EMBEDDED SYSTEMS DEVELOPMENT

(CSE 320)

[LAB MANUAL]

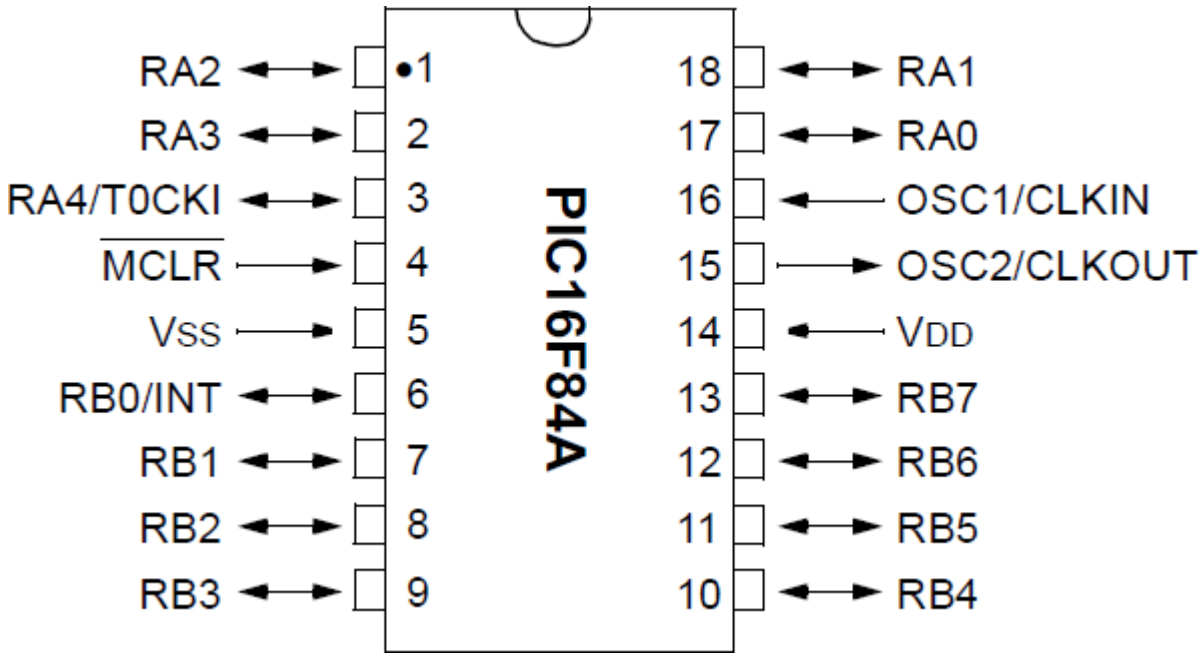
UNINVERSITY OF ENGINEERING & TECHNOLOGY LAHORE

COPMPUTER SCIENCE AND ENGINEERING DEPARTMENT

List of Experiments

Experiment No. 1	To Toggle LED Using PIC16F84A.
Experiment No. 2	Serial Communication Through PIC16F84A.
Experiment No. 3	To Toggle LED Using PIC16F877A.
Experiment No. 4	Serial Communication Through PIC16F877A.
Experiment No. 5	DC Motor Direction Control Through PIC16F877A.
Experiment No. 6	DC Motor Speed Control Through PIC16F877A.
Experiment No. 7	To Interface LCD With PIC16F877A.
Experiment No. 8	Encoder Reading Using External Interrupt.
Experiment No. 9	To Interface LCD And Key Pad With PIC16F877A.
Experiment No. 10	Stepper Motor Control Using PIC16F877A.
Experiment No. 11	To Switch Bulb On And Off Using Relay
Experiment No. 12	PIC16F877A &L M35 Based A/D Conversion
Experiment No. 13	PIC16F877A and Push Button
Experiment No.14	Semester Project
Experiment No.15	Semester Project

Introduction to PIC16F84A



- Pin no.1 RA2 Second pin on port A. Has no additional function
- Pin no.2 RA3 Third pin on port A. Has no additional function.
- Pin no.3 RA4 Fourth pin on port A. TOCK1 which functions as a timer is also found on this pin
- Pin no.4 MCLR Reset input and Vpp programming voltage of a microcontroller
- Pin no.5 Vss Ground of power supply.
- Pin no.6 RB0 Zero pin on port B. Interrupt input is an additional function.
- Pin no.7 RB1 First pin on port B. No additional function.
- Pin no.8 RB2 Second pin on port B. No additional function.
- Pin no.9 RB3 Third pin on port B. No additional function.
- Pin no.10 RB4 Fourth pin on port B. No additional function.
- Pin no.11 RB5 Fifth pin on port B. No additional function.
- Pin no.12 RB6 Sixth pin on port B. 'Clock' line in program mode.
- Pin no.13 RB7 Seventh pin on port B. 'Data' line in program mode.
- Pin no.14 Vdd Positive power supply pole.
- Pin no.15 OSC2 Pin assigned for connecting with an oscillator
- Pin no.16 OSC1 Pin assigned for connecting with an oscillator
- Pin no.17 RA0 Second last pin on port A. No additional function
- Pin no.18 RA1 First pin on port A. No additional function.

XT Oscillator

- Crystal oscillator is kept in metal housing
- One ceramic capacitor of 30pF whose other end is connected to the ground needs to be connected with each pin.
- Oscillator and capacitors can be packed in joint case with three pins. Such element is called ceramic resonator.

16F84A Features

• 1024 words of program memory
• 68 bytes of Data RAM
• 64 bytes of Data EEPROM
• 14-bit wide instruction words
• 8-bit wide data bytes
• 15 Special Function Hardware registers

Four interrupt sources

- External RB0/INT pin
- TMR0 timer overflow
- PORTB<7:4> interrupt-on-change
- Data EEPROM write complete

I/O Ports

PORTA and TRISA Registers: PORTA is a 5-bit wide, bi-directional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input. Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

PORTB and TRISB Registers: PORTB is an 8-bit wide, bi-directional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input. Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin).

TIMER0 Module

The Timer0 module timer/counter has the following features:
• 8-bit timer/counter
• Readable and writable
• Internal or external clock select
• Edge select for external clock
• 8-bit software programmable prescaler
• Interrupt-on-overflow from FFh to 00h

Interrupts

The PIC16F84A has 4 sources of interrupt:
- External interrupt RB0/INT pin
- TMR0 overflow interrupt
- PORTB change interrupts (pins RB7:RB4)
- Data EEPROM write complete interrupt

INT INTERRUPT

External interrupt on RB0/INT pin is edge triggered: either rising if INTEDG bit (OPTION_REG<6>) is set, or falling if INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, the INTF bit (INTCON<1>) is set. This interrupt can be disabled by clearing control bit INTE (INTCON<4>). Flag bit INTF must be cleared in software via the Interrupt Service Routine before re-enabling this interrupt. The INT interrupt can wake the processor from SLEEP only if the INTE bit was set prior to going into SLEEP. The status of the GIE bit decides whether the processor branches to the interrupt vector following wake-up.

TMR0 INTERRUPT

An overflow (FFh to 00h) in TMR0 will set flag bit T0IF (INTCON<2>). The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>).

PORTB INTERRUPT

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit RBIE (INTCON<3>).

DATA EEPROM INTERRUPT

At the completion of a data EEPROM write cycle, flag bit EEIF (EECON1<4>) will be set. The interrupt can be
enabled/disabled by setting/clearing enable bit EEIE (INTCON<6>).

EECON1 REGISTER (ADDRESS 88h)

U-0	U-0	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
—	—	—	EEIF	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7-5 Unimplemented: Read as '0'

bit 4 EEIF: EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 WRERR: EEPROM Error Flag bit
1 = A write operation is prematurely terminated
0 = The write operation completed

bit 2 WREN: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 WR: Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete.
0 = Write cycle to the EEPROM is complete

bit 0 RD: Read Control bit
1 = Initiates an EEPROM read RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **RBPU:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

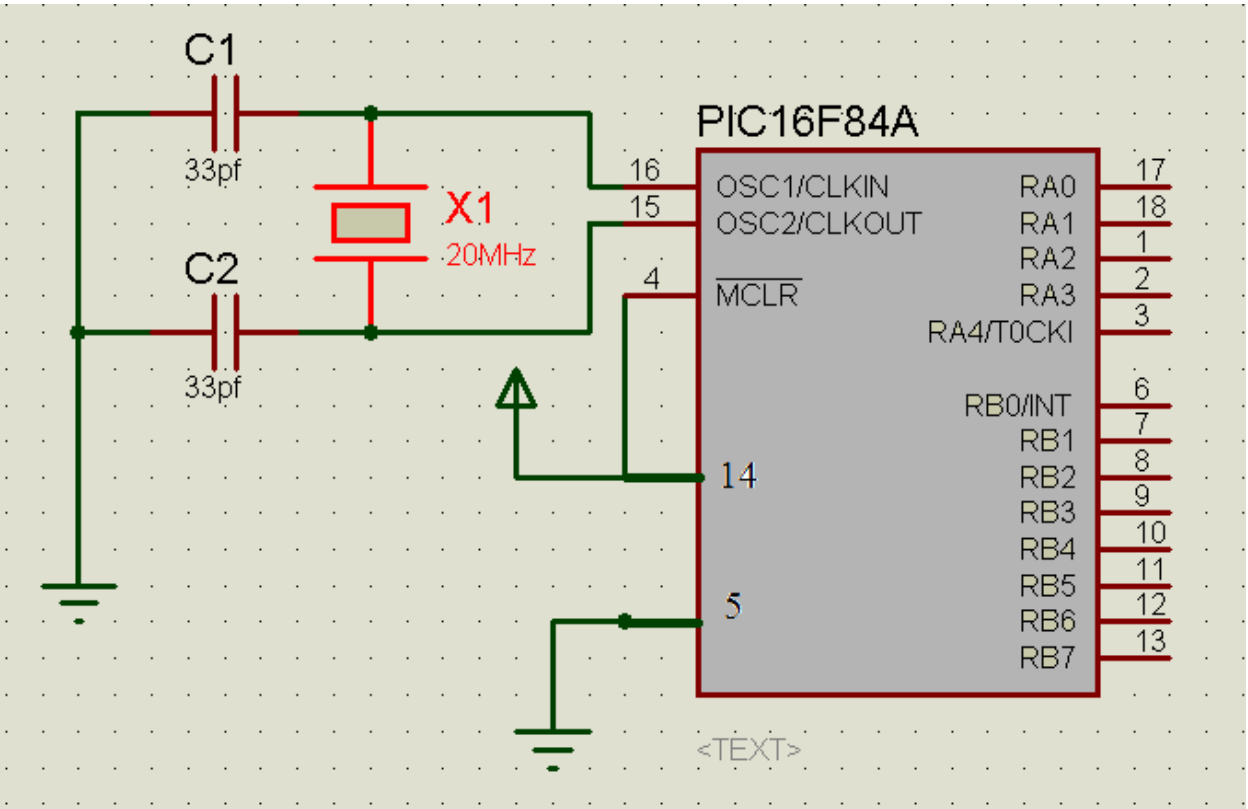
bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Basic Schematic Diagram



Embedded systems (LAB)
Experiment NO: 1
To toggle LED using PIC16F84A

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 1	**Dated:**
Semester:6TH	**Session:2020**
Lab/Project/Assignment #: 1	**CLOs to be covered:**
Lab Title: To toggle LED using PIC16F84A.	**Teacher Name: Ms. Sana Tasleem**

To toggle LED using PIC16F84A.

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

Rubrics for Current Lab (Optional):

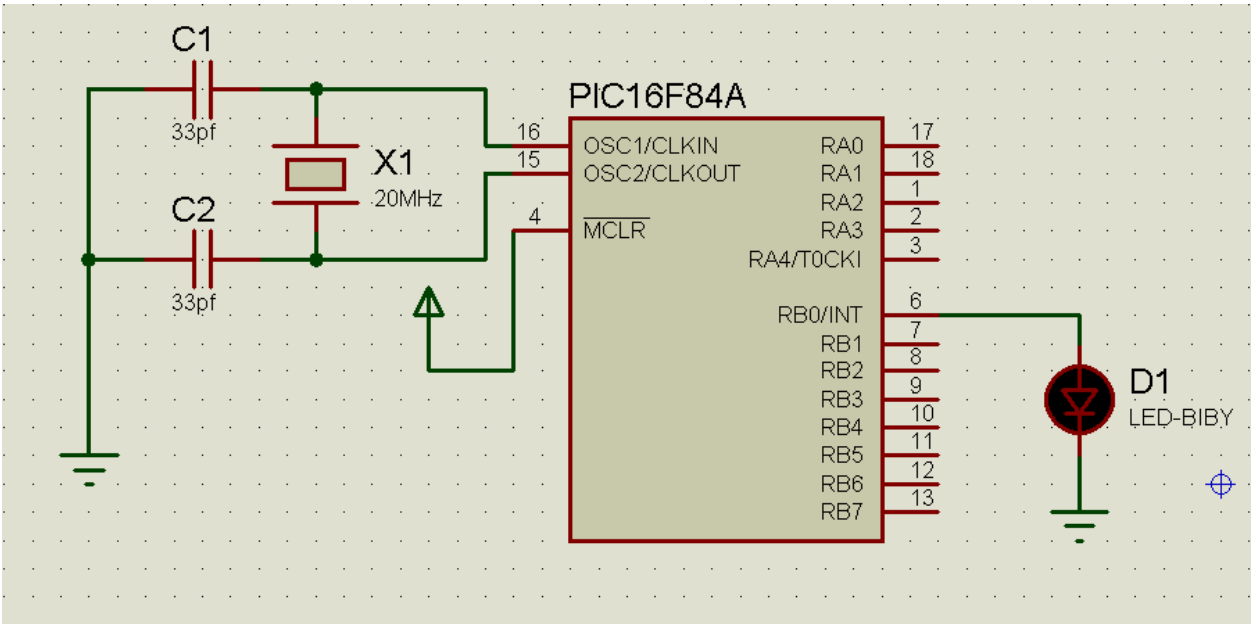
Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

LAB DETAILS:

Lab Goals/Objectives:

- To understand basic circuit of 16F84A
- To understand Timer functionality
- To toggle LED

Schematic Diagram



Description

To toggle the LED on any pin of the data port use the following function:
output_toggle (PIN_B0)
The delay can be introduced by using the following function:
delay_ms(10); OR delay_µs(10)
It produces the delay in millisecond and microsecond.
The LED can also be toggled by switching on and off LED using the following functions:
output_high(PIN_B0);
output_low(PIN_B0);
The delay can also be produced by timers. PIC18F84A has only one timer which is timer0 and it is of 8-bit.

The timer interrupt rate can be set as follow:

$$\text{Interrupt Rate} = \frac{clock\ frequency}{4 * 256 * pre\ scalar}$$

The delay can be calculated as:

$$\text{Delay} = \frac{1}{Interrupt\ Rate}$$

Source Code

The source code using delay function to toggle LED every second is given below:

```
void main()
{
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
  enable_interrupts(INT_TIMER0);
  enable_interrupts(GLOBAL);

  while(1)
  {
        output_toggle(PIN_B0);
        delay _ms(1000);

  }
}
```

The source code using timer interrupt to toggle LED every one second is given below:

```
TIMER0_isr()
{
  static long count=1;

  if(count>=19532)
  {
    count=0;
    output_toggle(PIN_B0);
  }
  count++;
}


void main()
{

  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
  enable_interrupts(INT_TIMER0);
  enable_interrupts(GLOBAL);

  while(1)
  {
    ;
  }
}
```

Experiment NO: 2

Serial Communication through PIC16F84A.

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 2	**Dated:**
Semester:6TH	**Session:2020**
Lab: 2	**CLOs to be covered:**
Lab Title: Serial Communication through PIC16F84A.	**Teacher Name: Ms. Sana Tasleem**

Serial Communication through PIC16F84A.

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

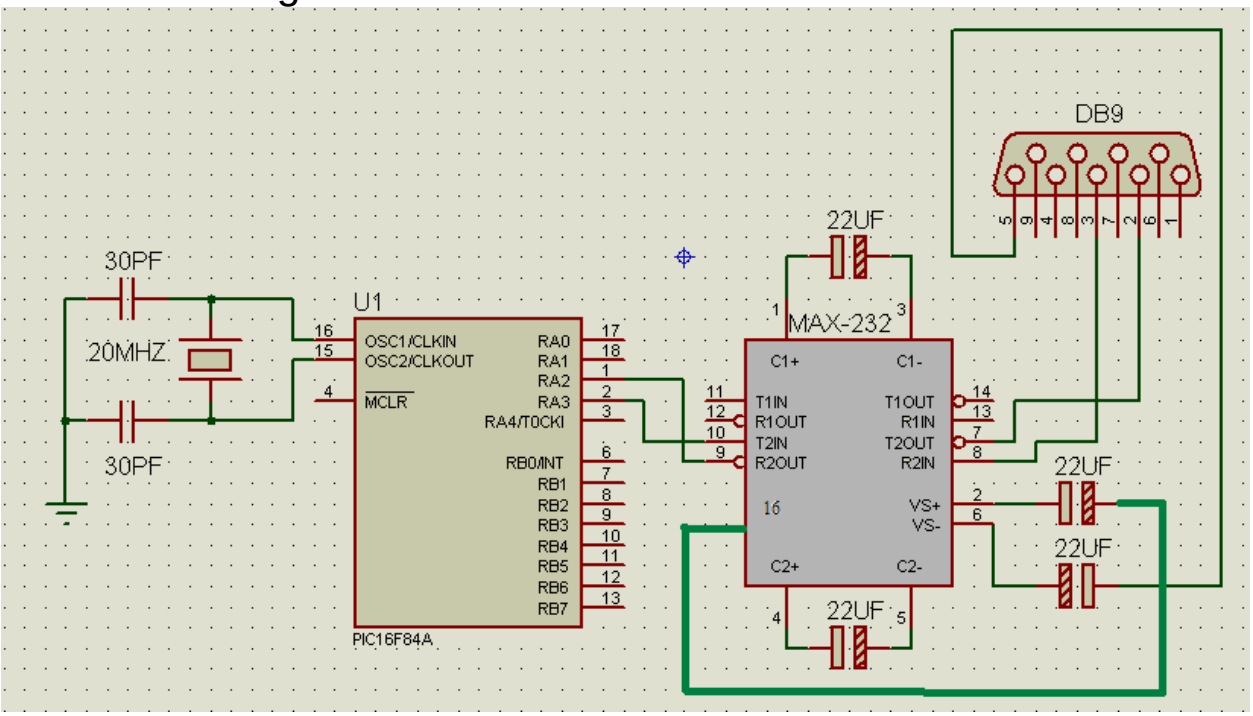
Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

- To send data from controller to PC Hyper Terminal

Schematic Diagram



Description

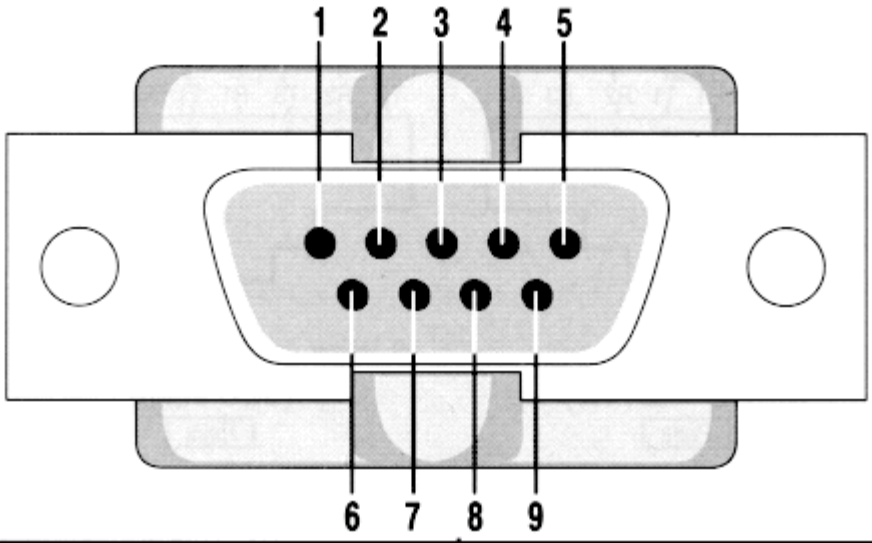
MAX-232

The MAX232 is an integrated circuit that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX232 is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals.

The drivers provide RS-232 voltage level outputs (approx. ± 7.5 V) from a single + 5 V supply via on-chip charge pumps and external capacitors. This makes it useful for implementing RS-232 in devices that otherwise do not need any voltages outside the 0 V to + 5 V range, as power supply design does not need to be made more complicated just for driving the RS-232 in this case.

The receivers reduce RS-232 inputs (which may be as high as ± 25 V), to standard 5 V TTL levels. These receivers have a typical threshold of 1.3 V, and a typical hysteresis of 0.5 V.

DB9



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Send/Receive Functions

getc()

Function: This function waits for a character to come in over the RS232 RCV pin and returns the character. If you do not want to hang forever waiting for an incoming character use kbhit() to test for a character available.

Returns: An 8 bit character.

putc()

putc (***cdata***)

Parameters: ***cdata*** is a 8 bit character.

Function: This function sends a character over the RS232 XMIT pin.

HEADER FILE

To set baud rate, parity, receive pin and transmit pin, add the following line at the end of the header file.

#use rs232(baud=9600,parity=N,xmit=PIN_A3,rcv=PIN_A2,bits=9)

Source Code

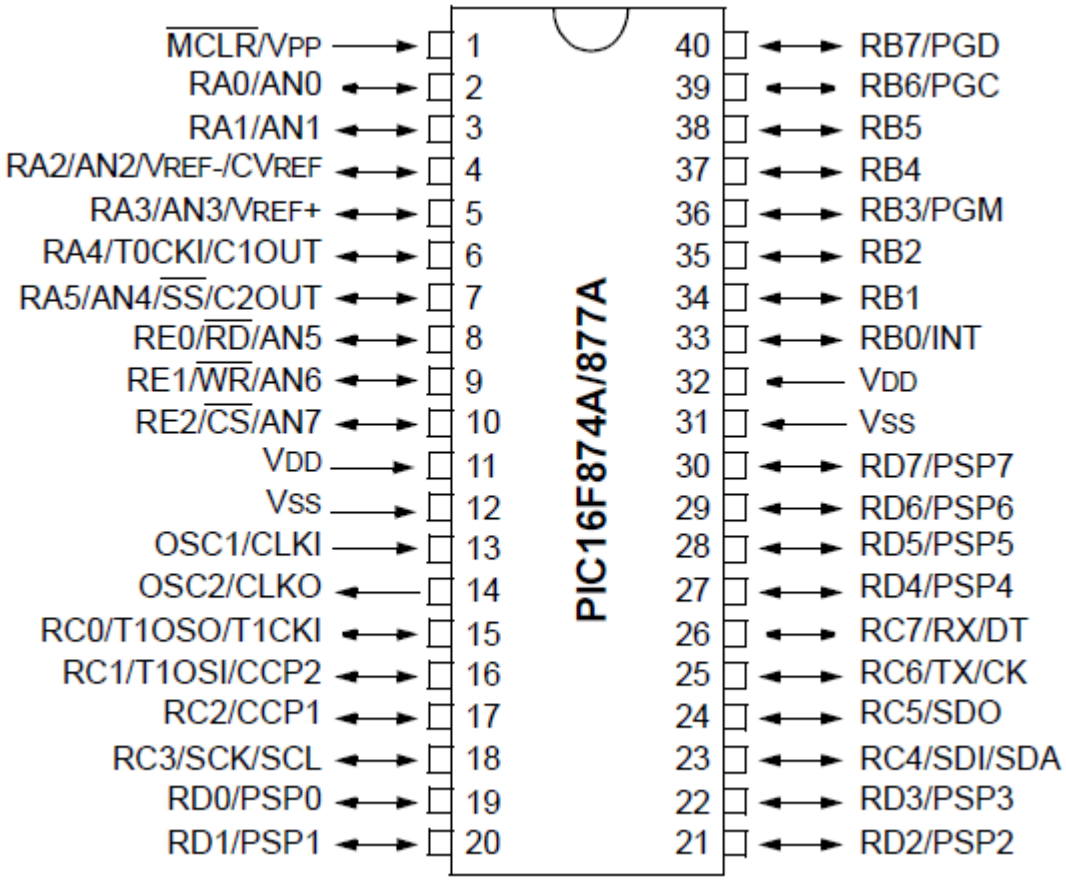
The source code to send a single character after avery one second to the hyper terminal is given below:

```
void main()
{

  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
  enable_interrupts(INT_TIMER0);
  enable_interrupts(GLOBAL);
```

```
  while(1)
  {
       putc('c');
      delay_ms(1000);
  }
}
```

Introduction to PIC16F877A



16F877A Features

- 14.3K Bytes Program Memory
- 368 Bytes Data SRAM
- 256 Bytes EEPROM
- 35 Instructions
- 5 I/O Ports (33 I/O Pins)
- 10-bit Analog-to-Digital Module (8 input channel)
- 2 CCP (PWM)
- 2 8-bit Timers, 1 16-bit Timer
- 2 Comparators
- Master Synchronous Serial Port (MSSP) Module

I/O PORTS

The corresponding data direction register is TRISX. Setting a TRISX bit (= 1) will make the corresponding PORTX pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISX bit (= 0) will make the corresponding PORTX pin an output (i.e., put the contents of the output latch on the selected pin).

PORTA and the TRISA Register:
PORTA is a 6-bit wide, bidirectional port.
PORTB and the TRISB Register:

PORTB is an 8-bit wide, bidirectional port.

PORTC and the TRISC Register:

PORTC is an 8-bit wide, bidirectional port.

PORTD and TRISD Registers:

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

PORTE and TRISE Register:

PORTE has three pins (RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7) which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers.

TIMERS

TIMER0 MODULE

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter
- Readable and writable
- 8-bit software programmable prescaler
- Internal or external clock select
- Interrupt on overflow from FFh to 00h
- Edge select for external clock

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
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **RBPU**

bit 6 **INTEDG**

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

TIMER1 MODULE

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 interrupt, if enabled, is generated on overflow which is latched in interrupt flag bit, TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a Timer
- As a Counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'
bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits
bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)
bit 2 **T1SYNC**: Timer1 External Clock Input Synchronization Control bit When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.
bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (FOSC/4)
bit 0 **TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

TIMER2 MODULE

Timer2 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). The TMR2 register is readable and writable and is cleared on any device Reset.

T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'
bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off
bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

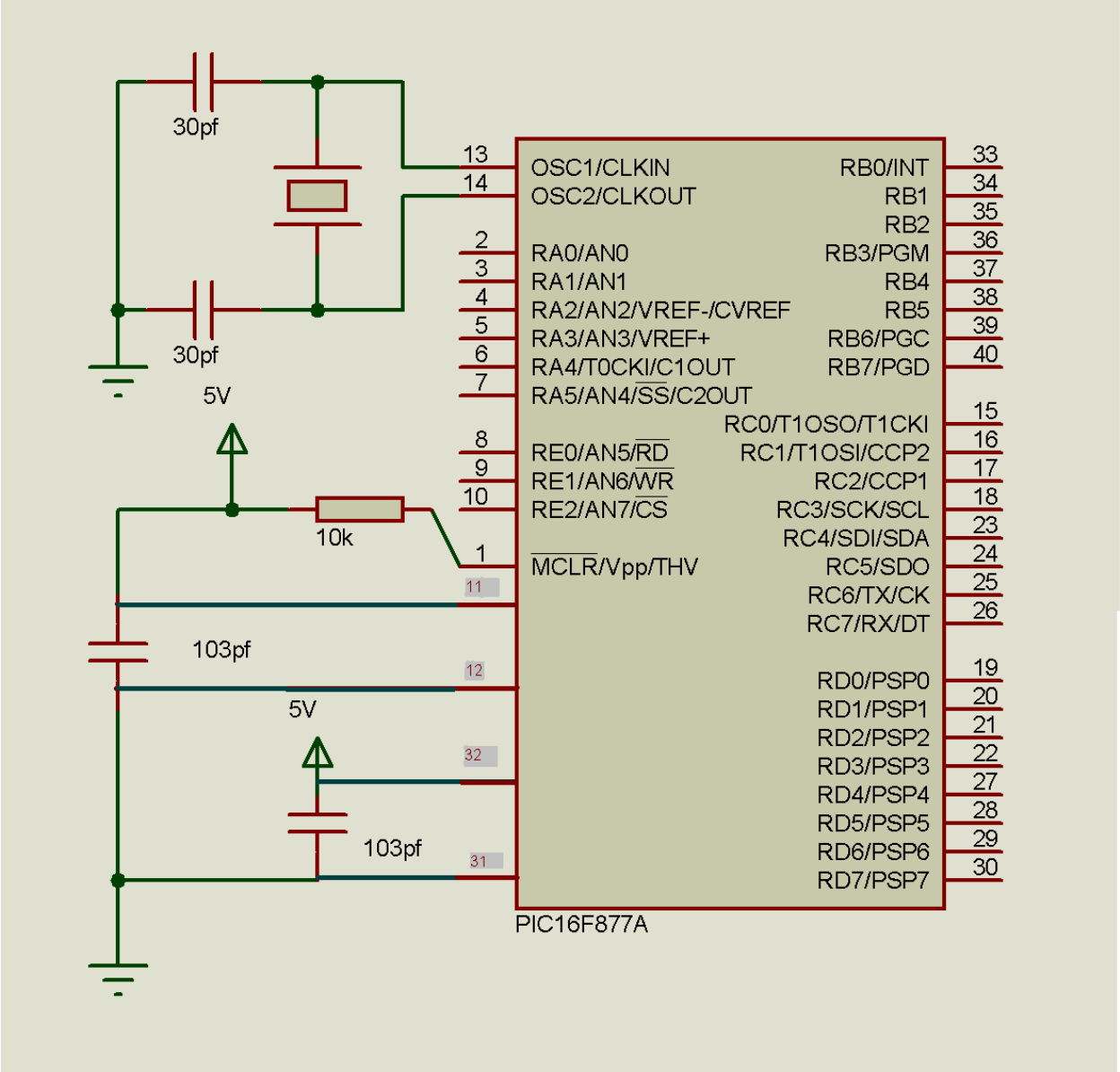
CAPTURE/COMPARE/PWM MODULES

Each Capture/Compare/PWM (CCP) module contains a 16-bit register which can operate as a:
• 16-bit Capture register
• 16-bit Compare register
• PWM Master/Slave Duty Cycle register

CCP MODE – TIMER RESOURCES REQUIRED

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

Basic Schematic Diagram



EXPERIMENT NO 3
To toggle LED using PIC16F877A.

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 3	**Dated:**
Semester:6TH	**Session:2020**
Lab: 3	**CLOs to be covered:**
Lab Title: To toggle LED using PIC16F877A.	**Teacher Name: Ms. Sana Tasleem**

To toggle LED using PIC16F877A.

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

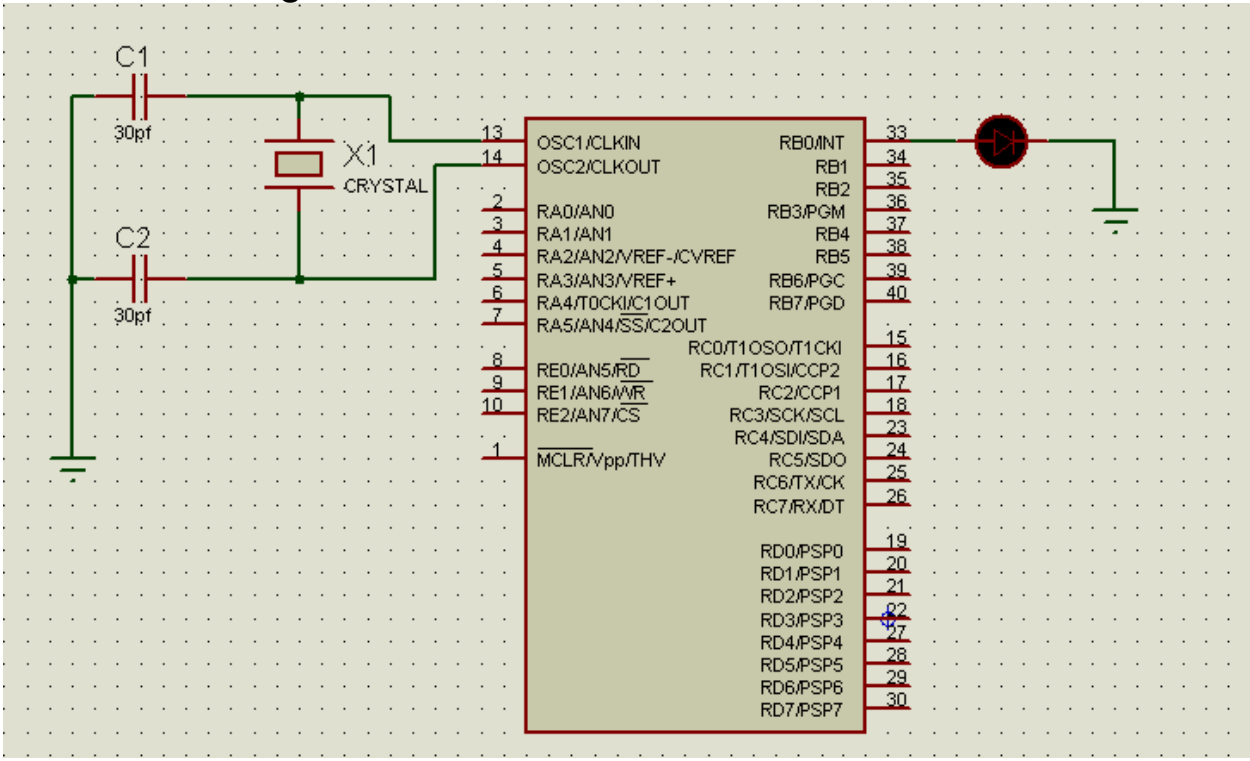
Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

- To understand basic circuit of 16F877A
- To understand Timer functionality
- To toggle LED

Schematic Diagram



Description

To toggle the LED on any pin of the data port use the following function:
output_toggle (PIN_B0)
The delay can be introduced by using the following function:
delay_ms(10); OR delay_µs(10)
It produces the delay in millisecond and microsecond.
The LED can also be toggled by switching on and off LED using the following functions:
output_high(PIN_B0);
output_low(PIN_B0);
The delay can also be produced by timers. PIC18F84A has only one timer which is timer0 and it is of 8-bit. The timer interrupt rate can be set as follow:

$$\text{Interrupt Rate} = \frac{clock\ frequency}{4 * 256 * pre\ scalar}$$

The delay can be calculated as:

$$\text{Delay} = \frac{1}{Interrupt\ Rate}$$

Source Code

The source code using delay function to toggle LED every second is given below:

```
void main()
{
```

```
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

  while(1)
  {
        output_toggle(PIN_B0);
        delay _ms(1000);

  }
}
```

The source code using timer interrupt to toggle LED every one second is given below:

```
TIMER0_isr()
{
  static long count=1;

  if(count>=19532)
  {
    count=0;
    output_toggle(PIN_B0);
  }
  count++;
}


void main()
{

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

  while(1)
  {
    ;
  }
}
```

EXPERIMENT NO 4

Serial Communication through PIC16F877A.

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 4	**Dated:**
Semester:6TH	**Session:2020**
Lab: 4	**CLOs to be covered:**
Lab Title: Serial Communication through PIC16F877A	**Teacher Name: Ms. Sana Tasleem**

Serial Communication through PIC16F877A

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

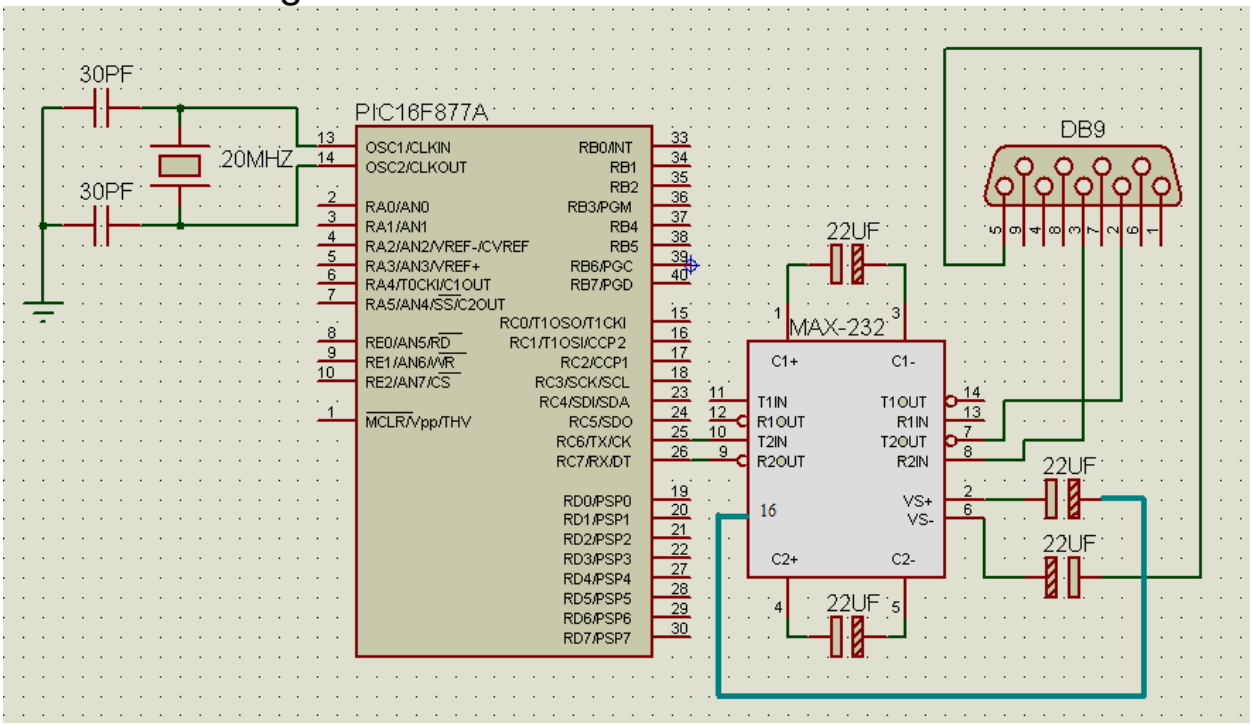
Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

- To understand the concept of serial communication
- To send data from controller to PC Hyper Terminal

Schematic Diagram



Description

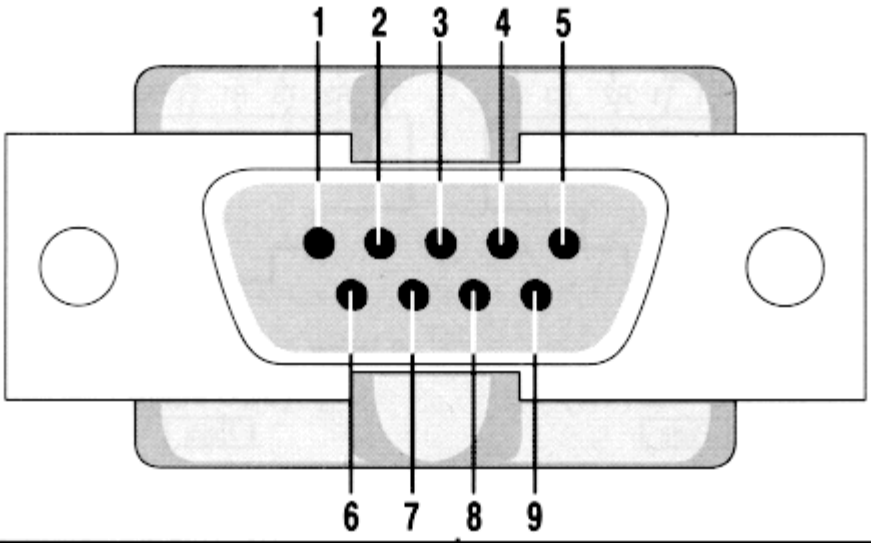
MAX-232

The MAX232 is an integrated circuit that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX232 is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals.

The drivers provide RS-232 voltage level outputs (approx. ± 7.5 V) from a single + 5 V supply via on-chip charge pumps and external capacitors. This makes it useful for implementing RS-232 in devices that otherwise do not need any voltages outside the 0 V to + 5 V range, as power supply design does not need to be made more complicated just for driving the RS-232 in this case.

The receivers reduce RS-232 inputs (which may be as high as ± 25 V), to standard 5 V TTL levels. These receivers have a typical threshold of 1.3 V, and a typical hysteresis of 0.5 V.

DB9:-



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Send/Receive Functions

getc()

Function: This function waits for a character to come in over the RS232 RCV pin and returns the character. If you do not want to hang forever waiting for an incoming character use kbhit() to test for a character available.

Returns: An 8 bit character.

putc()

putc (***cdata***)

Parameters: ***cdata*** is a 8 bit character.

Function: This function sends a character over the RS232 XMIT pin.

HEADER FILE

To set baud rate, parity, receive pin and transmit pin, add the following line at the end of the header file.
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=9)

Source Code

The source code to send a single character after avery one second to the hyper terminal is given below:

```
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
```

```
  while(1)
  {
       putc('c');
      delay_ms(1000);
  }
}
```

EXPERIMENT NO 5
DC Motor Direction Control through PIC16F877A

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 5	**Dated:**
Semester:6TH	**Session:2020**
Lab: 5	**CLOs to be covered:**
Lab Title: DC Motor Direction Control through PIC16F877A	**Teacher Name: Ms. Sana Tasleem**

DC Motor Direction Control through PIC16F877A

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

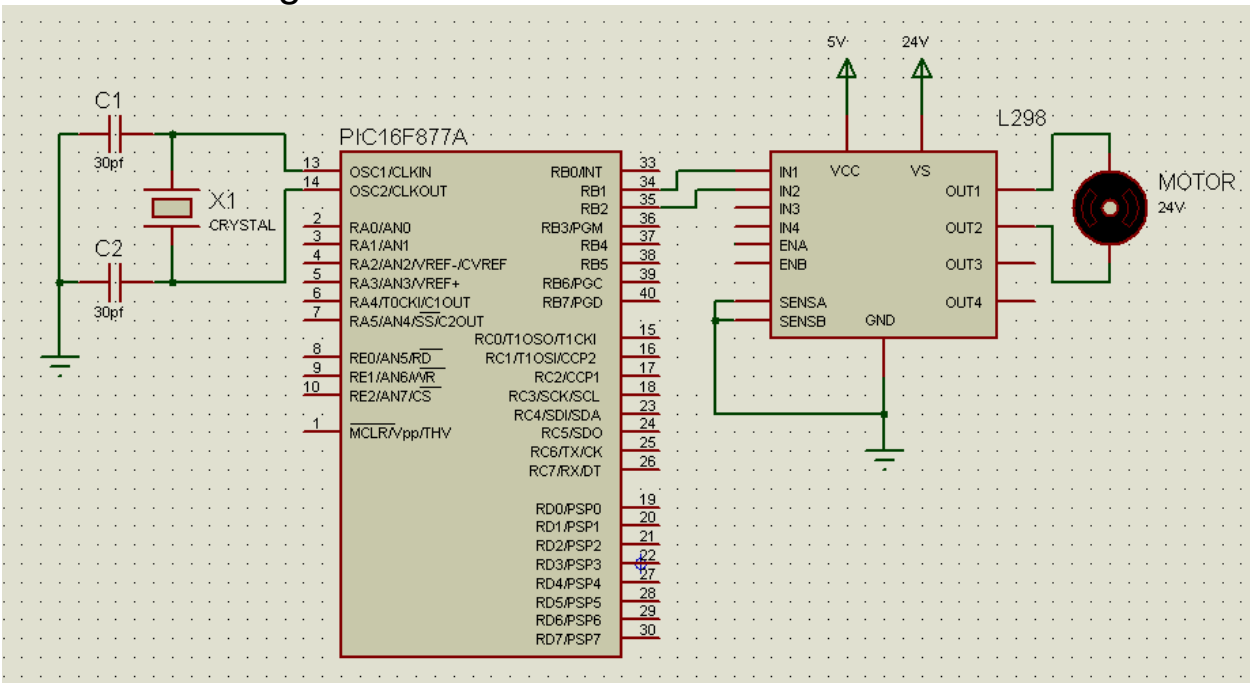
Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

- To understand the working of L298.
- To understand the PIC PWM Module.
- To drive a DC motor.

Schematic Diagram



L298

The L298 IC is consists of two H-Bridge modules, which can be used to drive two motors at a time. The H-Bridge module used in this experiment uses pin IN1(C),IN2(D) and ENA(Ven). IN1 and IN2 pins are used to control motor direction. ENA pin is used to turn on or off the H-Bridge module.

Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Source Code

The source code to send a single character after every one second to the hyper terminal is given below:

```
void main()
{
```

```
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

  output_low( PIN_B1 );
  output_low( PIN_B2 );

  while(1)
  {
        ;
  }
}
```

EXPERIMENT NO 6

DC Motor Speed Control through PIC16F877A.

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 6	**Dated:**
Semester:6TH	**Session:2020**
Lab: 6	**CLOs to be covered:**
Lab Title: DC Motor speed Control through PIC16F877A	**Teacher Name: Ms. Sana Tasleem**

DC Motor speed Control through PIC16F877A

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

Rubrics for Current Lab (Optional):

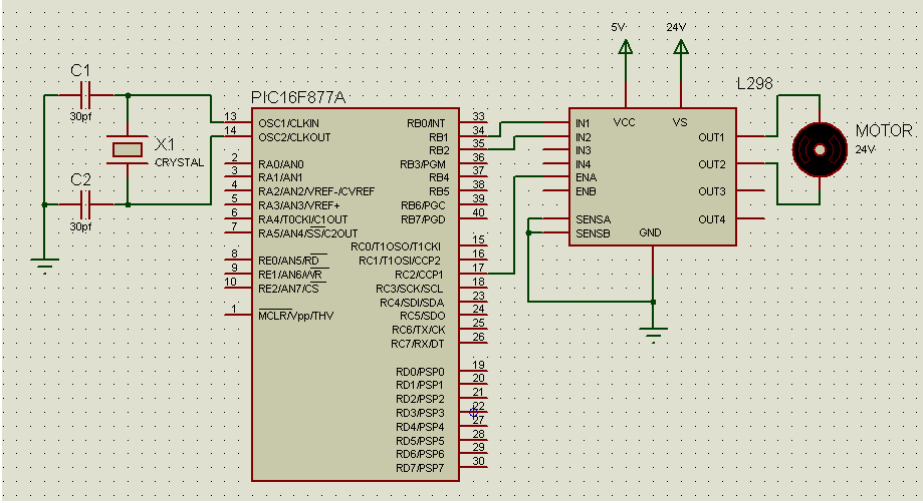
Scale	Marks	Level	Rubric

Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

- To understand the working of L298.
- To understand the PIC PWM Module.
- To drive a DC motor.

Schematic Diagram



Description

PWM Waveform

PWM Frequency

$$\text{PWM Frequency} = \frac{clock\ frequency}{4 * pre\ scalar\ *\ (1\ +\ pre\ scalar)}$$

For example, 20 MHz clock, prescaler = 1, period register = 255:

$$\text{PWM Frequency} = \frac{20000000}{4 * 1\ *\ (1\ +\ 255)}\ =\ 19.53\ \text{KHz}$$

PWM Related Functions

setup_ccp1(CCP_PWM) sets PWM mode
set_pwm1_duty(q) sets duty-cycle register to *q*
Note that *q* should not exceed the value of the period register
It is also necessary to configure counter/timer 2:
setup_timer_2(pre-scaler, period, 1);
pre-scaler is one of: T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_1
period (the period register) is an int 0-255

L298

The L298 IC is consists of two H-Bridge modules, which can be used to drive two motors at a time. The H-Bridge module used in this experiment uses pin IN1(C),IN2(D) and ENA(Ven). IN1 and IN2 pins are used to control motor direction. ENA pin is used to turn on or off the H-Bridge

module.

Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

HEADER FILE

To set baud rate, parity, receive pin and transmit pin, add the following line at the end of the header file.

#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=9)

Source Code

The source code to send a single character after every one second to the hyper terminal is given below:

```
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

  setup_ccp1(CCP_PWM);                    // PWM Generate Mode
  setup_timer_2(T2_DIV_BY_4,63,1);        // 19.53KHz Frequency
```

```
  set_pwm1_duty(128);                    // 50% duty cycle

  output_low( PIN_B1 );
  output_low( PIN_B2 );

  while(1)
  {
        ;
  }
}
```

EXPERIMENT NO 7

To interface LCD with PIC16F877A.

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 4	**Dated:**
Semester:6TH	**Session:2020**
Lab: 4	**CLOs to be covered:**
Lab Title: to interface LCD with PIC 16F877A	**Teacher Name: Ms. Sana Tasleem**

To interface LCD with PIC16F877A

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

Rubrics for Current Lab (Optional):

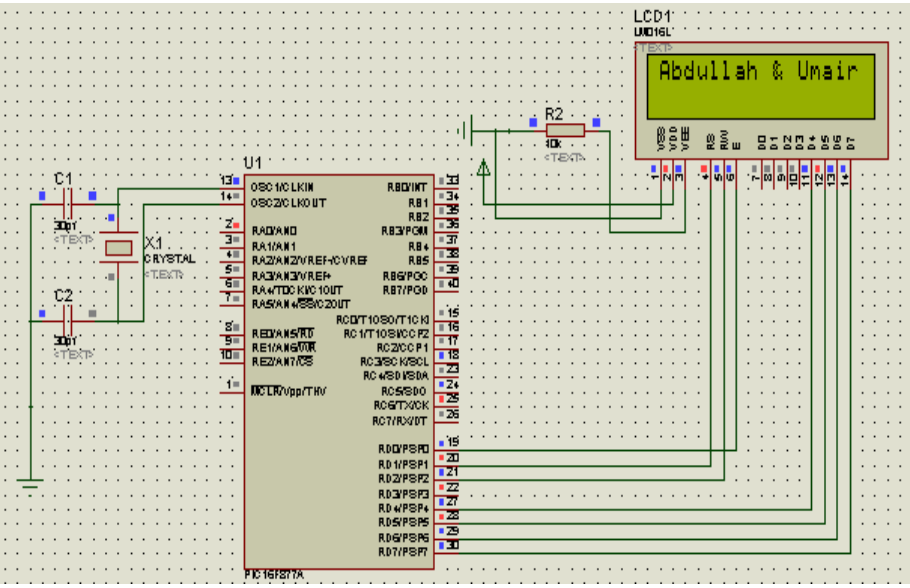
Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.

Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

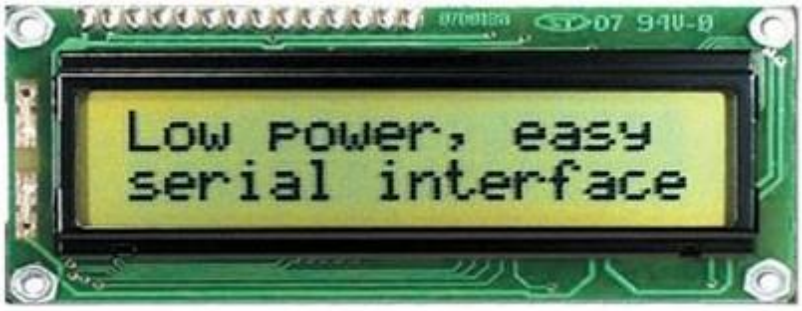
- To understand the functionality of LCD.
- To interface LCD with microcontroller

Schematic Diagram



Theory of Operation

Liquid crystal Display (LCD) is one of the common method of displaying information. In this experiment we will use 2x16 display LCD as shown in figure below:



It has 16 pins and description of each pin is given below:

Pin No.	Symbol	Description
1	VSS	Ground(0V).
2	VDD	Power supply for logic (+5V)
3	V0	Power supply for LCD driver
4	RS	Register Select Input: "High" for Data register (for read and write) "Low" for Instruction register (for write), Busy flag, address counter (for read)
5	R/W	Read/Write signal: " High" for Read mode. "Low" for Write mode.
6	E	Enable. Start signal for data read /write.
7	DB0	Data input/output (LSB)
8	DB1	Data input/output
9	DB2	Data input/output
10	DB3	Data input/output
11	DB4	Data input/output
12	DB5	Data input/output
13	DB6	Data input/output
14	DB7	Data input/output (MSB)
15 or K	LED(-)	Cathode of LED backlight
16 or A	LED(+)	Anode of LED backlight

The LCD has a 7-wire interface (4 data and 3 control) and the connections are hard-wired as follow:

```
#define en pin_a1
#define rw pin_a2
#define rs pin_a3
#define d4 pin_d0
#define d5 pin_d1
#define d6 pin_d2
#define d7 pin_d3
```

The functionality of LCD is already defined in PIC C compiler in lcd.c. Include this file as follow:

```
#include ”lcd.c”
```

To initialize LCD we will use following function:

```
Lcd_init();
```

This function is defined in lcd.c. This function setup the PIC I/O pins used to Communicate with LCD and initializes LCD registers.

The various routines can be used to control the display as follow:

lcd_clear()	clear complete display
lcd_home()	goto 1st character on 1st line
lcd_backspace()	backspace by 1 character
lcd_panleft()	pan complete display left
lcd_panright()	pan complete display right
lcd_gotoxy(int x, int y)	goto x character on y line
lcd_putc(char c)	write character at current pos

Besides these /f can be used to clear screen of LCD and /n ca be used to go to the second line.

printf() (print formatted) function is used for output to the LCD, for example:

```
printf(lcd_putc, "\fTime = %d s", t);
```

Or the following function can be used,

```
lcd_putc("\f CS&E");
```

Source Code

```
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  lcd_init();
  delay_ms(6);
while(1)
{
lcd_putc("\fComputer Engineering Department");
delay_ms(2000);
lcd_putc("\f Embedded Lab");
delay_ms(2000);

}
```

EXPERIMENT NO 8

To interface LCD and key pad with PIC16F877A.

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 8	**Dated:**
Semester:6TH	**Session:2020**
Lab: 8	**CLOs to be covered:**
Lab Title: to interface LCD and keypad with using PIC 16F877A	**Teacher Name: Ms. Sana Tasleem**

To interface LCD and key pad with PIC16F877A

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

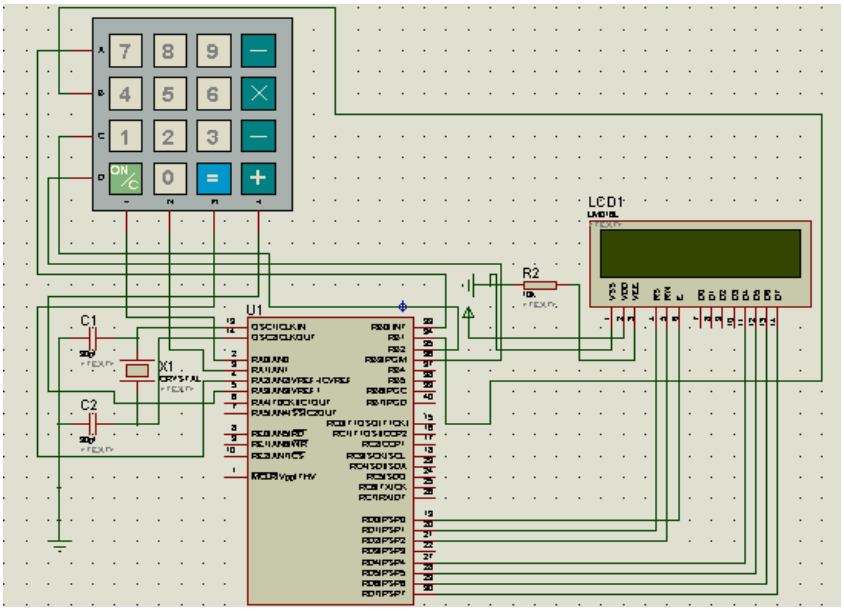
Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

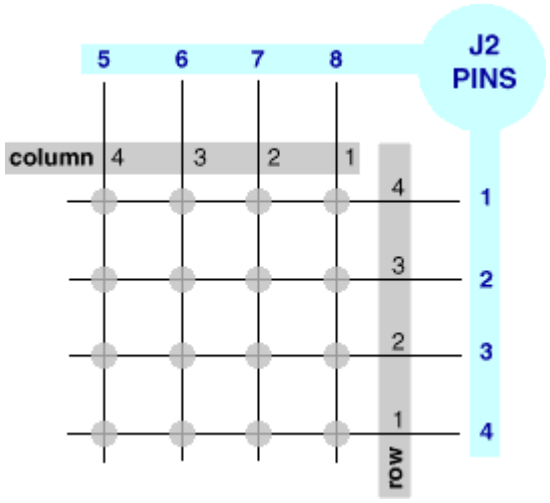
- To understand the functionality of keypad.
- To interface keypad with microcontroller and LCD.

Schematic Diagram



Theory of Operation

A 4x4 matrix keypad consists of 16 switches wired across 4 row and 4 column connections as shown here:



Generic matrix keypad.

This arrangement allows just 8 circuit connections to read the states of 16 switches. Row and column numbers are arbitrary and vary from one keypad manufacturer to another. One problem with using a mechanical switch is that when it is closed, it “bounces” for a few milliseconds (i.e it makes a number of very rapid make and break actions before the contacts become stable, which lets the micro-controller see that it has been pressed several times, and gives the wrong information. There are two ways to debounce a switch:

• Hardware, using an R-S flip-flop.
• Software, using a time delay of usually 20 mS.

The second method is the one used in software programming. To debounce, we use a time delay loop (usually 20-30ms) longer than the duration of the switch bouncing action.

Scanning and Identifying Keys

The rows are connected to output port and column are connected to input port. If no key has been pressed, reading the input port will 1s for all column since they are all connected to Vcc. If all rows are grounded and key is pressed, one of the column will have 0 sice the key pressed provides the path to ground. It is the function of microcontroller toscan the keyboard continuously to detect and identify the key pressed. To detect the pressed key, microcontroller ground all rows by providing 0 to the output latch, then it reads the columns. If the data read from the columns is 1111, no key has been pressed and process continues until a key press is detected. However, if one of the column bit has zero, this means that the key pressed has

occurred. After key press is detected, the microcontroller will go through the process of identifying the key. Starting with top row, the microcontroller ground it by providing a low to row 1 only and then it reads the columns. If data reads is all 1's, no key in that row is activated and the process is moved to the next row. It grounds the next row, read the columns and checks for any zero. This process continues until the row is identified. After identification of row in which the key has been pressed, the next task is to find out which column the pressed key belongs to. This should be easy since microcontroller knows at any time which row and column are being accessed. After the key has been accessed, it is compared with an array and its corresponding value is printed on the LCD. The operation of LCD has already been explained in the previous experiment.

In this experiment, keypad rows are connected to portb(0-3) and columns to porta(0-3). In 1st while loop code waits until all columns are in state high.Then in 2nd loop it waits until any button of keypad is pressed.When keypad button is pressed that column goes low.In loop, delay_ms(20) is called to wait for debounce.

Source Code

```
void main()
{
   unsigned char colloc,rowcol,a;
   int i=0;
   int j=0;
   int index;
   unsigned char keypad[4][4]={'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
   lcd_init();
  delay_ms(6);

  lcd_putc("\f WELCOME");

  port_b_pullups(TRUE);
```

```
  *trisa = 0x00;
  *trisb = 0xf0;
  *porta=0x0f;
  *portb=0x00;
  while(1)
  {

do
{
*trisa = 0x00;
///*trisd = 0xff;
*portb=0x00;
//*trisd = 0xff;
*porta =0x0f;
*trisa=0xff;

}
while(*porta!=0x0f);

   d
   o
   {

    d
    o
     {
       *trisa = 0x00;
       delay_ms(20);

       *porta &=0x0f;
       *trisa = 0xff;

     }
     while(*porta==0x0f);


       *trisa = 0x00;
       delay_ms(20);
       //colloc=*portb;
       *porta &=0x0f;
       *trisa = 0xff;

   }
   while(*porta==0x0f);

while(1)
{
```

```
*portb=0xfe;
*trisa = 0x00;
*porta &=0x0f;
*trisa = 0xff;
colloc=*porta;
if(colloc!=0x0f)
{

rowcol=0;
break;
}

*portb=0xfd;
*trisa = 0x00;
*porta &=0x0f;
*trisa = 0xff;
colloc=*porta;
if(colloc!=0x0f)
{
rowcol=1;
break;
}

//output_d(0xfb);
*portb=0xfb;
*trisa = 0x00;
*porta &=0x0f;
*trisa = 0xff;
colloc=*porta;
if(colloc!=0x0f)
{
rowcol=2;
break;
}

*portb=0xf7;
*trisa = 0x00;
*porta &=0x0f;
*trisa = 0xff;

rowcol=3;

break;
```

```
}
if(*porta==0x0e)
{
if(rowcol==0)
{
index=1;
}
else if(rowcol==1)
{
index=2;
}
else if(rowcol==2)
{
index=3;
}
else if(rowcol==3)
{
index=4;
}

}

if(*porta==0x0d)
{
if(rowcol==0)
{
index=5;
}
else if(rowcol==1)
{
index=6;
}
else if(rowcol==2)
{
index=7;
}
else if(rowcol==3)
{
index=8;
}

}

if(*porta==0x0b)
{
//a =keypad[rowcol][2];
```

```
if(rowcol==0)
{
index=9;
}
else if(rowcol==1)
{
index=10;
}
else if(rowcol==2)
{
index=11;
}
else if(rowcol==3)
{
index=12;
}
}
if(*porta==0x07)
{

if(rowcol==0)
{
index=13;
}
else if(rowcol==1)
{
index=14;
}
else if(rowcol==2)
{
index=15;
}
else if(rowcol==3)
{
index=16;
}
}


if(index==1)
{
lcd_putc("Welcome to");
}
if(index==2)
{
```

```
lcd_putc("U.E.T Lahore");
}
if(index==3)
{
lcd_putc("Department");
}
if(index==4)
{
lcd_putc("of Computer Science");
}
if(index==5)
{
lcd_putc("and Engineering");
}
if(index==6)
{
lcd_putc("Embedded System");
}
if(index==7)
{
lcd_putc("LAB");
}
if(index==8)
{
lcd_putc(" PIC16F877A");
}
if(index==9)
{
lcd_putc("PIC16F877");
}
if(index==10)
{
lcd_putc("PIC16F84A ");
}
if(index==11)
{
lcd_putc("Atmel 8051");
}
if(index==12)
{
lcd_putc("Atmel 8052");
}
if(index==13)
{
lcd_putc("PIC16F627A");
}
```

```
if(index==14)
{
lcd_putc("PIC16F628A ");
}

if(index==15)
{
lcd_putc("PIC16F630 ");
}
if(index==16)
{
lcd_putc("END");
}


}

}
```

EXPERIMENT NO 9

Encoder reading using external interrupt Through "U-Shaped Sensor".

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 9	**Dated:**
Semester:6TH	**Session:2020**
Lab: 9	**CLOs to be covered:**
Lab Title: Encoder reading using exyernal interrupt thorugh "U-shaped sensor".	**Teacher Name: Ms. Sana Tasleem**

Encoder reading using external interrupt Through "U-Shaped Sensor".

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

Rubrics for Current Lab (Optional):

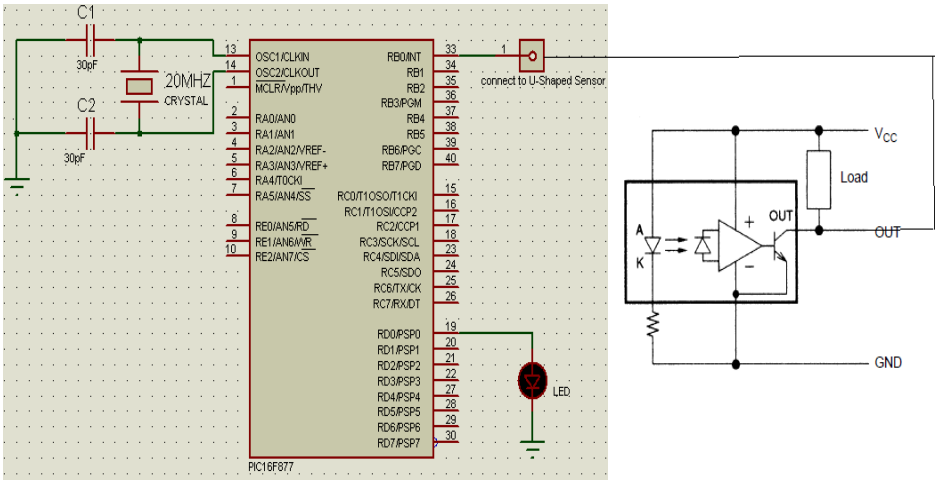
Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.

Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

- To understand the usage of external interrupts in PIC16F877A.
- To understand the working of U-Shaped Sensor.

Schematic Diagram

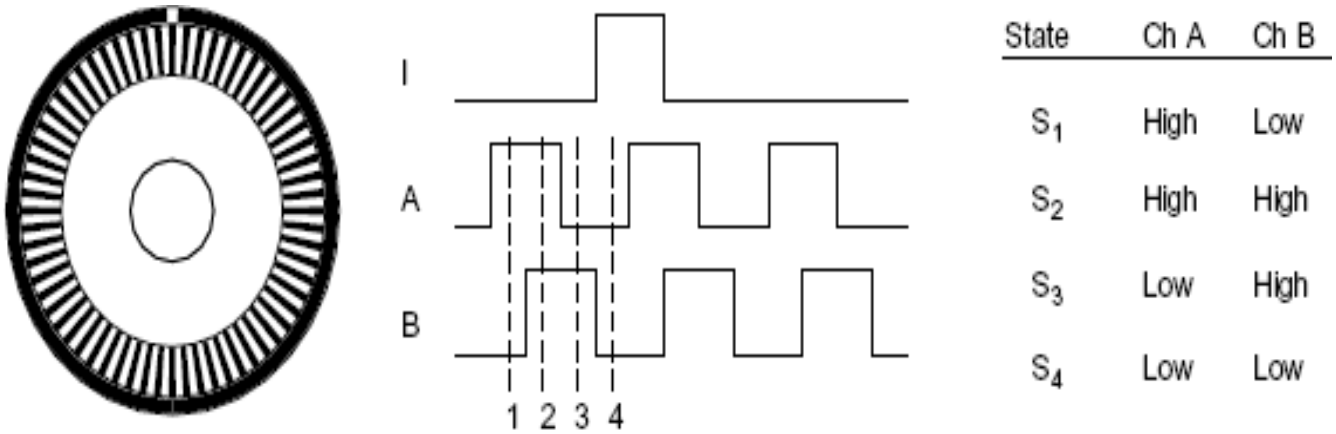


Description

Encoder

The simplest type of incremental encoder is a single-channel tachometer encoder, basically an instrumented mechanical light chopper that produces a certain number of sine or square wave pulses for each shaft revolution. Adding pulses increases the resolution (and subsequently the cost) of the unit. These relatively inexpensive devices are well suited as velocity feedback sensors in medium-to high-speed control systems, but run into noise and stability problems at extremely slow velocities dueto quantization errors.

In addition to low-speed instabilities, single-channel tachometer encoders are also incapable of detecting the direction of rotation and thus cannot be used as position sensors. Phase-quadrature incremental encoders overcome these problems by adding a second channel, displaced from the first, so the resulting pulse trains are 90^0 out of phase as shown in Fig.



State	Ch A	Ch B
S_1	High	Low
S_2	High	High
S_3	Low	High
S_4	Low	Low

The observed phase relationship between Channel A and B pulse trains can be used to determine the direction of rotation with a phase-quadrature encoder, while unique output states S_1 - S_4 allow for up to a four-fold increase in resolution. The single slot in the outer track generates one index pulse per disk rotation

Sensor

Features of Photomicrosensors

The Photomicrosensor is a compact optical sensor that senses objects or object positions with an optical beam. The transmissive Photomicrosensor and reflective Photomicrosensor are typical

Photomicrosensors. The transmissive Photomicrosensor incorporates an emitter and a transmissive that face each other as shown in Figure 1. When an object is located in the sensing position between the emitter and the detector, the object intercepts the optical beam of the emitter, thus reducing the amount of optical energy reaching the detector.

The reflective Photomicrosensor incorporates an emitter and a detector as shown in Figure 2. When an object is located in the sensing area of the reflective Photomicrosensor, the object reflects the optical beam of the emitter, thus changing the amount of optical energy reaching the detector.

"Photomicrosensor" is an OMRON product name. Generally, the Photomicrosensor is called a photointerrupter.

Figure 1. Transmissive Photomicrosensor

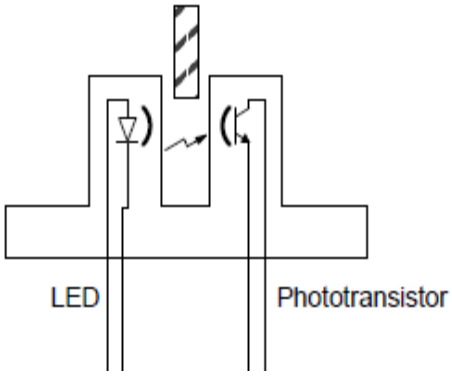
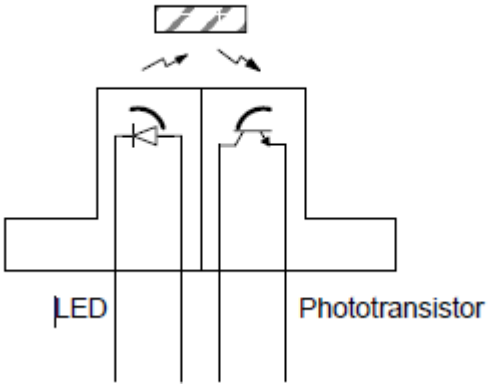


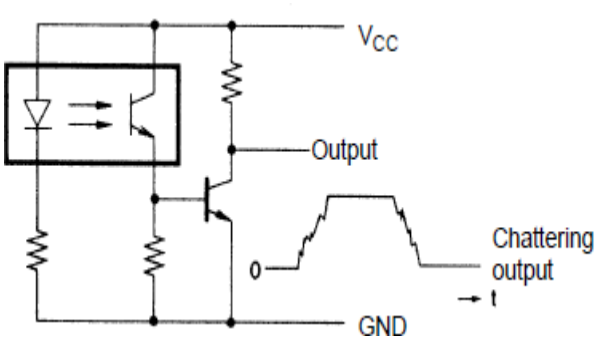
Figure 2. Reflective Photomicrosensor



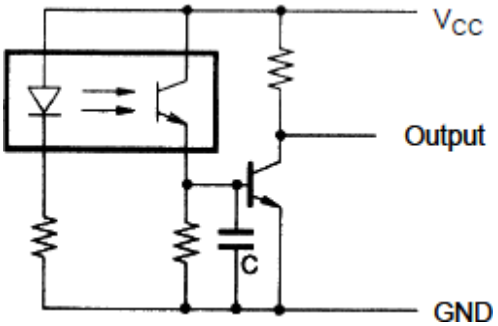
When using the transmissive Photomicrosensor to sense any object that vibrates, moves slowly, or has highly reflective edges, make sure to connect a proper circuit which processes the output

of the transmissive Photomicrosensor so that the transmissive Photomicrosensor can operate properly, otherwise the transmissive Photomicrosensor may have a chattering output signal as shown in Figure. If this signal is input to a counter, the counter will have a counting error or operate improperly. To protect against this, connect a 0.01- to 0.02-μF capacitor to the circuit as shown in Figure or connect a Schmitt trigger circuit to the circuit as shown in Figure.

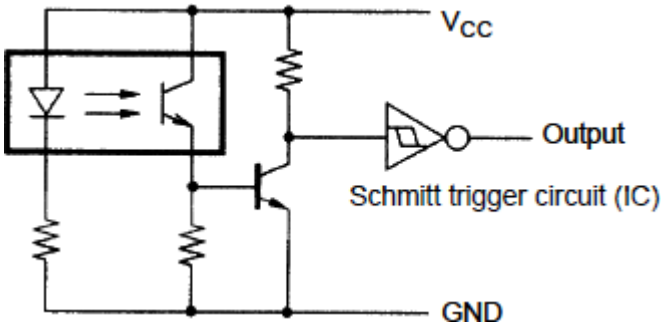
Chattering Principle



Chattering Preventation(1)



Chattering Preventation(2)



Source Code

The source code to toggle LED after every 10 interrupts is given below:

```
int count=1;

#int_EXT
EXT_isr()
{
        count++;
}




void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

 enable_interrupts(INT_EXT);        // Enable External Interrupt
 enable_interrupts(GLOBAL);         // Enable All Interrupts
 ext_int_edge( H_TO_L );            // Interrupt from High to Low

  while(1)
  {
        If(count>=10)
        {
                output_toggle(PIN_D0);
                count=1;
        }
  }
}
```

EXPERIMENT NO 10
Stepper motor control using PIC16F877A

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 10	**Dated:**
Semester:6TH	**Session:2020**
Lab: 10	**CLOs to be covered:**
Lab Title: Stepper motor control using PIC 16F877A	**Teacher Name: Ms. Sana Tasleem**

Stepper motor control using PIC16F877A

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

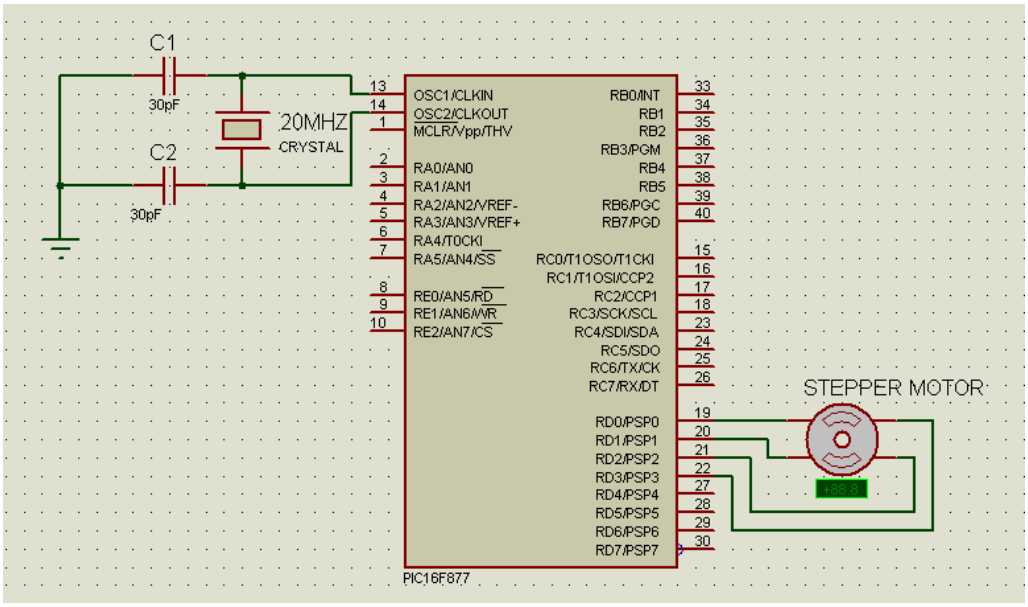
Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

Objectives

- To understand the working of stepper motor.
- To control stepper motor using microcontroller.

Schematic Diagram



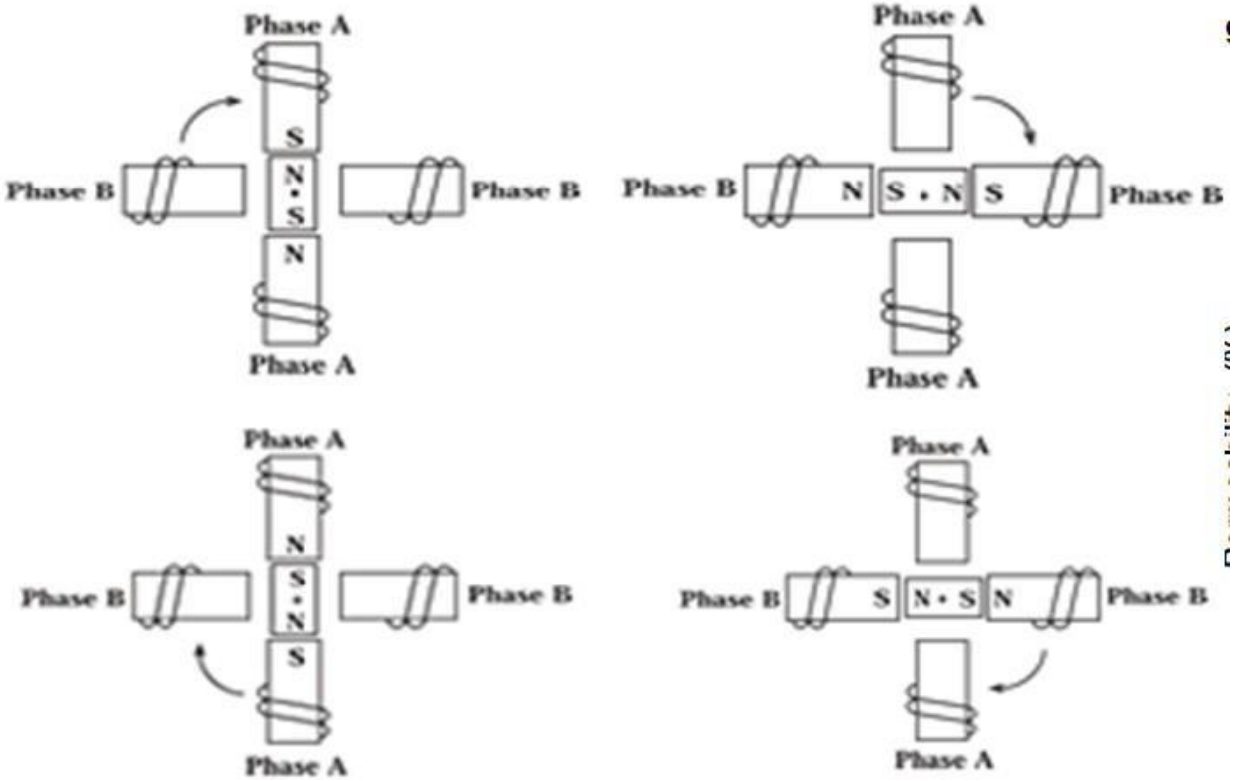
Description

Stepper Motor

The stepper motor is driven by feeding it a stream of electric pulses. These are most directly adaptable for digital controlling methods. Each pulse makes the motor rotate by a fixed angle (e.g. 1.8º). It Moves in discrete steps and is controlled with two signals step and direction.

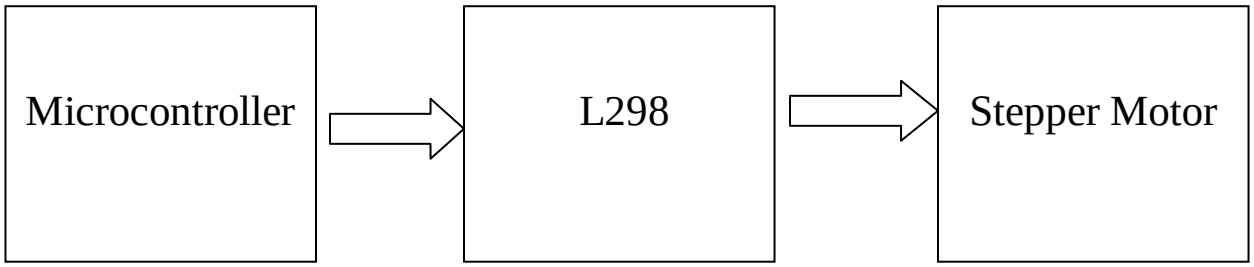
The rotor is a permanent magnet, configured so as to have a series of equally spaced (angularly) sets of poles along the circumference. The stator has a corresponding number of coils. For a required motion, only one of the stator coils is activated, causing the rotor to align its poles in opposition to the electromagnetic poles of the energized coil. Further, by activating a combination of coils, the selection of the right combination gives further resolution in the steps taken by the motor. Yet another method used by stepper motor drive units is application of pulses of different voltages to different coils. By proper selection of the voltage levels applied, smaller steps can be produced, thus making the motor more precise.

Stepper Motor with One Phase On



Note

If the ampere rating of the stepper motor is more than 500mA then it cannot be drive directly from the controller. For this purpose L298 IC must be used.



Source Code

The source code to control the stepper motor is given below:

```
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);

   while(1)
  {
        // To successfully drive the motor, you have to find the right pattern
        //  Change the delay as required.

        D0-High        D1-Low        D1-Low        D1-Low
```

```
        Delay

        D0-Low      D1-High     D1-Low      D1-Low

        Delay

        D0-Low      D1-Low      D1-High     D1-Low

        Delay

        D0-Low      D1-Low      D1-Low      D1-High

        Delay
    }
}
```

EXPERIMENT NO 11

To switch bulb on and off using relay

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 11	**Dated:**
Semester:6TH	**Session:2020**
Lab: 11	**CLOs to be covered:**
Lab Title: to switch bulb on and off using relay	**Teacher Name: Ms. Sana Tasleem**

To switch bulb on and off using relay

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

Rubrics for Current Lab (Optional):

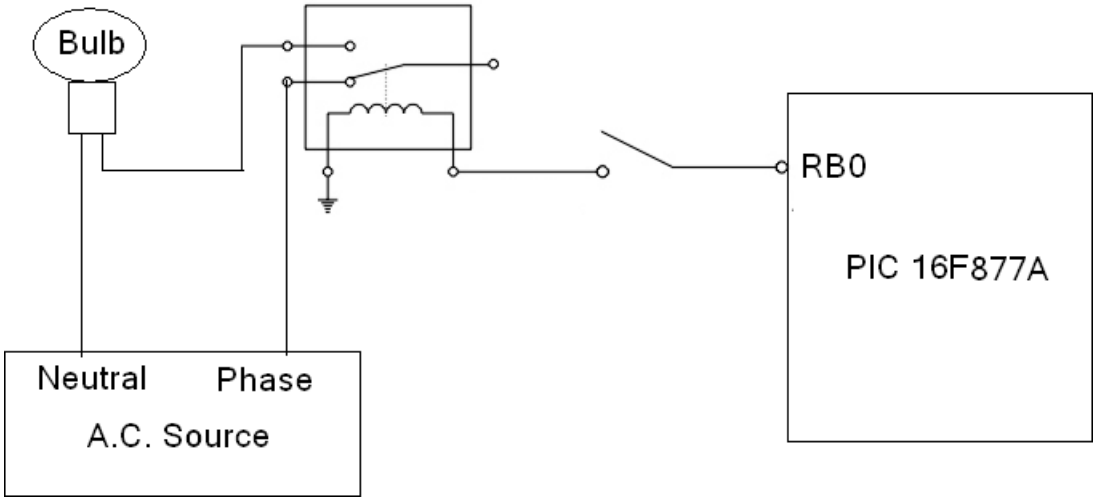
Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.

Not Acceptable	**0**	L6	Did not attempt

Objectives

- To understand basic functionality of a relay
- To control working of relay using microcontroller

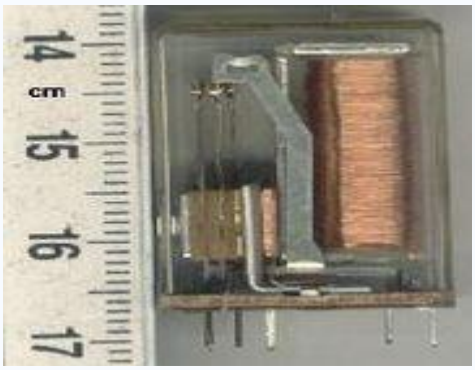
Schematic Diagram



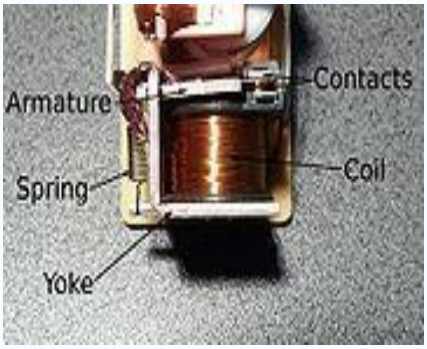
Description

A relay is an electrical switch that opens and closes under the control of another electrical circuit. In the original form, the switch is operated by an electromagnet to open or close one or many sets of contacts. It was invented by Joseph Henry in 1835. Because a relay is able to control an output circuit of higher power than the input circuit, it can be considered to be, in a broad sense, a form of an electrical amplifier.

Basic Design and Operation



Small relay as used in electronics



Simple electromechanical relay

A simple electromagnetic relay, such as the one taken from a car in the first picture, is an adaptation of an electromagnet. It consists of a coil of wire surrounding a soft iron core, an iron yoke, which provides a low reluctance path for magnetic flux, a moveable iron armature, and a set, or sets, of contacts; two in the relay pictured. The armature is hinged to the yoke and mechanically linked to a moving contact or contacts. It is held in place by a spring so that when the relay is de-energised there is an air gap in the magnetic circuit. In this condition, one of the two sets of contacts in the relay pictured is closed, and the other set is open. Other relays may have more or fewer sets of contacts depending on their function. The relay in the picture also has a wire connecting the armature to the yoke. This ensures continuity of the circuit between the moving contacts on the armature, and the circuit track on the Printed Circuit Board (PCB) via the yoke, which is soldered to the PCB.

When an electric current is passed through the coil, the resulting magnetic field attracts the armature, and the consequent movement of the movable contact or contacts either makes or breaks a connection with a fixed contact. If the set of contacts was closed when the relay was de-energised, then the movement opens the contacts and breaks the connection, and vice versa if the contacts were open. When the current to the coil is switched off, the armature is returned by a force, approximately half as strong as the magnetic force, to its relaxed position. Usually this force is provided by a spring, but gravity is also used commonly in industrial motor starters. Most relays are manufactured to operate quickly. In a low voltage application, this is to reduce noise. In a high voltage or high current application, this is to reduce arcing.

If the coil is energized with DC, a diode is frequently installed across the coil, to dissipate the energy from the collapsing magnetic field at deactivation, which would otherwise generate a voltage spike dangerous to circuit components. Some automotive relays already include that diode inside the relay case. Alternatively a contact protection network, consisting of a capacitor and resistor in series, may absorb the surge. If the coil is designed to be energized with AC, a small copper ring can be crimped to the end of the solenoid. This "shading ring" creates a small out-of-phase current, which increases the minimum pull on the armature during the AC cycle.

By analogy with the functions of the original electromagnetic device, a solid-state relay is made with a thyristor or other solid-state switching device. To achieve electrical isolation an optocoupler can be used which is a light-emitting diode (LED) coupled with a photo transistor.

Source Code

The source code using delay function to switch on and off bulb every second using relay is given below:

```
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  while(1)
  {
        output_toggle(PIN_B0);
        delay _ms(1000);

  }
}
```

EXPERIMENT NO. 12

PIC16F877A and LM35 Based Temperature Monitor

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 12	**Dated:**
Semester:6TH	**Session:2020**
Lab: 12	**CLOs to be covered:**
Lab Title: PIC 16F877A and LMS35 based temperature monitor	**Teacher Name: Ms. Sana Tasleem**

PIC16F877A and LM35 Based Temperature Monitor

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

This is a simple project showing you how to read LM35 analog temperature sensor using a PIC microcontroller and to switch a certain load at a certain temperature.

Software in use:

CCS PIC-C Compiler

Some concepts you need to Know

ADC is an internal module used to read analog voltages in the form of a digital representation, in this project I've used 16F877a which includes a 10Bit resolution ADC module having 8 channels A0-A5 and E0-E2.
One important parameter of the ADC module is it's reference voltage (Vref), which is the maximum voltage an ADC can read, in our case Vref = 5V which is the supply voltage.
Another important parameter is the ADC resolution, which determines the minimum value of analog voltage can read (ADC Step size).
For example, our ADC is 10Bit resolution with a 5V reference, the range of voltages starting at 0V and ended by 5V is to be divided into equal steps starting at 000 and ended by 1023 (2^10 - 1).
i.e. if the input voltage is 5V which is the max value, the ADC will read it as 1023, if the input was 2.5V, the reading would be 512 and so on.

The ADC step is simply calculated using the equation : Step = Vref/1024 , in our case its 4.883 mV, that's the minimum voltage our ADC can read, so:
an input of 4.883mV would give us a reading of 001
an input of 9.766mV would give us a reading of 002, and so on.

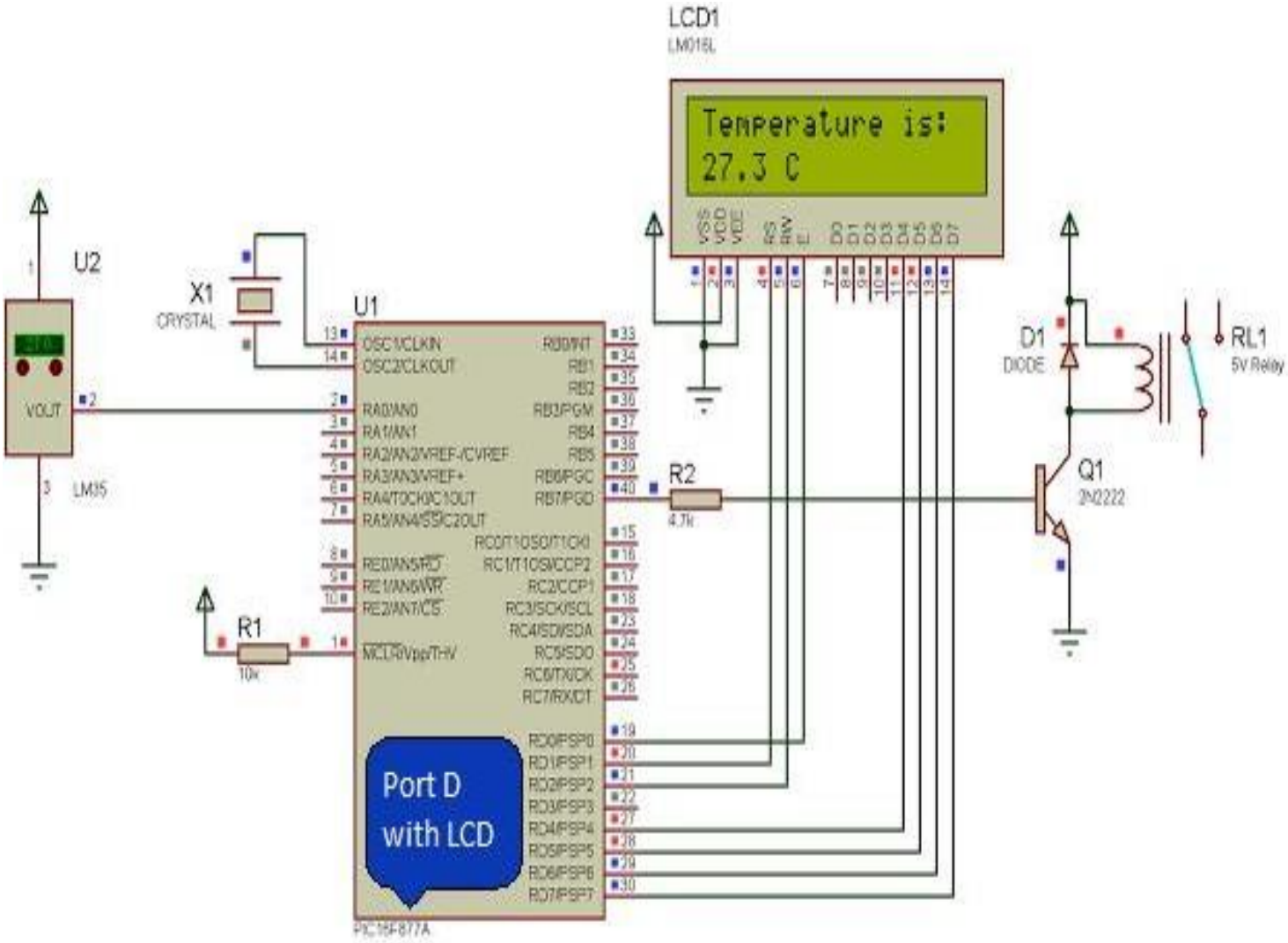
LM35 Temperature Sensor:

LM35 is a Three-Pins (Vcc,Output,GND) high precision temperature sensor having a resolution of 10mV/C starting at 0V (i.e. an output of 0V represents a temperature of 0C).
So,
10mV ---> 1C
20mV ---> 2C
370mV ---> 37.0C and so on.

Converting ADC Reading to Celsius degrees:

Knowing that our ADC has a step size of 4.883mV, converting our digital reading back to voltage is simply done by multiplying the digital reading by the step size:
Vin (in Volts) = DigitalReading * 0.004883
Now, knowing our sensor's sensitivity is 10mV/C, converting this voltage to Celsius is simply done by dividing the input voltage by 0.01, So:
Temperature (C) = Vin/0.01 = DigitalReading * 0.4883

The Schematic Design:



EXPERIMENT NO. 13

PIC16F877A and Push Button

Course Name: Embedded Systems	**Course Code: CSE 320**
Assignment Type: Lab 13	**Dated:**
Semester:6TH	**Session:2020**
Lab: 13	**CLOs to be covered:**
Lab Title: PIC 16F877A and Push Button	**Teacher Name: Ms. Sana Tasleem**

PIC16F877A and Push Button

Lab Evaluation:

Lab No:	Operation of Tools (CLO1)			Understanding of Circuit (CLO2)			Understanding of IDE and flashing procedure (CLO3)			Understanding of Firmware (CLO4)			Soldering ofCircuit on Vero board (CLO5)			Use of Instruction/Technologies to measure and debug the output (CLO6)			Practicing Safety (CLO7)		Total (20)
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	

Rubrics for Current Lab (Optional):

Scale	Marks	Level	Rubric
Excellent	**5**	L1	Execute and implement all the codes with proper explanation with graphical view if required.
Very Good	**4**	L2	Execute all codes and create an understanding level with required hardware.
Good	**3**	L3	Locally executed codes and try to fill lab report.
Basic	**2**	L4	Locally executed shared in this lab using PIC C compiler and MPLAB(IDE)
Barely Acceptable	**1**	L5	Tried to run code but could not do so. Effort can be tracked on hardware.
Not Acceptable	**0**	L6	Did not attempt

We learn how to use an output pin of PIC Microcontroller by blinking an LED with a delay of 1 second. In this tutorial we will learn how to read the status of an input pin and to make decisions according to its state. For the demonstration of its working we are connecting an LED and a Micro Switch to PIC 16F877A microcontroller. Pin connected to LED is configured as an output and pin connected to switch is configured as input. We will program in such a way that when the switch is pressed, LED will glow for 2 seconds

VDD and VSS of PIC Microcontroller is connected to 5V and GND respectively to provide necessary power for its operation. 8MHz crystal will provide clock for the operation of the microcontroller and 22pF capacitors will stabilize the oscillations produced by the crystal. Pin RD0 (PIN 19) is configured as an input pin to which switch is connected and Pin RB0 (PIN 33) is configured as an output pin to which LED is connected. A 10KΩ resistor is used along with switch to ensure that the pin RD0 is at Logic HIGH (VDD) state when the switch is not pressed. Whenever the switch is pressed pin RD0 becomes Logic LOW (VSS) state. A 470Ω resistor is used to limit the current through the LED..

Source Code

```
void main()
{
  output_low(PIN_B0);                    //LED OFF
  output_float(PIN_D0);                  //Set RD0 as Input Pin
```

```
                    //OR set_tris_x(0b00000001)
  while(TRUE)
  {
    if(input_state(PIN_D0) == 0)
    {
          delay_ms(10);                         //De-bouncing time
          if(input_state(PIN_D0) == 0)
          {
              output_high(PIN_B0);              //LED ON
              delay_ms(2000);                   //2 Second Delay
              output_low(PIN_B0);               //LED OFF
          }
     }
  }
}
```

Connecting Switch to a Microcontroller