTERNAL MEASUREMENTS

By extending Timer1 to a 3-byte counter, as discussed in conjunction with Figure 13-7, external time-interval measurements can be extended to 3-byte values. Each Timer1 overflow can be handled with a low-priority interrupt. Each CCP1 interrupt might be handled with a high-priority interrupt if the minimum pulse width to be measured is less than 10 microseconds or so. For longer pulse-width measurements, CCP1 can be fielded with a low-priority interrupt, if the high-priority interrupt mechanism is to be reserved for some other application requiring its zero-latency feature.

Reading the Timer1 RAM extension variable, **TMR1X**, within the CCP1 handler requires the same care and technique used in Section 13.4. A valid 3-byte time stamp is thereby produced by each capture.

13.8 TIMER3 AND CCP2 USE

Timer3 has essentially the same capabilities as Timer1, as shown in Figure 13-11. Likewise, CCP2 has essentially the same capabilities as CCP1, as shown in Figure 13-12. As pointed out in Figure 13-11, **T3CON** contains two control bits that afford any one of three options:

- CCP1 and CCP2 can both be associated with Timer1.
- CCP1 and CCP2 can both be associated with Timer3.
- CCP1 can be associated with Timer1 while CCP2 is associated with Timer3.

Having two completely independent units can be useful for high-resolution measurements (with the timer's prescaler = 1) and for extended-range measurements (with the other timer's prescaler = 8). Another rationale for having two completely independent units arises when the CCP2/Timer3 is used in a "trigger special event" mode. It can trigger the analog-to-digital converter to start successive conversions automatically, with an arbitrary sample period, as will be discussed at the end of Section 16.3. Meanwhile, the CCP1/Timer1 combination can be used for captures or compares.

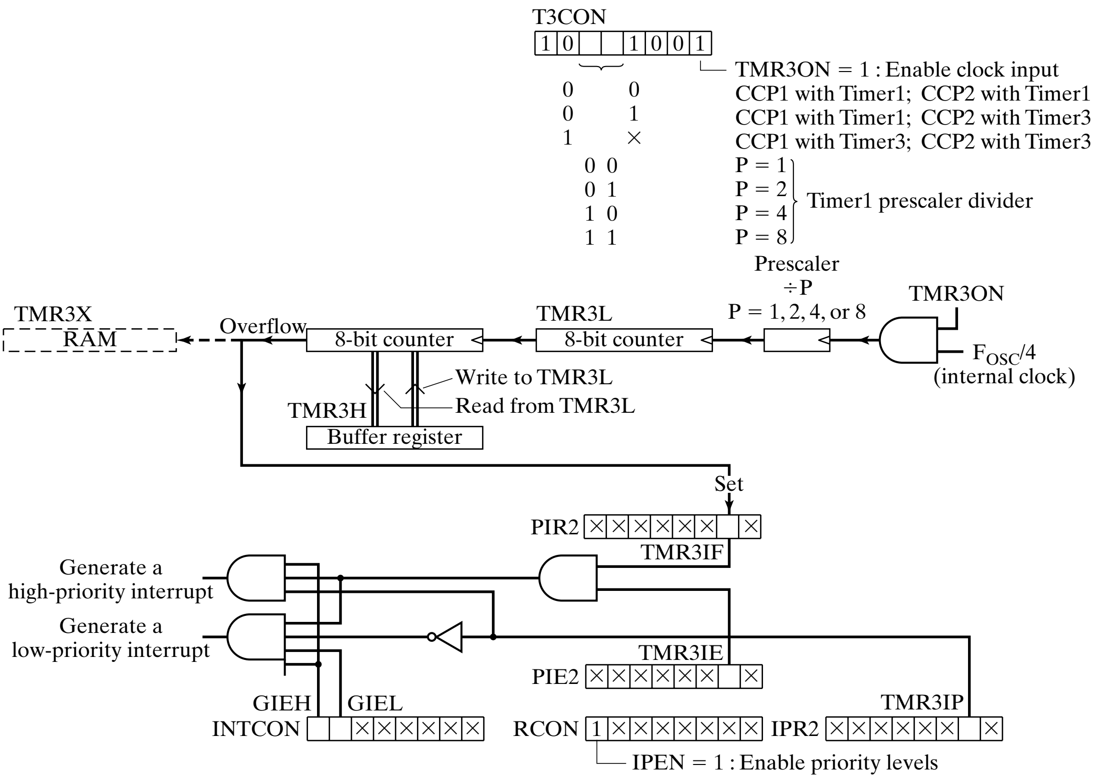


Figure 13-11 Timer3 operation.

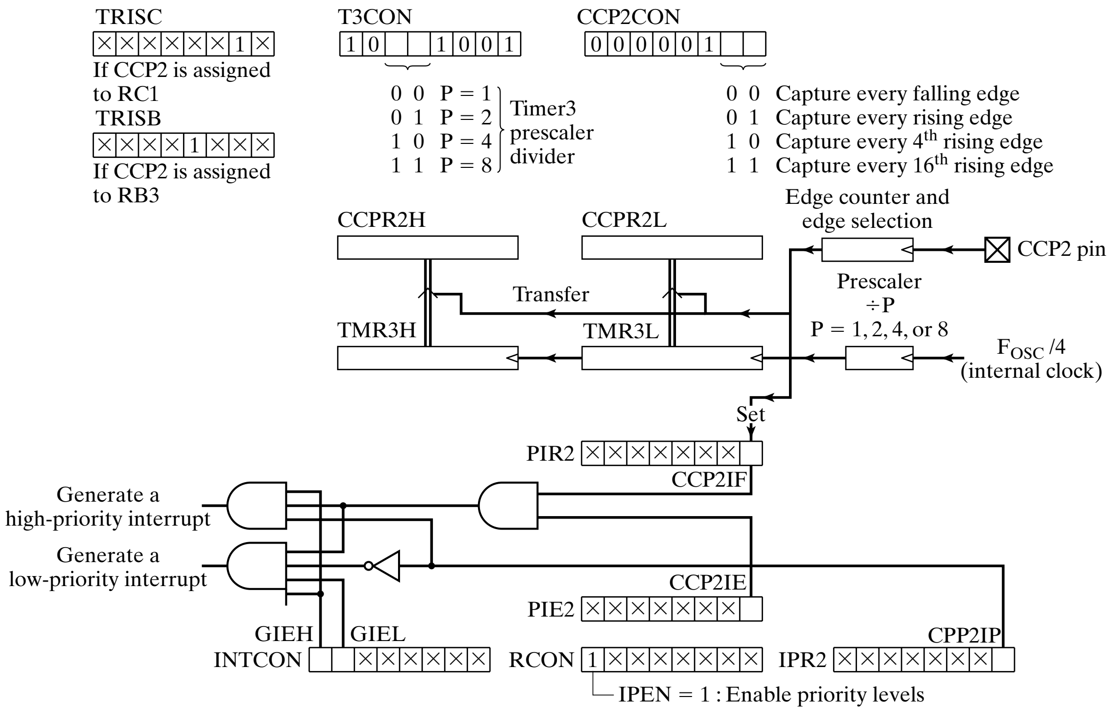


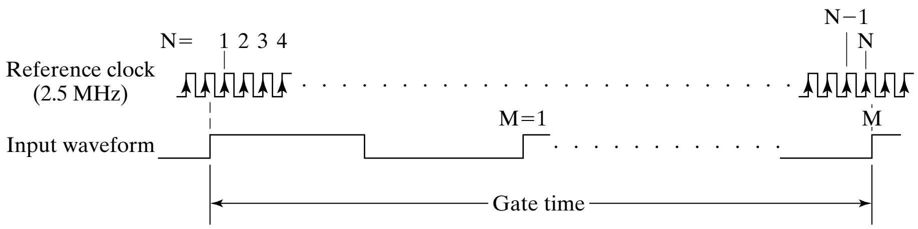
Figure 13-12 CCP2/Timer3 capture mode.

13.9 FREQUENCY MEASUREMENT

The QwikFlash instrument described in Chapter 4 will measure the frequency of the input to the RC1/CCP2 pin with the 50 parts-per-million accuracy afforded by the internal clock. A timing diagram of the measurement process is illustrated in Figure 13-13. Using the 3-byte **TMR3** (i.e., **TMR3X:TMR3H:TMR3L**) as a time base, the measurement begins when CCP2 is triggered by a rising edge of the input waveform to capture the start time (i.e., the value of **TMR3** at that time). Each successive rising edge of the input waveform must be counted. For high frequencies, this counting can be expedited by capturing every 16^{th} rising edge with **CCP2CON** = B'00000111', as specified in Figure 13-12. Within the interrupt service routine for CCP2, the **CCP2IF** flag is cleared, and a 3-byte **MX:MH:ML** variable can be incremented by 16. **TMR3** must be checked to determine whether the gate time has been exceeded, signaling the end of the measurement. If so, the **CCP2IE** interrupt enable bit is cleared to turn off further interrupts. The captured start time is subtracted from the captured stop time to form **NX:NH:NL**, the number of internal clock periods between **MX:MH:ML** cycles of the input waveform. The frequency is then calculated as

$$\text{Frequency} = \frac{M}{N} \times 2{,}500{,}000 \text{ Hz}$$

The multiplication and division subroutines for carrying out this calculation will be discussed in the next chapter.



For a gate time of $\approx$ 0.4 seconds and a 2.5 MHz reference clock, n $\approx$ 1,000,000.
Start measurement on a rising edge of the input waveform.
Stop measurement on the first rising edge of the input waveform after the nominal gate time has been exceeded.
M equals the integral number of clock periods of the input waveform occurring between Start and Stop.
N equals the number of reference clock periods occurring between Start and Stop.
Period = $(N/M) \times 0.4$ microseconds
Frequency = $(M/N) \times 2500000$ Hz
Resolution = ± 1 part in $\approx$ 1,000,000

Figure 13-13 Timing diagram for frequency measurement.

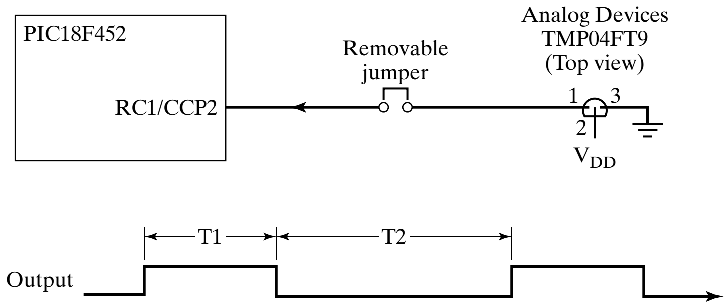
Determining when the gate time has been exceeded would seem to require that, within the CCP2 interrupt handler, the newly captured value of **TMR3** minus the start time be checked to see if it has exceeded the nominal gate time value of 1,000,000. If so, then the measurement has been completed. A simpler procedure entails noting that 1,000,000 = 0x0f4240. If **TMR3X** is initialized to 0x2f, then bit 6 of **TMR3X** will be set after as few as 0x400000 − 0x2fffff = D'1048577' clock cycles. Because the role of the gate time is to determine the resolution of the measurement, this will yield (slightly) better than one part-per-million resolution.

Within the mainline program, the **CCP2IE** interrupt enable bit can be monitored each time around the mainline loop. When the CCP2 interrupt handler clears it, signaling the end of the measurement, the mainline code takes the start time, the stop time, and **MX:MH:ML** and calculates and displays the frequency. A new measurement can be initiated by clearing the **CCP2IF** flag bit. Bit 7 of **TMR3X** can be set as a signal to the CCP2 interrupt handler that a new measurement has begun, so that it will, in turn, reinitialize **TMR3X** to the 0x2f value (discussed in the previous paragraph) and collect the start time. Finally, the **CCP2IE** bit is set, enabling CCP2 interrupts. The next rising edge of the input waveform will initiate a new measurement.

13.10 TEMPERATURE MEASUREMENT

In Section 10.3, the use of the voltage-output temperature transducer on the QwikFlash board was discussed in conjunction with the on-chip analog-to-digital converter. That transducer has a sensitivity of 10 millivolts per degree Fahrenheit while the ADC has a resolution of 5000 millivolts/1024. This translates into a measurement resolution of about half a degree Fahrenheit per increment. In Sections 15.8 and 17.9, two direct digital output temperature transducers will be considered, each using a serial output mechanism to transfer the temperature measurement back to the PIC18F452 microcontroller in Centigrade form.

An interesting alternative is presented by Analog Devices' TMP04 temperature transducer, available in the same TO-92 package as the LM34DZ part used on the QwikFlash board. Alternatively, it is



For Centigrade measurement with 1 °C resolution:

$$\text{Temperature} = 235 - \frac{400\,\text{T1}}{\text{T2}}$$

For Centigrade measurement with 0.1° C resolution:

$$\text{Temperature} = 2350 - \frac{4000\,\text{T1}}{\text{T2}}$$

For Fahrenheit measurement with 1° F resolution:

$$\text{Temperature} = 455 - \frac{720\,\text{T1}}{\text{T2}}$$

For Fahrenheit measurement with 0.1° F resolution:

$$\text{Temperature} = 4550 - \frac{7200\,\text{T1}}{\text{T2}}$$

Figure 13-14 Temperature measurement via time-interval measurements.

available in SO-8 and TSSOP-8 surface-mount packages. With a typical accuracy of ±1.5°C up to 100°C, it would seem to offer no advantage over the other choices. However, its output comes in the form of a pulse-width-modulated output having a nominal frequency of 35 Hz at room temperature. As shown in Figure 13-14, the output swings between 0 V and V_{DD}. The edges can be used to trigger CCP1 or CCP2 capture interrupts for time-interval measurements. Each period of the output consists of a “high” segment, denoted as T1, and a “low” segment, denoted as T2. T1 is nominally 10 milliseconds and is relatively insensitive to temperature change. (Analog Devices notes that T1 will not exceed 12 milliseconds over the rated temperature range of −25°C to +100°C.) With the equations of Figure 13-14, the nominal value of T2 at room temperature is about 19 milliseconds. These values for T1 and T2 mean that the measurements will be made with excellent resolution, better than 1 part in 10,000. Using the fixed-point multiplication and division subroutines of Sections 14.2 and 14.3 in the next chapter, the temperature is easily computed in either Centigrade or Fahrenheit and with a resolution that fits the application. Figure 13-14 lists the equations to compute the temperature so that each integer increment of the result represents 1 degree of temperature. The alternative equations produce a number wherein each integer increment of the result represents 0.1 degree of temperature. While these high-resolution results are unwarranted in terms of absolute temperature accuracy, they are quite accurate, and appropriate, for *incremental* temperature measurements.

PROBLEMS

13-1 Reading Timer1 What would be the consequence if all reads of the 2 bytes of Timer1 proceeded with a read of **TMR1H** followed immediately by a read of **TMR1L**?

13-2 CyclesToMicrosec subroutine

(a) Being sure to round off to the nearest integer, rewrite the subroutine of Figure 13-4 to implement the algorithm as

$$\text{Microseconds} = (\text{Cycles} \times 4)/10$$

(b) Which subroutine uses fewer instructions?
(c) What is the largest value of cycles that can be handled by each subroutine?

13-3 DecimalDisplay subroutine Rewrite the subroutine of Figure 13-5 as a new **DD1** subroutine that displays **AARGB0:AARGB1:AARGB2** as a decimal number on line 1 of the LCD. This new subroutine and the original, perhaps renamed **DD2**, can be used together to display two variables.

13-4 DisplayMax3 subroutine The **DisplayMax** subroutine of Figure 13-6 displays the 2-byte variable **MAXH:MAXL** in microseconds on the second line of the LCD. Write an expanded version, **DisplayMax3**, that will do the same for **MAXX:MAXH:MAXL**.

13-5 Incrementing TMR1X

(a) Using the low-priority interrupt's polling routine structure of Figure 9-4, show the modification to the polling sequence and create a **TMR1handler** subroutine to increment **TMR1X**.
(b) What is the effect of any *latency* introduced by using this low-priority interrupt to increment **TMR1X**?
(c) Are the **STARTX** and **STOPX** macros of Figure 13-9 still able to read **TMR1X:TMR1H:TMR1L** without error, even in the worst case? Explain.

13-6 CCP1handler subroutine Write a low-priority interrupt handler to form **MAXH:MAXL**, the maximum time interval between repeated rising and falling edges on the CCP1 input pin, as discussed in Section 13.5.

13-7 CCP1 high-priority interrupt service routine

(a) Recast the solution of Problem 13-6 as the sole source of high-priority interrupts.
(b) What is the minimum positive pulse width that can be measured? Explain.
(c) What is the minimum time between the trailing edge of one pulse and the leading edge of the next? Explain.

13-8 Internal time-interval measurements

(a) Compare measuring an internal time interval with the **START** and **STOP** macros of Figure 13-2 with your answer to part *(b)* of the last problem. Explain the difference.
(b) Section 13.6 offers an alternative scheme for measuring an internal time interval. What is the minimum time interval that can be measured in this way? Assume there are no other interrupt sources.

13-9 Extended external time-interval measurement

(a) With an internal 2.5 MHz clock and no prescaling, what is the maximum time interval that can be measured?
(b) With an internal 10 MHz clock and no prescaling, what is the maximum time interval that can be measured? What is the resolution of the measurement in this case?