Timers: Timer0 Tutorial (Part 1)

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Printed on recycled paper.

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a “DS” number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is “DSXXXXXA”, where “XXXXX” is the document number and “A” is the revision level of the document.

INTRODUCTION

This chapter contains general information that will be useful to know before using the Timers Tutorial. Items discussed in this chapter include:

- Document Layout
- The Microchip Web Site
- Customer Support
- Document Revision History

DOCUMENT LAYOUT

This document provides an introduction to Timer0.

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support
- Development Systems Information Line

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://support.microchip.com

DOCUMENT REVISION HISTORY

Revision A (September 2007)

- Initial Release of this Document.

Timers: Timer0 Tutorial (Part 1)

OBJECTIVES

At the end of this lab you should be able to:

1. Describe the main components of the Timer0 peripheral.
2. Configure the Timer0 peripheral registers to produce a working firmware application.

PREREQUISITES

In order to successfully complete this lab you should:

1. Understand basic circuit theory.
2. Understand basic digital electronic components such as gates, multiplexers and memory registers.
3. Understand binary numbering systems and basic binary arithmetic.
4. Have some programming experience in the C Language.
5. Have completed the Introduction to MPLAB® IDE/PICC-LITE™ Compiler Tutorial (DS41322).

EQUIPMENT REQUIRED

This lab has been developed to run completely in MPSIM. However, you will need the following:

1. You will need to download the free MPLAB Integrated Development Environment available at the following url:

   http://www.microchip.com

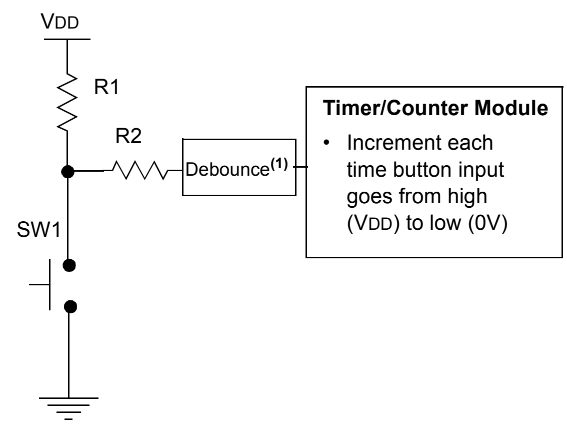
   When prompted, unzip the contents of the file into a temporary folder on your desktop and then install.
2. Download and install the free HI-TECH PICC-LITE™ compiler.
3. Once both programs are installed, complete the Introduction to MPLAB® IDE/PICC-LITE™ Compiler Tutorial (DS41322). This lab assumes that you have done so and will expand on that knowledge.
4. It is also recommended that you download a copy of the PIC16F690 data sheet (DS41262) from www.microchip.com.

INTRODUCTION TO TIMER/COUNTER PERIPHERALS

So what is the difference between a counter and a timer? Both components count events. The distinction actually comes from how this result is used. For example, we could count the number of times that a pushbutton is pressed by connecting it to the input of a Timer/Counter module (see Figure 1-1).

Note that a software or hardware debounce will be required. In this example, the input to the module is tied to VDD (high) and when the pushbutton is pressed it grounds that input (low). The module will increment every time there is a logic level change (i.e., high→low or low→high) and store the result for later reference. Here the module is being used as a counter.

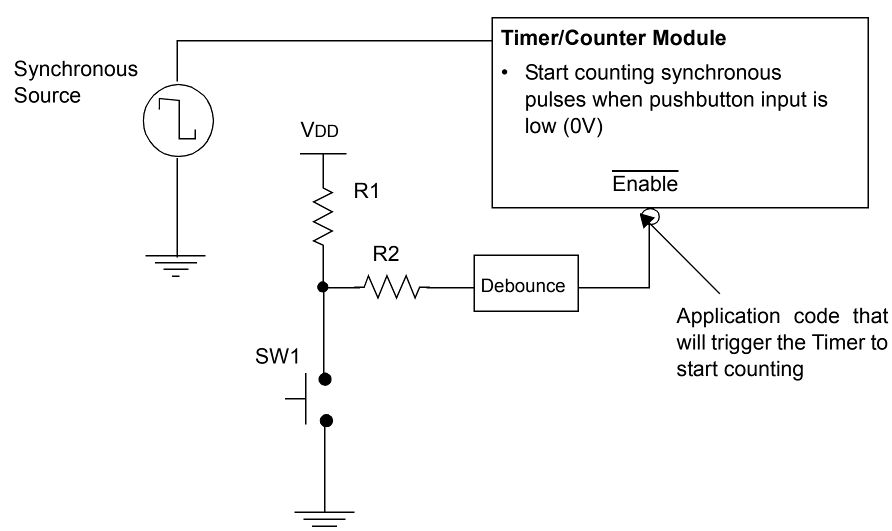
FIGURE 1-1: TIMER/COUNTER USED AS A COUNTER



Note 1: Debounce can be implemented in hardware or software.

Alternately, if we input a periodic signal such as a clock source or oscillator on the input of the module and have it begin incrementing when the pushbutton is pressed and then stop when the pushbutton is released, we could use the value stored to actually calculate the time between button press/release events (see Figure 1-2). If the clock period (1/Frequency) is multiplied by the value (usually in binary) stored, we would know exactly how long the pushbutton was pressed. In this example, the counter is still counting, but this time it is a periodic clock signal that will be used as a reference and the counter is therefore being used as a timer.

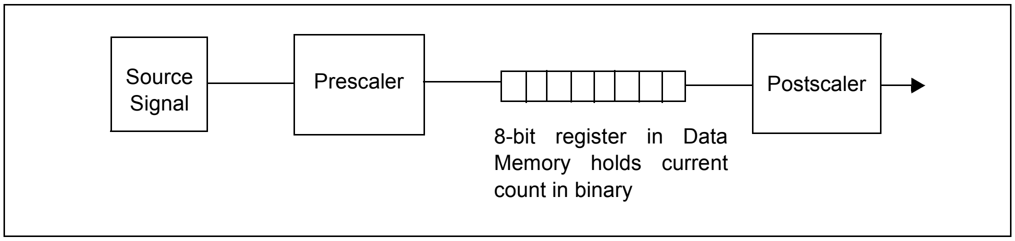
FIGURE 1-2: TIMER/COUNTER USED AS A TIMER



PIC® MID-RANGE MICROCONTROLLERS TIMER MODULES

In the next few labs we will look at three timer/counters that are available in the mid-range PIC microcontroller family. You will notice that each of these timer/counters have unique features but also have some characteristics that are common (see Figure 1-3).

FIGURE 1-3: MAIN COMPONENTS OF A TIMER/COUNTER MODULE



First, a source is required. This could either be a synchronous clock provided by an oscillator or an event that isn't so periodic such as a pushbutton connected to the Timerx Clock Input pin (TxCKI).

Next, the current value will need to be stored somewhere. On the mid-range PIC microcontroller this will be an 8 to 16-bit register located in data memory depending on which of the three Timer/Counter modules is used. A binary value in this register will increment by one depending on a selected edge of a signal transition. For example, if we configured the module to increment on the negative edge of a signal, the value register would increase by one every time a high-to-low transition on the source is encountered.

Finally, each of the three timer/counters has at least one scaler. These scalers may come before (prescaler) or after (postscaler) the timer/counter value register. A scaler works by dividing an input signal by a specific value. For example, if we have a periodic input signal with a frequency of 1000 Hz (1000 Hz = 1/1 mS) and passed it through a prescaler set to 2:1, the value in the timer/counter value register would increment by 1 for every two signal logic transitions or at a frequency of 500 Hz. The signal is slowed down.

TIMER0

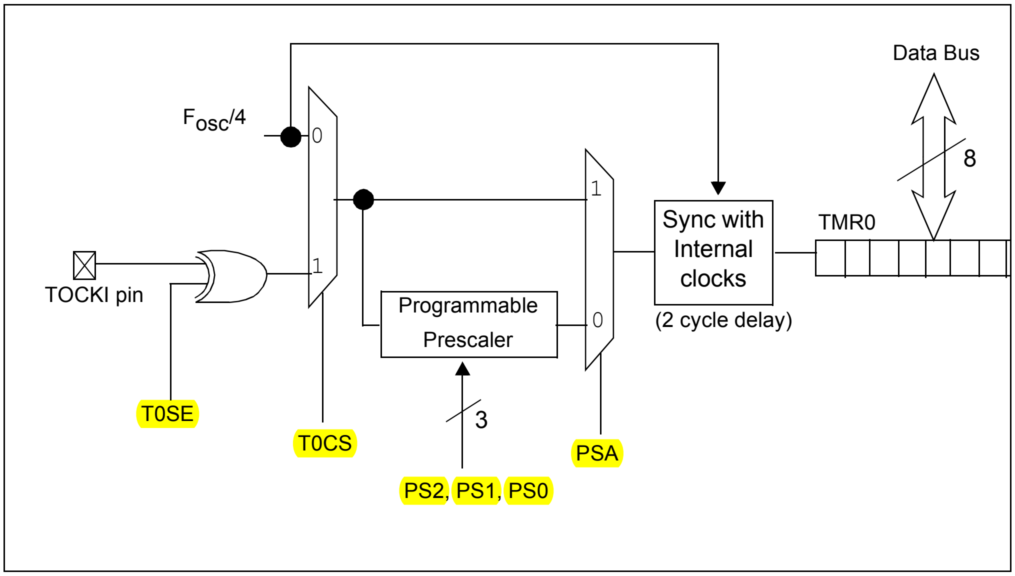
INTRODUCTION

Timer0 is an 8-bit Timer/Counter module with the following features:

1. 8-bit prescaler (shared with WDT).
2. Selectable internal or external clock source.
3. Interrupt on overflow (255$\rightarrow$0).
4. Source edge selection (positive or negative going edge).

To configure the Timer0 module the OPTION_REG Special Function Register (SFR) is used. Figure 1-4 shows the various components that make up the Timer0 module including the bits from the OPTION_REG that affect each.

FIGURE 1-4: SIMPLIFIED DIAGRAM OF TIMER0 MODULE



OPERATION

In this section we will step through the various blocks of the Timer0 module and configure each using the OPTION_REG.

Selecting the Timer0 Source (see Figure 1-5)

T0CS: Timer0 Clock Source Select bit

`1` = Signal present on T0CKI (Timer0 Clock Input) pin used as Timer0 source

`0` = Internal instruction cycle clock used as source

Internal instruction cycle = (Microcontroller oscillator Frequency)/4

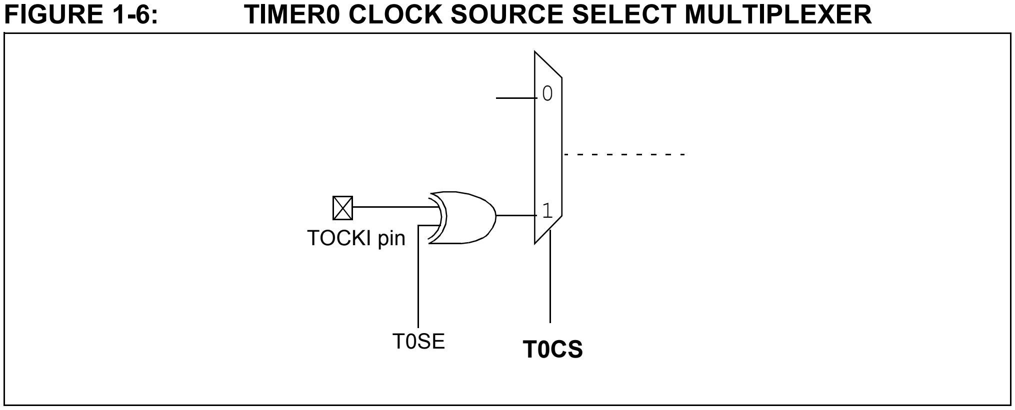
FIGURE 1-5: TIMER0 SOURCE SELECT BIT

OPTION_REG

		T0CS					

The Timer0 source is selected using the Timer0 Clock Source Select bit. Setting or clearing this bit will select one of the channels on the multiplexer (see Figure 1-6).

FIGURE 1-6: TIMER0 CLOCK SOURCE SELECT MULTIPLEXER



SOURCE EDGE SELECTION (EXTERNAL SOURCE ONLY)

T0SE: Timer0 Source Edge Select bit (see Figure 1-7)

`1` = TMR0 register increments on high-to-low transition on T0CKI pin

`0` = TMR0 register increments on low-to-high transition on T0CKI pin

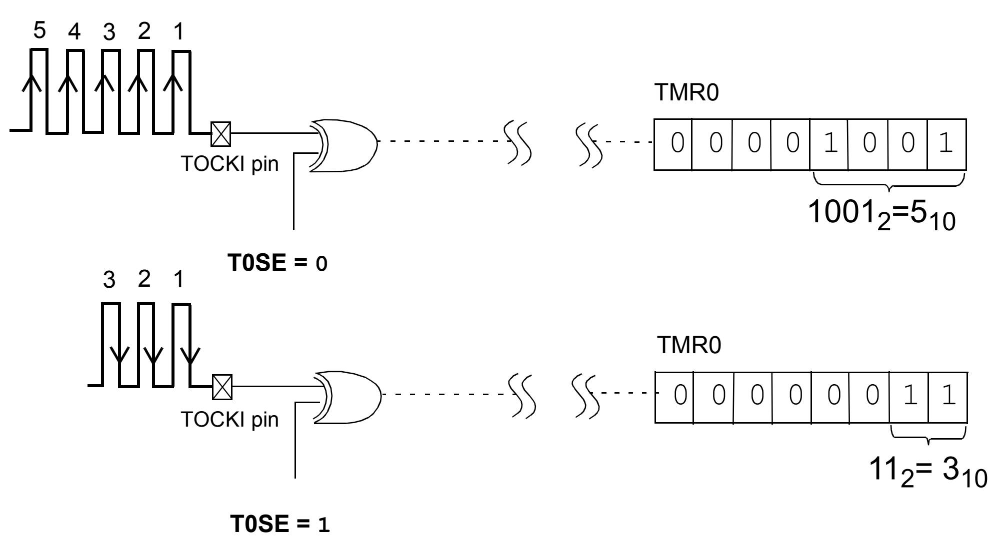
This bit is XOR'd with the logic level on the T0CKI pin.

FIGURE 1-7: TIMER0 SOURCE EDGE SELECT BIT

OPTION_REG

			T0SE				

FIGURE 1-8: EFFECT OF TIMER0 SOURCE EDGE SELECT BIT



PRESCALER ASSIGNMENT AND CONFIGURATION

The software programmable is available for use with either the Timer0 peripheral or another peripheral called the Watchdog Timer, but not both simultaneously. To assign the prescaler to Timer0, the Prescaler Assignment bit needs to be cleared (`0`).

PSA: Prescaler Assignment bit (see Figure 1-9)

`1` = Prescaler is assigned to the Watchdog Timer module

`0` = Prescaler is assigned to Timer0 module

FIGURE 1-9: PRESCALER ASSIGNMENT AND PRESCALER RATE SELECT BITS

OPTION_REG

				***PSA**	**PS2**	**PS1**	**PS0**

***If PSA bit is SET (`1`), Prescaler is assigned to Watchdog Timer and PS2:PS0 have no effect (TMR0 RATE = 1:1)**

The prescaler will determine how many source edges will increment the TMR0 register value by 1. The Timer0 prescaler on the mid-range microcontrollers can be configured to increment the value in TMR0 from a 1:1 ratio to selected source edges or up to a ratio of 1:256 (see Figure 1-10).

PS2:PS0: Prescaler Rate Select bits

FIGURE 1-10: PRESCALER RATE SELECTION

PS2, PS1, PS0	*TMR0 RATE
000	1:2
001	1:4
010	1:8
011	1:16
100	1:32
101	1:64
110	1:128
111	1:256

***If PSA = `1` (Prescaler Assigned to Watchdog Timer), TMR0 Rate = 1:1**

HANDS-ON LAB

Procedure

Part 1: Configuring Timer0

1. Create a new project in MPLAB IDE using the following:
   a) Select the PIC16F690 as the device.
   b) Select HI-TECH PICC-LITE™ as the Language Toolsuite.
   c) Create a folder on your C:\ drive and store the project there.
2. In the MPLAB IDE workspace, create a new file and copy the code in Example 1-1 into it.

EXAMPLE 1-1: HANDS-ON LAB CODE

```
#include    <pic.h>
 //Configure device
           __CONFIG(INTIO & WDTDIS & PWRTDIS & MCLRDIS &
                  UNPROTECT & BORDIS & IESODIS & FCMDIS);

//-----------------------DATA MEMORY

unsigned char counter; //counter variable to count
                      //the number of TMR0 overflows

//----------------------PROGRAM MEMORY

/*------------------------------------------------------------
           Subroutine:  INIT
           Parameters:  none
           Returns:     nothing
           Synopsys:    Initializes all registers
                        associated with the application
------------------------------------------------------------*/
Init(void)
{
           TMR0 = 0;//Clear the TMR0 register


/*Configure Timer0 as follows:

           -   Use the internal instruction clock
           as the source to the module
           -   Assign the Prescaler to the Watchdog
           Timer so that TMR0 increments at a 1:1
           ratio with the internal instruction clock*/

           OPTION = 0B00001000;

}

/*------------------------------------------------------------
           Subroutine:  main
           Parameters:  none
           Returns:     nothing
           Synopsys:    Main program function
-------------------------------------------------------------*/
main(void)
{
           Init(); //Initialize the relevant registers

           while(1)
           {
               //Poll the T0IF flag to see if TMR0 has overflowed

               if (T0IF)
               {
                   ++counter;//if T0IF = 1 increment the counter
                            //variable by 1

                   T0IF = 0;//Clear the T0IF flag so that
                            //the next overflow can be detected
               }
           }


}
```

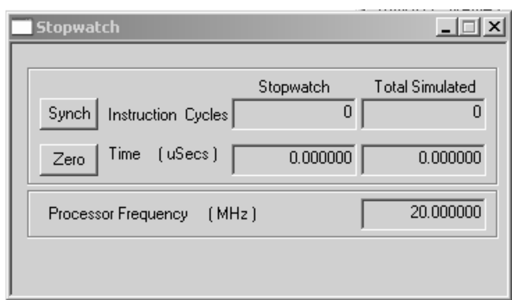
3. Build the project by pressing the Build Project icon . There should be no errors.
4. Select the MPLAB SIM as the debugger.
5. Open a Watch window and add the TMR0, INTCON and OPTION_REG Special Function Registers. Add the counter symbol. Configure the Watch window to allow binary, hexadecimal and decimal values to be seen.
6. Press the Animate icon in the Debugger toolbar and confirm that the following occurs:
   a) On a TMR0 overflow (255 → 0) the T0IF flag is set briefly (INTCON<2>).
   b) When the T0IF flag sets, the counter variable will increment by 1.

Part 2: Timing Analysis

Next, we will check to see that TMR0 is actually incrementing 1:1 with the internal instruction clock.

1. Open the Stopwatch by selecting *Debugger > Stopwatch.*
2. Notice that in the Stopwatch window a specific Processor Frequency has been selected as the default (see Figure 1-11). The Stopwatch will base all timing analysis based off this selected frequency. To change the Processor Frequency select Debugger>Settings and change the Processor Frequency to 8 MHz (max. internal oscillator frequency on the PIC16F690) under the Osc/Trace tab.

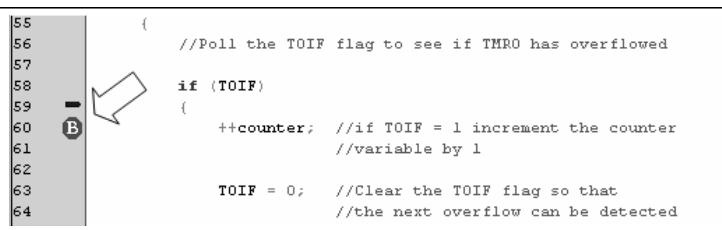
FIGURE 1-11: STOPWATCH WINDOW



The Stopwatch feature can be used to evaluate timing parameters of your firmware application.

3. Setup a break point next to the line in the source code that increments the counter variable (see Figure 1-12).

FIGURE 1-12: SETTING A BREAK POINT



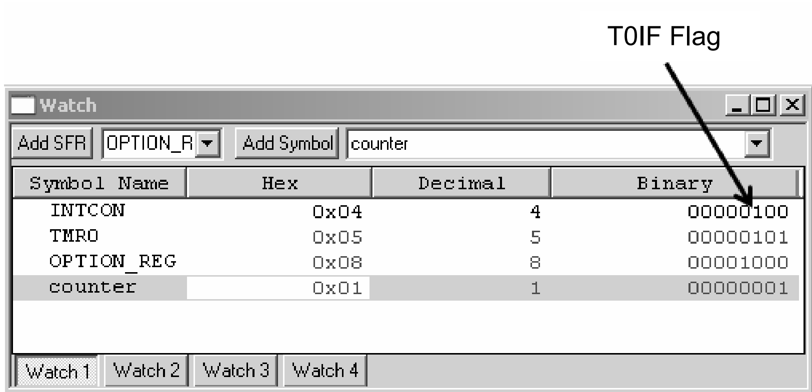
Note:	The specific line number in your code may differ from that shown.

Setting the break point here will allow the Stopwatch to analyze the time interval between successive counter variable increments.

4. Press the Run icon on the Debugger toolbar. This should run, then stop at the break point and the Stopwatch window should now be updated showing the time from Program start to the first instance that the counter variable is to be incremented from zero. Note that the counter symbol in the Watch window has not been incremented yet since that line of code has not been executed.

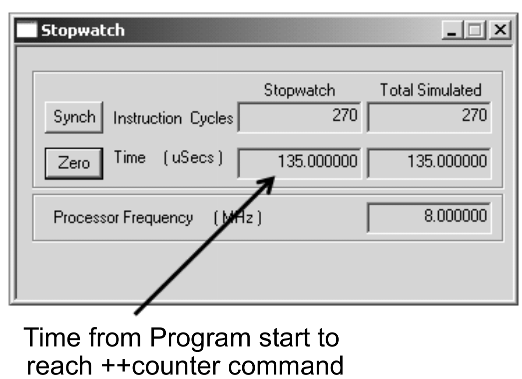
   Press the Step into icon on the Debugger toolbar to execute that line of code. The Watch window should now resemble Figure 1-13.

FIGURE 1-13: THE WATCH WINDOW



The Stopwatch window should resemble Figure 1-14.

FIGURE 1-14: UPDATED STOPWATCH WINDOW



5. Let's do some math to calculate how long it should take to increment the counter variable using a 1:1 TMR0 rate.

EQUATION 1-1: DETERMINING INTERNAL INSTRUCTION CLOCK CYCLE PERIOD

$$\text{Internal instruction cycle} = 1 / [(\text{Processor Frequency}) / 4] = 1 / (8\ \text{MHz} / 4) = 500\text{nS}$$

   Since TMR0 is an 8-bit register it will count from 0 to 255 ($2^8 - 1$) before overflowing and setting the T0IF flag incrementing counter.

   Therefore:

EQUATION 1-2: DETERMINING TIME TO INCREMENT COUNTER VARIABLE

Time to increment counter variable = Internal instruction cycle x 2^8
(we must count the zero) = 500nS x 256 = 128µS

Referring back to the Stopwatch figure, notice it indicates 135 µS.

Press the Zero button in the Stopwatch window to clear. Press the Run icon in the toolbar one more time then the Step into button.

Notice that the timing seems more in keeping with the calculations above?

Why is it that the first time it takes longer to reach the very first counter increment?

Part 3: Writing to TMR0

In this section we will look at the effects of writing to the TMR0 register.

1. Change the main() function in your application code as shown in Example 1-2.

EXAMPLE 1-2: ALTERED MAIN FUNCTION

```
main(void)
{
  Init(); //Initialize the relevant registers
  while(1);  //sit here and wait
}
```

This code will execute Init() and then simply wait at the infinity while loop.

2. Step through the code using the Step into icon on the Debugger toolbar noting that the TMR0 register increments with each instruction clock cycle as shown in the Stopwatch window.
3. Next, change the main() function as shown in Example 1-3.

EXAMPLE 1-3: ADDING A WRITE TO TMR0 REGISTER

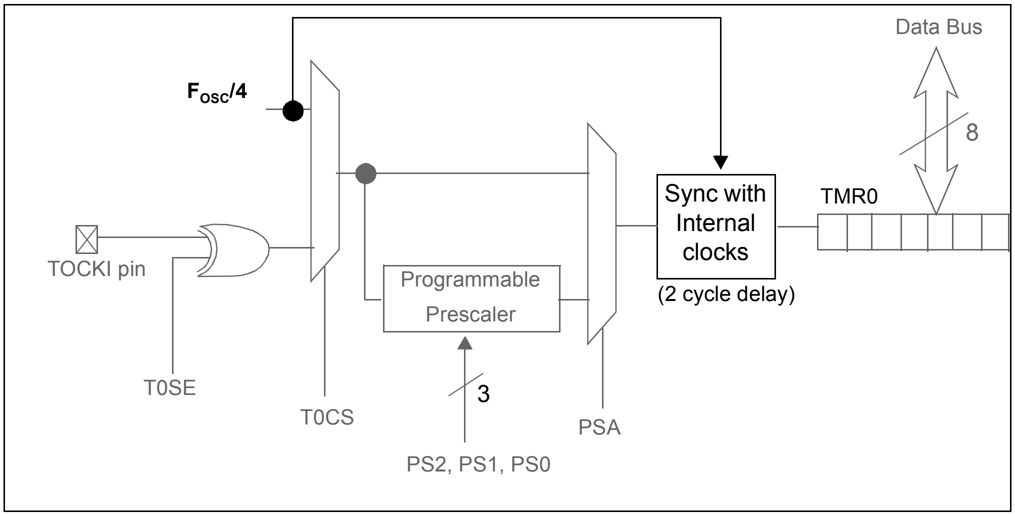
```
main(void)
{
    Init(); //Initialize the relevant registers
    TMR0 = 1; //Write to the TMR0 register
    while(1); //sit here and wait
}
```

4. Step through the code again and notice how many instruction clock cycles after the TMR0 = `1`; command is executed, it takes for the value in TMR0 takes to increment.

It is somewhat difficult to analyze this characteristic when programming in C. Remember that each line of C code could generate multiple lines of Assembly code in the background. However, you should have noticed a delay following the TMR0 = `1`; instruction before TMR0 incremented.

Referring back to the Timer0 block diagram, note that the signal entering Timer0 requires two instruction clock cycles to synchronize TMR0 register. This ensures an accurate, synchronized count (see Figure 1-15).

FIGURE 1-15: SIMPLIFIED TIMER0 BOCK DIAGRAM



In conclusion, if you are writing to the TMR0 register your code will need to accommodate this synchronization process by offsetting the value to be written. For more information on this topic refer to the PIC16F690 data sheet (DS41262).

EXERCISES

Return the main() to the code shown in Example 1-4 below and try exercises 1 through 4.

EXAMPLE 1-4: ORIGINAL MAIN FUNCTION CODE

```
main(void)
{
  Init(); //Initialize the relevant registers
  while(1)
  {
      //Poll the T0IF flag to see if TMR0 has overflowed

      if (T0IF)
      {
            ++counter;//if T0IF = 1 increment the counter
            //variable by 1

            T0IF = 0;//Clear the T0IF flag so that
            //the next overflow can be detected
      }
}
```

1. Configure Timer0 as follows:
   - Use the internal instruction clock as the source for the module.
   - Assign the prescaler to Timer0 and Clear the Prescaler Rate Select bits to zero.
2. How long does it take to increment the counter variable?
3. Modify the equation developed earlier to accommodate for the prescaler.
4. Configure Timer0 to increment the counter variable every 4 mS (give or take a few microseconds). We will implement a more efficient Timer in subsequent labs.

NEXT TIME

In the next lab, Introduction to Timer0 (Part 2), we will look at using an external source and implement interrupts.

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

09/10/07