Experiment #10 (HW#5)
Introduction to Timer0 Module (timer & counter)

1. Prerequisites

ENCS 538, C programming language, PICC compiler, Microchip PIC16F877A datasheet.

2. Objectives

- Understanding how to use timers with interrupts to obtain the needed resolution (i.e. 10 ms).
- Building and programming embedded systems that use time in their functionalities (i.e. Counters Applications).

3. Background

The timers of the PIC16F887 microcontroller can be briefly described in only one sentence. There are three completely independent timers/counters marked as TMR0, TMR1 and TMR2. However, in this experiment we would like to focus on Timer0 only since the Timer0 can be used to build the target application of this experiment. The target application of this experiment is to build BCD counters to count from 00 to 99 and then display the value of counter on seven segment display. The counter is incremented by 1 each second, therefore, the Timer0 will be used to define a one second resolution using ISR mechanism.

3.1 Timer0 module (TMR0)

The timer TMR0 has a wide range of applications in practice. Very few programs don't use it in some way. It is very convenient and easy to use for writing programs or subroutines for generating pulses of arbitrary duration, time measurement or counting external pulses (events) with almost no limitations.

The timer TMR0 module is an 8-bit timer/counter with the following features:

- 8-bit timer/counter.
- 8-bit pre-scaler (shared with Watchdog timer).
- Programmable internal or external clock source.
- Interrupt on overflow.
- Programmable external clock edge selection.

If the periodic interrupt occurs every 10ms, the ISR will have a system tick period of 10ms, or a rate of 100Hz. With a tick rate defined, you can specify delays to a resolution of one timer tick period, e.g. delays of 10ms, 20ms, 1s, 2s, are possible.
The resolution of the tick rate is calculated using the following equation, where F_{osc} is the input clock frequency, Period the resolution of the tick rate, Pre-scaler is configured in Timer0 control register, and TMR0 is the value of Timer0 that should be loaded to achieve the desired tick rate.

$$Period = (256 - TMR0) * \frac{4}{F_{osc}} * Prescaler$$

Example: Suppose that the desired value of the tick rate is 10ms, the input clock frequency (F_{osc}) is 4MHz, and the pre-scaler is 256. However, it is important to mention that the potential values of pre-scaler are (1, 2, 8, 16, 32, 64, 128, and 256) and it is configured in the control register for Timer0. Now, by applying the given inputs on the above equation, the value of TMR0 is 217.

$$10 * 10^{-3} = (256 - TMR0) * \frac{4}{4 * 10^{6}} * 256 \quad \rightarrow \quad TMR0 = 256 - 39 = 217$$

Figure 1 represents the timer TMR0 schematic with all bits which determine its operation. These bits are stored in the OPTION_REG Register.

Notes: Watchdog timer must be disabled when using the internal or external frequencies. Internal frequency = external frequency / 4

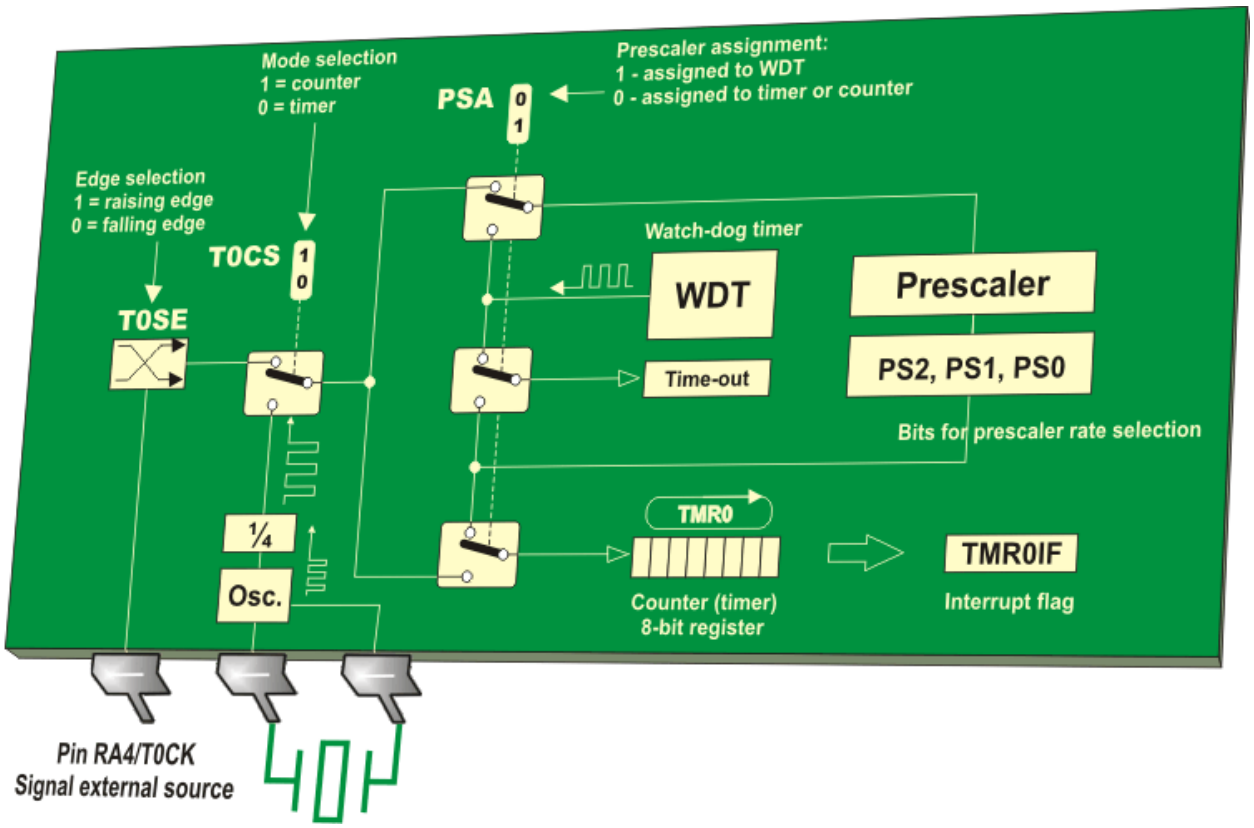


Figure 1: TMR0 Module

OPTION_REG Register

	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Features
OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- **RBPU - PORTB Pull-up enable bit**
  - 1 - PORTB pull-up resistors are disabled; and
  - 0 - PORTB pins can be connected to pull-up resistors.

- **INTEDG - Interrupt Edge Select bit**
  - 1 - Interrupt on rising edge of INT pin (0-1); and
  - 0 - Interrupt on falling edge of INT pin (1-0).

- **T0CS - TMR0 Clock Select bit**
  - 1 - Pulses are brought to TMR0 timer/counter input through the RA4 pin; and
  - 0 - Internal cycle clock (Fosc/4).

- **T0SE - TMR0 Source Edge Select bit**
  - 1 - Increment on high-to-low transition on TMR0 pin; and
  - 0 - Increment on low-to-high transition on TMR0 pin.

- **PSA – Pre-scaler Assignment bit**
  - 1 – Pre-scaler is assigned to the WDT; and
  - 0 – Pre-scaler is assigned to the TMR0 timer/counter.

- **PS2, PS1, PS0 – Pre-scaler Rate Select bit**
  - Pre-scaler rate is adjusted by combining these bits as seen in the table 4-1, the same combination of bits gives different pre-scaler rate for the timer/counter and watch-dog timer respectively.

PS2	PS1	PS0	TMR0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

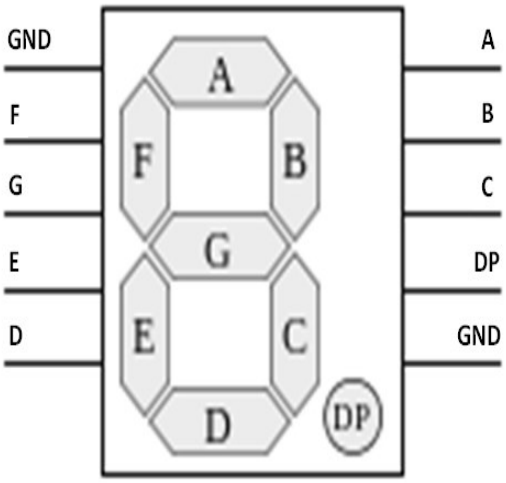
Pre-scaler Rates

3.2 The seven segment display IC

A seven-segment display (SSD), or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot-matrix displays.

Seven-segment displays are widely used in digital clocks, electronic meters, and other electronic devices for displaying numerical information.

Hexadecimal digits can be displayed on seven-segment displays. A particular combination of uppercase and lowercase letters are used for A–F; this is done to obtain a unique, unambiguous shape for each letter (otherwise, a capital D would look identical to a 0 and a capital B would look identical to an 8). Also the digit 6 must be displayed with the top bar lit to avoid ambiguity with the letter b)



Hexadecimal encodings for displaying the digits 0 to F

Digit	gfedcba	a	b	c	d	e	f	g
0	0×3F	**on**	**on**	**on**	**on**	**on**	**on**	off
1	0×06	off	**on**	**on**	off	off	off	off
2	0×5B	**on**	**on**	off	**on**	**on**	off	**on**
3	0×4F	**on**	**on**	**on**	**on**	off	off	**on**
4	0×66	off	**on**	**on**	off	off	**on**	**on**
5	0×6D	**on**	off	**on**	**on**	off	**on**	**on**
6	0×7D	**on**	off	**on**	**on**	**on**	**on**	**on**
7	0×07	**on**	**on**	**on**	off	off	off	off
8	0×7F	**on**	**on**	**on**	**on**	**on**	**on**	**on**
9	0×6F	**on**	**on**	**on**	**on**	off	**on**	**on**
A	0×77	**on**	**on**	**on**	off	**on**	**on**	**on**
b	0×7C	off	off	**on**	**on**	**on**	**on**	**on**
C	0×39	**on**	off	off	**on**	**on**	**on**	off
d	0×5E	off	**on**	**on**	**on**	**on**	off	**on**
E	0×79	**on**	off	off	**on**	**on**	**on**	**on**
F	0×71	**on**	off	off	off	**on**	**on**	**on**

4. Procedure

4.1 Overview

In this experiment, we're going to use the PIC16F877A MCU to implement a 2-digit counter that counts from 00 to 99 and repeats. The outputs that you shall get are two 7segment displays, one that counts every 1 second, and the other counts every 10 seconds.

The flow of the code can be summarized as (for more clarification, check the code's comments):

- Define 2 variables to control the value displayed on each 7-segment.
- Set the OPTION_REG to values that fit into equation parameters needed to obtain a 0.01 second interrupts.
- Also, adjust the pullup resistors enabling bit to enable connecting them on the specified output ports.
- Within the routine that gets invoked on every interrupt that happens upon TMR0 overflow (programmed to be each 0.01 seconds), reset the TMR0 register to 217 and reset the timer0 interrupt flag to obtain another 0.01; increase each digit counter.
- Start an endless loop that gets interrupted on every TMR0 overflow, where it increases the number on the first display when the its counter reaches 100 (that is 100*0.01=1sec); and increases the number on the second display when the its counter reaches 1000 (that is 1000*0.01=10sec).

The pin-to-pin connections that you are going to apply are according to the following table:

PORT D	1st Seven Segment	PORT B	2nd Seven Segment
D0	a	B0	a
D1	b	B1	b
D2	c	B2	c
D3	d	B3	d
D4	e	B4	e
D5	g	B5	g
D6	f	B6	f
D7	DP	B7	DP

4.2 The implementation

1. Open **MPLAB IDE**.
2. Make new project, and add the codes **7seg_main.c** provided in the experiment codes section.
3. Compile the project using **HI-TECH UNVERSAL ANSI COMPILER** for **PIC16F877A**.
4. Open Proteus ISIS.
5. Under components, list the following: **PIC16F877A**, **7SEG-COM-CAT-GRN, RES**, **CRYSTAL** and **CAP**.

6. Under terminal mode: use the **POWER** and **GND**.
7. Implement circuit provided in Figure 2 (set pullup resistors to 330Ohms).
8. Load the generated **HEX** onto the micro-controller. Set its external frequency to **4MHz**.
9. Click on **"play"**.
10. Download the program on the **PIC16F877A** using **PicKit3**.
11. Apply the connections on hardware.
12. Observe the outputs you get and compare them to the simulated ones.

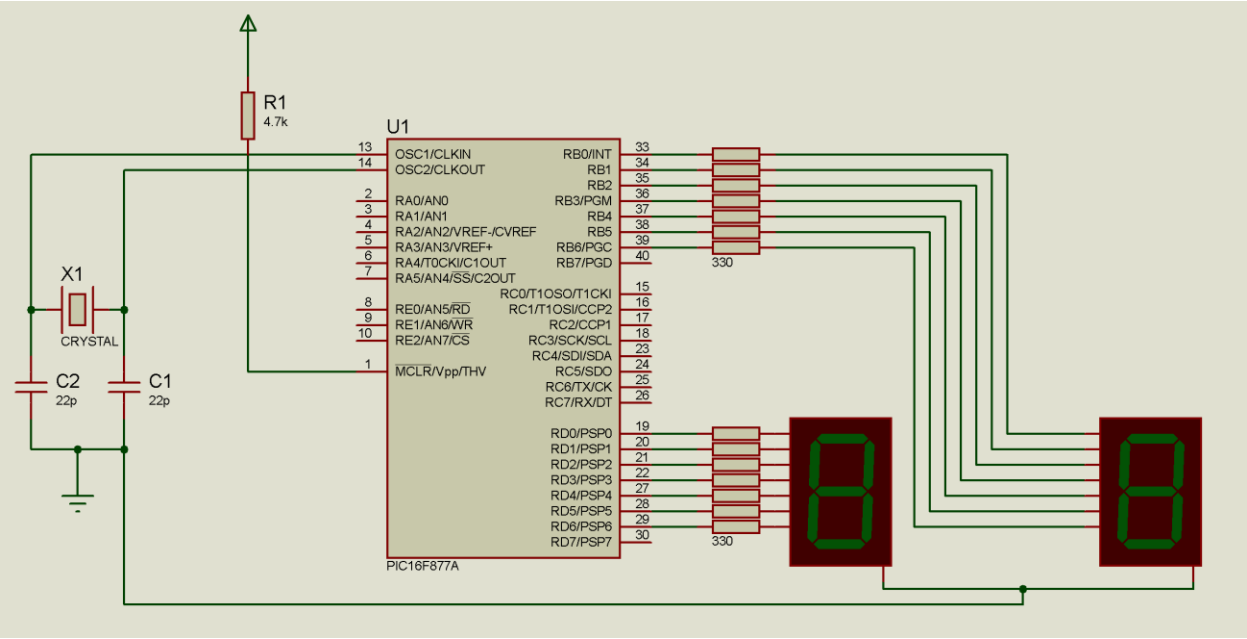


Figure 2: Experiment schematic diagram

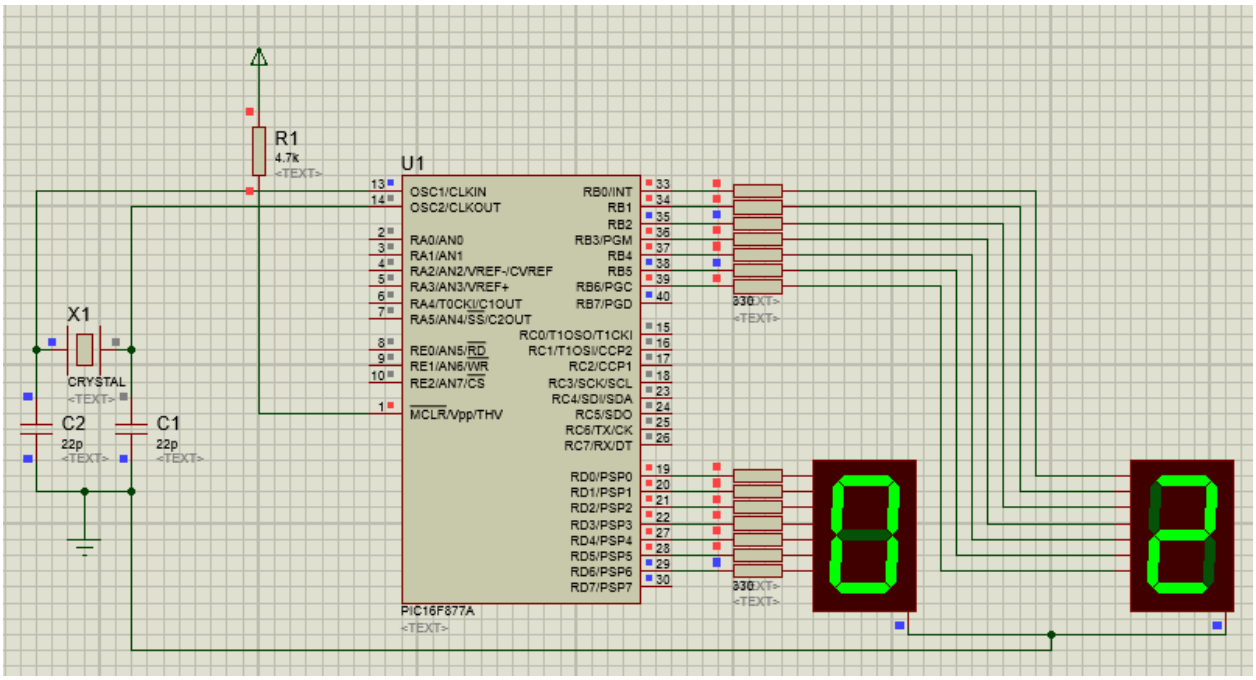


Figure 3: Experiment outputs

4. Experiment Codes

7seg_main.c

```
#include <pic.h>

__CONFIG(DEBUG_OFF & WDTE_OFF & LVP_OFF & FOSC_HS & BOREN_ON);

char display_digit[10]= {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
// binary values to be used in representing decimal Number from 0 to 0
int d1Counter=0; // used to trigger d2 on every second
int d2Counter=0; // used to trigger d2 on every 10 seconds
int d1=0; // 7-segment display1 index
int d2=0; // 7-segment display2 index

void InitMain() {
  PORTB = 0; // Set PORTB to 0
  PORTD = 0; // Set PORTD to 0
  TRISB = 0; // PORTB is output
  TRISD= 0; // PORTD is output
}

void main() {

 InitMain();

 GIE=1; // Enable Global Interrupt
 T0IE = 1; // Timer0 Interrput Enable

 // Prescaler Rate Select bit , 111 means multiply Timer0 by 256
      PSA=0;//Prescaler is assigned to the TMR0 timer/counter.
 TMR0 =217;
 OPTION_REG = 0x47;
 PORTB= display_digit[d1];
 PORTD= display_digit[d2];
 while (1) {  // Endless loop
      if(d1Counter==100){ // we achieved 1 second
                  d1++; // increase d1 on every second
                  if(d1==10) d1=0;
                  PORTB= display_digit[d1];
                  d1Counter=0; // reset for the next second
      } // end of 1 second if-statement
      if(d2Counter==1000){ // we achieved 10 seconds
                  d2++; // increase d1 on every 10 seconds
                  if(d2==10) d2=0;
                  PORTD= display_digit[d2];
                  d2Counter=0; // reset for the next 10 seconds
      } // end of 10 seconds if-statement
 } // end of while(1)
} // end of main

void interrupt IntVector( void ){ // invoked on every TMR0 int (happens
every 0.01 seconds)
 if (T0IE && T0IF) { // check if the Timer0 is overflowed and Timer
Interrupt is enabled
       T0IF = 0; // set Timer Overflow flag to zero for the next interrupt
```

```
        TMR0 =217;  /* reload the time value. It is calcualted based on this
equation Period = (256 - TMR0)*(4/fosc)*(Prescaler)
        in our case a 10ms period (resolution) should be achieved by set
Prescaler to 256 and the used Fsoc=4MHz
        0.01 = (256 -TMR0)*(4/4*10^6)*256 -----> TMR0= 256-39=217 */
        d1Counter++; // increment d1Counter
        d2Counter++; // increment d2Counter
 }
}
```