Practical Activity # 4
Timers (Timer1, Timer2 and CCP Module)

Exercise Objectives:
1. Configure Timer1, Timer2 and Capture, Compare and PWM
2. Use timers and CCP module to generate delays, digital pulses and pulse-width modulation (PWM)
3. Applying timers in basic embedded systems application

Student Outcomes:
At the end of the exercise, students must be able to:
1. learn how to configure the timers and CCP module in PIC16F877A
2. able to use timers for delays, pulse generation and modulation
3. develop an embedded systems application utilizing timers

Tools Required:
The following will be provided for the students:
1. MPLAB IDE v8.92 + MPLAB XC8 v1.33
2. Proteus v8.11

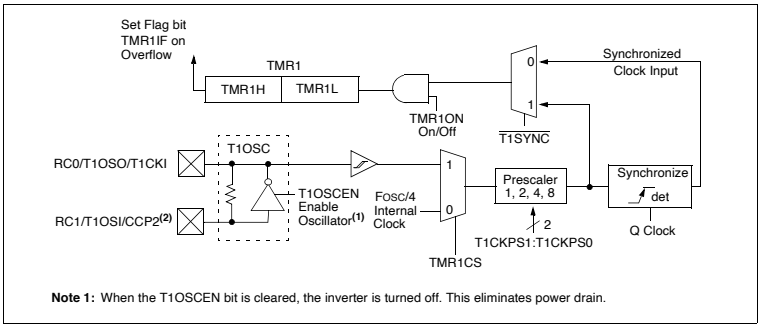
Delivery Process:
The instructor will conduct a briefing for this practical activity. Notes from the lecture class will provided for reference purposes. All inquiries pertaining to the latter must be referred to this manual.

Note:

Part I. Timer1 Module

It's a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. TMR1 interrupt is generated on overflow, if enabled. It has two modes operation: Timer or Counter. Timer mode increments every instruction cycle while counter mode increments from an external clock input. Timer1 can be enabled/disabled by setting or clearing control bit, TMR1ON (T1CON<0>). Timer1 overflow interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

FIGURE 6-2: TIMER1 BLOCK DIAGRAM



Overflow Timeout

Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is Fosc/4.

The timeout can be calculated given the oscillator frequency (Fosc) of the MCU and a prescaler value.

$$timeout = \frac{1}{\frac{F_{OSC}}{4}} \; x \; prescaler \; x \; Timer \; Max \; Count \qquad (1)$$

Timer1 has a selectable prescaler values 1:1, 1:2, 1:4 and 1:8. Since Timer1 is 16 bits wide, the Timer Max Count is 2^{16} or 65536_{10}.

Timer1 Exercise

Configure the Timer1 to generate an interrupt every 0.5 seconds. Create an ISR for Timer1 overflow interrupt where it blinks the LED connected at RA0. Assume Fosc = 4MHz and the timer operate in Timer Mode.

Using a prescaler of 1:8 and since the timeout is given at 0.5s:

From Equation 1: $$0.5\,s = \frac{1}{\frac{4MHz}{4}} \; x \; 8 \; x \; Timer \; Max \; Count$$

Timer Max Count must be determined in order to have a timeout of 0.5s:

$$Timer \; Max \; Count = 62500$$

The value of Timer Max Count will be used to reload the timer after interrupt. Below is the `main()` with the Timer1 configuration and the interrupt service routine for Timer1 overflow interrupt.

```
void main(void)
{
    ADCON1 = 0x06;   // set all pins in PORTA as digital I/O
    TRISA = 0x00;    // sets all of PORTA to output
    RA0 = 0;         // initialize RA0 to 0 (LED off)
    T1CON = 0x30;    // 1:8 prescaler, internal clock, Timer1 off
    TMR1IE = 1;      // enable Timer1 overflow interrupt (PIE1 reg)
    TMR1IF = 0;      // reset interrupt flag (PIR1 reg)
    PEIE = 1;        // enable all peripheral interrupt (INTCON reg)
    GIE = 1;         // enable all unmasked interrupts (INTCON reg)
    TMR1 = 0x0BDC;   // counter starts counting at 0x0BDC (3036)
    TMR1ON = 1;      // Turns on Timer1 (T1CON reg)

  for(;;)            // foreground routine
  {

  }
}
```

The value of 0x0BDC (3036) was derived from 65536-62500. By starting the counter at this value, the 0.5s timeout can be achieved.

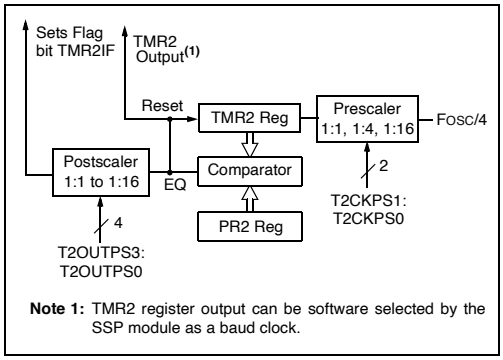
```
void interrupt ISR(void)
{
    GIE = 0;              // disable all unmasked interrupts (INTCON reg)

  if(TMR1IF==1)           // checks Timer1 interrupt flag
  {
      TMR1IF = 0;         // clears interrupt flag
      TMR1 = 0x0BDC;      // timer will start counting at 0x0BDC (3036)

      RA0 = RA0^1;        // toggles the LED at RA0
  }

  GIE = 1;                // enable all unmasked interrupts (INTCON reg)
}
```

Open Proteus and construct the required circuit with filename “LE4-1.pdsprj”. In MPLAB, create a new project using the project wizard with the name “CpE 3201-LE4”. Create a new source file with the filename “LE4-1.c” and add it to the project. Write the source code with the pre-processor directives. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE4-1.pdsprj”) by loading the generated .hex file to the microcontroller. Observe the output.

Part II. Timer2 Module

It is an 8-bit timer with a prescaler and a postscaler. It can be used as the PWM time base for the PWM mode of the CCP module(s). TMR2 register is readable and writable and is cleared on any device reset. It has an 8-bit period register, PR2. If TMR2=PR2 (match) it will generate an interrupt if enabled. a TMR2 interrupt. Timer2 increments from 00h until it matches PR2 and
then resets to 00h on the next increment cycle. Unlike Timer0 and Timer1, this timer module does not generate interrupt upon overflow.

FIGURE 7-1: TIMER2 BLOCK DIAGRAM



Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

Timer2 increments every clock cycle (Fosc/4). The timeout can be calculated given the oscillator frequency (Fosc) of the MCU and a prescaler value similar to Equation 1. The *Timer Max Count* is 2^8 or 256_{10}.

Timer2 Exercise

Generate a pulse-wave with a frequency of 1000Hz at 50% duty cycle. The output pulse-wave will be at RA0. Assume Fosc = 4MHz.

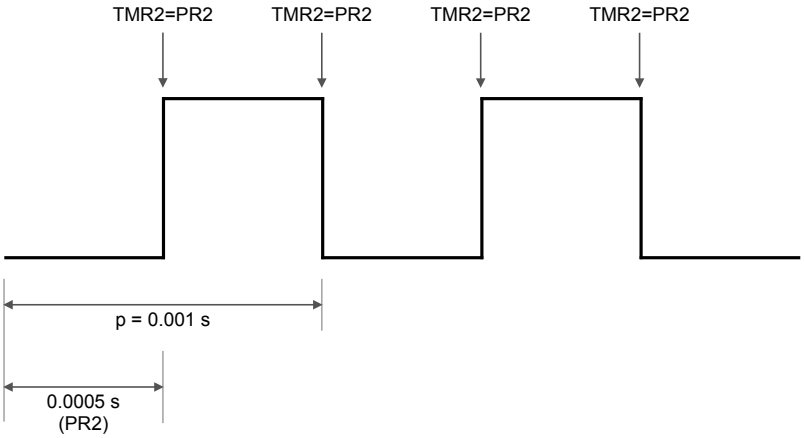


Fig. 1. TMR2 and PR2 Match

Every time TMR2 would be equal to PR2, interrupt is generated and the output of RA0 is toggled. Determining the period given the frequency of 1000Hz:

$$p = \frac{1}{f} = \frac{1}{1000} = 0.001\ s$$

Using a prescaler of 1:4 for Timer2, we can calculate the *Timer Max Count* at half-cycle:

From Equation 1:

$$0.0005\ s = \frac{1}{\frac{4Mhz}{4}}\ x\ 4\ x\ Timer\ Max\ Count$$

$$Timer\ Max\ Count = 125$$

Therefore, the value of PR2 must be set to 125 to match it to TMR2. When TMR2=PR2, the timeout for the half cycle is achieved then an interrupt can be generated. Below is the *main()* with the Timer2 configuration and the interrupt service routine for TMR2/PR2 match interrupt.

```
void main(void)
{
   ADCON1 = 0x06;   // set all pins in PORTA as digital I/O
   TRISA = 0x00;    // sets all of PORTA to output
   RA0 = 0;         // initialize RA0 to 0
   T2CON = 0x01;    // 1:4 prescaler, Timer2 off
   TMR2IE = 1;      // enable Timer2/PR2 match interrupt (PIE1 reg)
   TMR2IF = 0;      // reset interrupt flag (PIR1 reg)
   PEIE = 1;        // enable all peripheral interrupt (INTCON reg)
   GIE = 1;         // enable all unmasked interrupts (INTCON reg)
   PR2 = 0x7D;      // match value for TMR2(125)at half cycle
   TMR2ON = 1;      // Turns on Timer2 (T2CON reg)

  for(;;)           // foreground routine
  {


  }
}
```

The value of 0x7D (125) was derived from Timer Max Count at half-cycle timeout for the frequency of 1000Hz.

```
void interrupt ISR(void)
{
   GIE = 0;         // disable all unmasked interrupts (INTCON reg)
   (TMR2IF==1)      // checks Timer2 interrupt flag (TMR2=PR2)
   {
      TMR2IF = 0;   // clears interrupt flag
      RA0 = RA0^1;  // toggles the output signal at RA0
   }
   GIE = 1;         // enable all unmasked interrupts (INTCON reg)
}
```

Open Proteus and construct the required circuit with filename “LE4-2.pdsprj”. In MPLAB, create a new source file with the filename “LE4-2.c” and add it to the project “CpE 3201-LE4”. Write the source code above with the pre-processor directives. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE4-2.pdsprj”) by loading the generated .hex file to the microcontroller. Connect a virtual oscilloscope to RA0 and observe the output signal. Verify the frequency of the signal.

Part III. CCP Module

The PIC16F877A has two (2) Capture/Compare/PWM (CCP). Each module contains a 16-bit register which can operate as a 16-bit Capture register, 16-bit Compare register and PWM Master/ Slave Duty Cycle register.

Both the CCP1 and CCP2 modules are identical in operation, with the exception being the operation of the special event trigger.

The CCP1 Register (CCPR1) is comprised of two 8-bit registers: CCPR1L (low byte) and CCPR1H (high byte). The CCP1CON register controls the operation of CCP1. The special event trigger is generated by a compare match and will reset Timer1.

TABLE 8-1: CCP MODE – TIMER RESOURCES REQUIRED

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

TABLE 8-2: INTERACTION OF TWO CCP MODULES

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same TMR1 time base
Capture	Compare	The compare should be configured for the special event trigger which clears TMR1
Compare	Compare	The compare(s) should be configured for the special event trigger which clears TMR1
PWM	PWM	The PWMs will have the same frequency and update rate (TMR2 interrupt)
PWM	Capture	None
PWM	Compare	None

The CCP2 Register (CCPR2) is comprised of two 8-bit registers: CCPR2L (low byte) and CCPR2H (high byte). The CCP2CON register controls the operation of CCP2. The special event trigger is generated by a compare match and will reset Timer1 and start an A/D conversion (if the A/D module is enabled).

Capture Module

In Capture mode, CCPR1H:CCPR1L captures the 16-bit value of the TMR1 register when an event occurs on pin RC2/CCP1. An event is defined as one of the following: every falling edge, every rising edge, every 4th rising edge and every 16th rising edge. The type of event is configured by control bits, CCP1M3:CCP1M0 (CCPxCON<3:0>). The event input pin for the capture module is RC2 (PORTC<2>).

Capture Module Exercise

Determine the period of a pulse-wave signal input at RC2. Assume Fosc = 4Mhz. This exercise will determine the period by capturing the input signal at every rising edge.

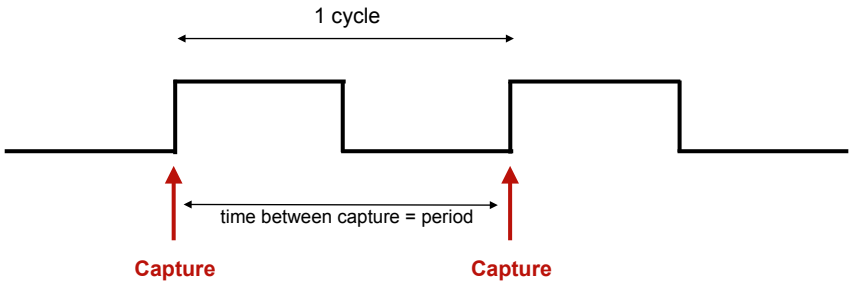


Fig. 2. Capture at Every Rising Edge

Using a prescaler of 1:8 for Timer1, we can calculate the time per increment:

From Equation 1*:

$$timeout = \frac{1}{\frac{4MHz}{4}} \; x \; 8 \; x \; 1$$

$$timeout = 8x10^{-6} \; s$$

** Timer Max Count is set to 1 since we need to determine the timeout per Timer1 increment.*

Normalizing the timeout to make it a whole number:

$$timeout_n = 8x10^{-6}\ s\ x\ 1x10^{6} = 8\ s$$

The normalization is used to eliminate working with real (floating point) numbers. Below is the *main()* with the CCP module configuration and the interrupt service routine for capture event interrupt.

```
void main(void)
{
    TRISC = 0x04;      // set RC2 to input
    T1CON = 0x30;      // 1:8 prescaler, Timer1 off
    CCP1CON = 0x05     // capture mode: every rising edge
    CCP1IE = 1;        // enable TMR1/CCP1 match interrupt (PIE1 reg)
    CCP1IF = 0;        // reset interrupt flag (PIR1 reg)
    PEIE = 1;          // enable all peripheral interrupt (INTCON reg)
    GIE = 1;           // enable all unmasked interrupts (INTCON reg)
    TMR1ON = 1;        // Turns on Timer1 (T1CON reg)

  for(;;)              // foreground routine
  {


  }
}
```

CCP1CON must be set to capture mode which generates an interrupt when a rising edge event at RC2 occurs.

```
void interrupt ISR(void)
{
    int period = 0;

    GIE = 0;              // disable all unmasked interrupts (INTCON reg)

    if(CCP1IF==1)         // checks CCP1 interrupt flag
    {
       CCP1IF = 0;              // clears interrupt flag
       TMR1 = 0                 // resets TMR1

       period = CCPR1/1000;     // transfers captured TMR1 value
                                // normalize the value (make the number smaller)
       period = period*8;       // multiply by the normalized TMR1 timeout
    }

    GIE = 1;              // enable all unmasked interrupts (INTCON reg)
}
```

In Proteus, construct the required circuit with filename “LE4-3.pdsprj”. In MPLAB, create a new source file with the filename “LE4-3.c” and add it to the project “CpE 3201-LE4”. Write the source code above with the pre-processor directives. Build the program and debug if necessary.

Compare Module

In Compare mode, the 16-bit CCPR1 register value is constantly compared against the TMR1 register pair value. When a match occurs, the RC2/CCP1 pin is: driven high, driven low or remains unchanged. The action on the pin is based on the value of control bits, CCP1M3:CCP1M0 (CCP1CON<3:0>). At the same time, interrupt flag bit CCP1IF is set.

Compare Module Exercise

Generate a pulse-wave with a frequency of 100Hz at 50% duty cycle. The output pulse-wave will be at RA0. Assume Fosc = 4MHz.

Determining the period given the frequency of 1000Hz:

$$p = \frac{1}{f} = \frac{1}{100} = 0.01\ s$$

Using a prescaler of 1:4 for Timer1, we can calculate the Timer Max Count at half-cycle:

From Equation 1: $$0.005\ s = \frac{1}{\frac{4MHz}{4}}\ x\ 4\ x\ Timer\ Max\ Count$$

$$Timer\ Max\ Count = 1250\ or\ 0x4E2$$

Therefore, CCPR1 must be set to 0x4E2. When TMR1=CCPR1, then generate an interrupt and toggle the output at RA0. Below is the *main()* with the CCP module configuration and the interrupt service routine for TMR1/CCP1 match interrupt.

```
void main(void)
{
   ADCON1 = 0x06;    // set all pins in PORTA as digital I/O
   TRISA = 0x00;     // sets all of PORTA to output
   RA0 = 0;          // initialize RA0 to 0
   T1CON = 0x20;     // 1:4 prescaler, Timer1 off
   CCP1CON = 0x0A    // compare mode: generate interrupt on match
   CCP1IE = 1;       // enable TMR1/CCP1 match interrupt (PIE1 reg)
   CCP1IF = 0;       // reset interrupt flag (PIR1 reg)
   CCPR1 = 0x4E2     // set the match value to TMR1
   PEIE = 1;         // enable all peripheral interrupt (INTCON reg)
   GIE = 1;          // enable all unmasked interrupts (INTCON reg)
   TMR1ON = 1;       // Turns on Timer2 (T1CON reg)

  for(;;)            // foreground routine
  {


  }
}
```

CCP1CON must be set to compare mode which generates an interrupt when TMR1=CCPR1. This example does not use the CCP1 output at RC2.

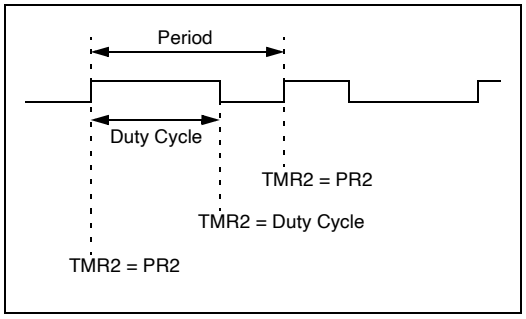
```
void interrupt ISR(void)
{
   GIE = 0;             // disable all unmasked interrupts (INTCON reg)
   if(CCP1IF==1)        // checks CCP1 interrupt flag
   {
       CCP1IF = 0;      // clears interrupt flag
       RA0 = RA0^1;     // toggles the output signal at RA0
   }
   GIE = 1;             // enable all unmasked interrupts (INTCON reg)
}
```

Open the Proteus firmware project “LE4-2.pdsprj”. In MPLAB, create a new source file with the filename “LE4-4.c” and add it to the project “CpE 3201-LE4”. Write the source code above with the pre-processor directives. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE4-2.pdsprj”) by loading the generated .hex file to the microcontroller. Connect a virtual oscilloscope to RA0 and observe the output signal. Verify the frequency of the signal.

PWM Module

In Pulse Width Modulation (PWM) mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCP1 pin is multiplexed with the PORTC data latch, the TRISC<2> bit must be cleared to make the CCP1 pin an output.

FIGURE 8-4: PWM OUTPUT



PWM Period

The PWM period is specified by writing to the PR2 register. The PWM period can be calculated using the following formula:

$$\text{PWM Period} = [(PR2) + 1] \bullet 4 \bullet T_{OSC} \bullet (\text{TMR2 Prescale Value}) \quad (2)$$

Note: Tosc = 1/Fosc where is Fosc is oscillator frequency.

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCP1 pin is set (exception: if PWM duty cycle = 0%, the CCP1 pin will not be set)
- The PWM duty cycle is latched from CCPR1L into CCPR1H

PWM Duty Cycle

The PWM duty cycle is specified by writing to the CCPR1L register and to the CCP1CON<5:4> bits which is up to 10-bit resolution is available. The CCPR1L contains the eight MSBs and the CCP1CON<5:4> contains the two LSBs. This 10-bit value is represented by CCPR1L:CCP1CON<5:4>. The following equation is used to calculate the PWM duty cycle in time:

$$\text{PWM Duty Cycle} = (\text{CCPR1L:CCP1CON<5:4>}) \bullet T_{OSC} \bullet (\text{TMR2 Prescale Value}) \quad (3)$$

Note: Tosc = 1/Fosc where is Fosc is oscillator frequency.

CCPR1L and CCP1CON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPR1H until after a match between PR2 and TMR2 occurs (i.e., the period is complete). In PWM mode, CCPR1H is a read-only register.

Setup for PWM

The following steps should be taken when configuring the CCP module for PWM operation:

1. Set the PWM period by writing to the PR2 register.
2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
3. Make the CCP1 pin an output by clearing the TRISC<2> bit.
4. Set the TMR2 prescale value and enable Timer2 by writing to T2CON.
5. Configure the CCP1 module for PWM operation.

PWM Exercise

Configure the CCP1 Module to operate in PWM mode which outputs a signal with a 70% duty cycle at a frequency of 500Hz. Assume that Fosc = 4MHz.

Calculating the value of PR2 given a period of 1/500Hz (0.002 s) and Timer2 prescaler at 1:16:

From Equation 2: $$0.002\ s = [(PR2) + 1]\ x\ 4\ x\ (2.5x10^{-7})\ x\ 16$$

$$PR2 = 124\ (0x7C)$$

Calculating the 10-bit resolution value given a duty cycle of 70%. The PWM duty cycle should be given in time that is 0.7 x 0.002s = 1.4ms.

From Equation 3: $$1.4x10^{-3} = (CCPR1L : CCP1CON<5:4>)\ x\ \frac{1}{4MHz}\ x\ 16$$

$$(CCPR1L : CCP1CON<5:4>) = 350\ or\ 0101011110_2$$

$$CCPR1L = 01010111_2\ or\ 0x57$$

$$CCP1CON<5:4> = 10_2\ or\ 0x2$$

Below is the main() with the CCP module configuration for PWM.

```
void main(void)
{
   /* following the steps in setting up PWM */
   PR2 = 0x7C;          // set value for PR2
   CCPR1L = 0x57;       // set value for (8 MSBs)
   CCP1CON = 0x2C;      // set value for (2 LSBs), PWM mode
   TRISC = 0x00;        // sets all of PORTC (RC2) to output
   RC2 = 0;             // initialize RC2 to 0
   T2CON = 0x06;        // 1:16 prescaler, Timer2 on

  for(;;)               // foreground routine
  {


  }
}
```

CCP1CON sets the 2-bit LSB of the PWM resolution as well as setting the mode to PWM. See the PIC16F87X data sheet for more information.

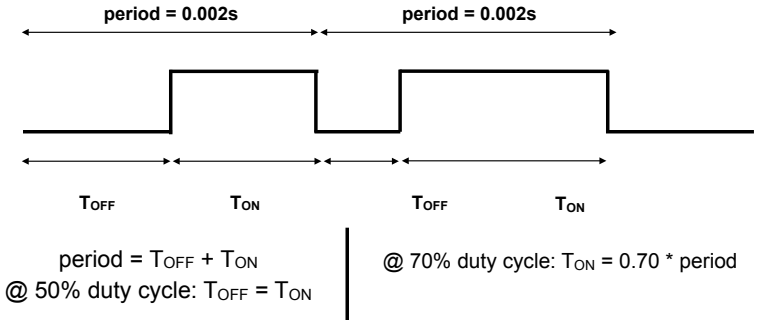


Fig. 3. Signal at 50% and 70% Duty Cycle

Open the Proteus firmware project “LE4-5.pdsprj”. In MPLAB, create a new source file with the filename “LE4-5.c” and add it to the project “CpE 3201-LE4”. Write the source code above with the pre-processor directives. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE4-5.pdsprj”) by loading the generated .hex file to the microcontroller. Connect a virtual oscilloscope to RC2 and observe the output signal. Verify the frequency and the duty cycle of the signal.

Part IV. Hands-on Exercise

Part I

Open the source code in the **Capture Module Exercise** “LE4-3.c” and save it as “LE4-6.c”. Add the source file to the project “CpE 3201-LE4”. Modify the code to display the period to an LCD. In

Proteus, create a firmware project with the filename “LE4-6.pdsprj” and construct the required circuit. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE4-6.pdsprj”) by loading the generated .hex file to the microcontroller. Connect a virtual oscilloscope to RA0. Observe the output signal and the displayed period in the LCD. Test the system with different input frequency (within 1Hz to 1000Hz). Record the period with respect to the input frequency.

Part II

Open the Proteus and create a new firmware project file with the filename “LE4-7.pdsprj”. Connect two two (2) push button switch to any unused port and a virtual oscilloscope on RC2. In MPLAB, create a new source file with the filename “LE4-7.c” and add it to the project “CpE 3201-LE4”. Write a program that will generate a pulse-wave at different frequency and duty cycle. Switch 1 (SW1) will cycle the frequency from 10Hz, 100Hz, 1000Hz while switch 2 (SW2) will cycle the duty cycle from 10%, 25%, 50%, 75%, 95%. Use the PWM module of CCP1 to generate the signal. The foreground routine shall check the switches and modify the PR2 and (CCPR1L:CCP1CON<5:4>) to generate the correct frequency and duty cycle. The default frequency is 10Hz and duty cycle is 10%. Calculate the values for the PR2 and (CCPR1L:CCP1CON<5:4>) for each frequency and duty cycle.

Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE4-7.pdsprj”) by loading the generated .hex file to the microcontroller. Connect a virtual oscilloscope to RC2. Observe the frequency and duty cycle of the output signal while pressing SW1 and SW2.

Instructions for submission:

- Write a report that contains the data in Part I and calculations in Part II. The filename of the document should be “<LAST NAME>_LE4.pdf”.
- Submit the report together with the following:
  - LE4-6.pdsprj
  - LE4-6.c
  - LE4-7.pdsprj
  - LE4-7.c
- Submit the files in Canvas in the correct order.

Assessment

Criteria	Outstanding (4 pts)	Competent (3 pts)	Marginal (2 pts)	Not Acceptable (1 pt)	None (0 pts)
Feature Configuration	Configuration was properly done.	Configuration has minor errors.	-	Configuration is incorrect.	No project created.
Functionality	The systems function perfectly.	The system functions with minor issues.	The system has several issues which already affect the functionality.	The presented system is non-functioning.	No project created.
Firmware	Firmware created successfully without issues and followed the requirements.	Firmware created successfully with some issues and followed the requirements.	Firmware has issues and missing some of the requirements.	Firmware has several issues and did not follow the requirements.	No project created.

Copyright Information

Author: Van B. Patiluna (vbpatiluna@usc.edu.ph)
Contributors: none
Date of Release: March 4, 2021
Version: 1.0.1

References

PIC16F87X Data Sheet, Microchip Technology Inc. 2003.
CpE 3201 Lecture Notes on Timers in PIC16F877A.

Change Log:

Date	Version	Author	Changes
November 14, 2018	1.4	Van B. Patiluna	Original Laboratory Guide for CpE 426N.
March 4, 2021	1.0	Van B. Patiluna	- Removed references to MB90F387S. - Revised the structure of the exercise. - Revise the hands-on exercise. - Revised the rubric to a unified one.
March 4, 2021	1.0.1	Van B. Patiluna	- Corrected some factual errors. - Added ADCON1 configuration for digital I/O in PORTA.