AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

DEPARTMENT OF INFORMATION TECHNOLOGY ENGINEERING

LABORATORY MANUAL

Programming Skill Development Lab

Subject: Programming Skills Development Laboratory	Class: SE(IT)
Teaching Scheme	**Examination scheme**
Practical: *2 hrs / week*	Term Work: *25 Marks*
	Practical: *25 Marks*

Sr. No.	Description
I.	**Institute and Department Vision, Mission, Quality Policy, Quality Objectives, PEOs, POs and PSOs**
II.	**List of Experiments**
	Group A
1.	Study of Embedded C programming language (Overview, syntax, One simple program like addition of two numbers).
2.	Write an Embedded C program to add array of n numbers.
3.	Write an Embedded C program to transfer elements from one location to another for following: i) Internal to internal memory transfer ii) Internal to external memory transfer
4.	Write an Embedded C menu driven program for: i) Multiply 8-bit number by 8-bit number ii) Divide 8-bit number by 8-bit number
5	Write an Embedded C program for sorting the numbers in ascending and descending order.
	Group B
6	Write an Embedded C program to interface PIC 18FXXX with LED & blinking it using specified delay.
7	Write an Embedded C program for Timer programming ISR based buzzer on/off.

8	Write an Embedded C program for External interrupt input switch press, output at relay.
9	Write an Embedded C program for LCD interfacing with PIC 18FXXX.
	Group C
10	Write an Embedded C program for Generating PWM signal for servo motor/DC motor.
11	Write an Embedded C program for PC-to-PC serial communication using UART.
12	Write an Embedded C program for Temperature sensor interfacing using ADC & display on LCD.
	Group D:
13	Study of Arduino board and understand the OS installation process on Raspberry-pi.
14	Write simple program using Open-source prototype platform like Raspberry-Pi/Beagle board/Arduino for digital read/write using LED and switch Analog read/write using sensor and actuators.

Vision and Mission of the Institute

Vision

To create opportunities for rural students to become able engineers and technocrats through continual excellence in engineering education.

Mission

Our mission is to create self-disciplined, physically fit, mentally robust and morally strong engineers and technocrats with high degree of integrity and sense of purpose who are capable to meet challenges of ever advancing technology for the benefit of mankind and nature. We, the management, the faculty and staff, therefore promise to strive hard and commit ourselves to achieve this objective through a continuous process of learning and appreciation of needs of time.

Vision and Mission of the Department

Vision

To transfer the rural learners into competent I.T. engineers and technocrats in emerging areas of I.T. Engineering education through continual excellence for the benefit of society..

Mission

Mission of Information Technology Department is elaborated in following three mission statements.

M1: To empower the youths in rural communities to be self-disciplined, physically fit, mentally robust and morally strong I.T. professionals.

M2. To provide cutting-edge technical knowledge through continuous process in rapidly changing environment as per need of industry and surrounding world.

M3: To provide opportunities for intellectual and personal growth of individuals in rural platform using high quality Information Technology education.

AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

To train the Information Technology Engineering students-

1. To develop competent I.T. graduate with knowledge of fundamental concepts In mathematics, science, engineering and ability to provide solution to complex engineering problem by analyzing, designing and developing using modern I.T. software and tools.
2. To prepare I.T. graduate with professional skills of better communication, teamwork to manage projects in I.T. field at global level and ability to conduct investigations of complex problems using research based knowledge and research methods.
3. To develop I.T graduates with ethical practices, societal contributions through communities understanding impact of professional engineering solutions in societal and environmental context and ability of lifelong learning.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

The graduate is expected to acquire

1. Apply principles of science, mathematics along with programming paradigms and problem solving skills using appropriate tools, techniques to expedite solution in I.T. domain.
2. Demonstrate core competencies related to I.T. in domain of Data structures & algorithms, Software Engineering & Modeling, Hardware, Distributed Computing, Networking & security, Databases, Discrete mathematics & algebra, Machine Learning, Operating System.
3. Demonstrate leadership qualities and professional skills in modern I.T. platform for creating innovative carrier paths in placements, entrepreneurship and higher studies

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

- **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

QUALITY POLICY

THE AMRUTVAHINI COLLEGE OF ENGINEERING IS COMMITTED TO DEVELOP IN YOUNG MINDS THE STATE – OF – THE – ART TECHNOLOGY AND HIGH ACADEMIC AMBIENCE BY SYNERGISING SPIRITUAL VALUES AND TECHNOLOGICAL COMPETENCE CONTINUALLY IN A LEARNING ENVIRONMENT.

QUALITY OBJECTIVES

- To strive hard for academic excellence and synergizing spiritual & moral values.
- To improve overall development of student.
- To enhance industry-institute interaction.
- To provide assistance for placement & entrepreneurship development.
- To promote and encourage R&D activities.

Programming Skill Development Lab			
Experiment No: 1	Study of Embedded C programming language (Overview, syntax, One simple program like addition of two numbers).	**Page**	**1/4**

Aim: Study of Embedded C programming language (Overview, syntax, One simple program like addition of two numbers).

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXXmicrocontroller kit

Theory: Embedded C programming plays a key role to make the microcontroller run & perform the preferred actions. At present, we normally utilize several electronic devices like mobile phones, washing machines, security systems, refrigerators, digital cameras, etc. The controlling of these embedded devices can be done with the help of an embedded C program. For example, in a digital camera, if we press a camera button to capture a photo then the microcontroller will execute the required function to click the image as well as to store it.
Embedded C programming builds with a set of functions where every function is a set of statements that are utilized to execute some particular tasks. Both the embedded C and C languages are the same and implemented through some fundamental elements like a variable, character set, keywords, data types, declaration of variables, expressions, statements. All these elements play a key role while writing an embedded C program.

In embedded system programming C code is preferred over other language. Due to the following reasons:

- Easy to understand
- High Reliability
- Portability
- Scalability

Steps to Build an Embedded C Program
There are different steps involved in designing an embedded c program like the following.

- Multiline Comments . . .... Denoted using /*……*/
- Single Line Comments . .... Denoted using //
- Preprocessor Directives . . . . . #include<…> or #define
- Global Variables . . . . . Accessible anywhere in the program
- Function Declarations . . . . . Declaring Function
- Main Function . . . . . Main Function, execution begins here
  {
  Local Variables . . . . . Variables confined to main function
  Function Calls . . . . . Calling other Functions
  Infinite Loop . . . . . Like while(1) or for(;;)
  Statements . . . . .

```
      ….
      ….
      }
   o  Function Definitions . . . . . Defining the Functions
      {
      Local Variables . . . . . Local Variables confined to this Function
      Statements . . . . .
      ….
      ….
      }
```

Main Factors of Embedded C Program

The main factors to be considered while choosing the programming language for developing an embedded system include the following.

Program Size

Every programming language occupies some memory where embedded processor like microcontroller includes an extremely less amount of random-access memory.

Speed of the Program

The programming language should be very fast, so should run as quickly as possible. The speed of embedded hardware should not be reduced because of the slow-running software.

Portability

For the different embedded processors, the compilation of similar programs can be done.

- Simple Implementation
- Simple Maintenance
- Readability

Advantages

- It is very simple to understand.
- It executes simply a single task at once
- The cost of the hardware used in the embedded c is typically so much low.
- The applications of embedded are extremely appropriate in industries.
- It takes less time to develop an application program.
- It reduces the complexity of the program.
- It is easy to verify and understand.
- It is portable from one controller to another.

Disadvantages

- At a time, it executes only one task but can't execute the multi-tasks
- If we change the program then need to change the hardware as well
- It supports only the hardware system.
- It has a scalability issue
- It has a restriction like limited memory otherwise compatibility of the computer.

Applications of Embedded C Program

- Embedded C programming is used in industries for different purposes
- The programming language used in the applications is speed checker on the highway, controlling of traffic lights, controlling of street lights, tracking the vehicle, artificial intelligence, home automation, and auto intensity control.

What is MPLAB?

MPLAB is a software program that runs on your PC to provide a development environment for your embedded system design. In other words, it is a program package that makes writing and developing a program easier. It could best be described as developing environment for a standard program language that is intended for programming microcontrollers.

Get started to **MPLAB X Programming IDE.**

Step1: Creating a new project

- ➢ Go to the File Tab.
- ➢ Click on New Project.
- ➢ Step1: Choose Project:
- ➢ Select: Microchip Embedded -> Standalone Project. Click Next.

Step2: Select Device:

- ➢ Select: Family -> Advanced 8 Bit MCU (PIC18).
- ➢ Select: Device: PIC18F4550. Click Next.

Step3: Select Tool: Simulator. Click Next.

Step4: Select Compiler ->XC8. Click Next.

Step5: Select Project Name and Folder.

- ➢ Give Project Name.
- ➢ Select project Location using Browse Button.
- ➢ Uncheck Set as main project option.
- ➢ Click Finish.

Step6: Creating a new Source file and Header File.

- ➢ Go to the Project location in the Project window.
- ➢ Click the + sign to open the project space.
- ➢ Right Click on the Source Files folder (for a C file) and Header files (for a .h file).
- ➢ New - > C Source file / or C Header File.

Step7: Opening an existing project.

- ➢ Go to the File Tab.
- ➢ Select Open Project.

  Browse to the location and select the project name.X file (project file). Click on Open Project

Input: Two numbers

Output: Addition of given numbers

Conclusion: Based on the understanding of the basic concepts of Embedded C Programming along with its environment setup used in a simple program for addition of two numbers.

Programming Skill Development Lab			
Experiment No: 2	Write an Embedded C program to add array of n numbers.	**Page**	**1/2**

Aim: Write an Embedded C program to add array of n numbers.

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXXmicrocontroller kit

Theory: There is actually not much difference between C and Embedded C apart from few extensions and the operating environment. Both C and Embedded C are ISO Standards that have almost same syntax, datatypes, functions, etc.

Embedded C is basically an extension to the Standard C Programming Language with additional features like Addressing I/O, multiple memory addressing and fixed-point arithmetic, etc. C Programming Language is generally used for developing desktop applications, whereas Embedded C is used in the development of Microcontroller based applications.

C arrays are declared in the following form type name [number of elements];

For example, if we want an array of five integers, we write in C:
int numbers[5];
For a five character array
char letters[5];
type name [number of elements]={comma-separated values}
For example, if we want to initialize an array with five integers, with 1, 3, 5, 0, 9, as the initial values: int number[5]={1,3,5,0,9};

Let's see the logic to calculate the sum of the array elements. Suppose **arr** is an integer array of size N (arr[N]), the task is to write the C Program to sum the elements of an array.

Logic to calculate the sum of the array elements:

1. Create an intermediate variable 'sum'.

2. Initialize the variable 'sum' with 0.

3. To find the sum of all elements, iterate through each element, and add the current element to the sum.

//Logic within the loop
sum = sum + arr[i];

Program:
```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>
void main(void)
 {

   int i,sum,n;
   int number[] = {1,2,3,4,5,6,7,8,9,10};    // array of 10 numbers
   sum = 0;                              // initialize sum as zero
     for(i=0; i<=9;i++)
          {                   //indexing start from 0 to 9
      sum = sum+number[i];
         }

   TRISB =0;                             //initialize Port_B as output
   PORTB = sum;                             // from sum to PORT_B

   //n = 0xFF + 0XFF;
}
```

Input: Array of size N (arr[N])

Output: Sum of the elements of an array

Conclusion: We have implemented sum of elements of an array using embedded C programming.

Programming Skill Development Lab			
Experiment No: 3	Write an Embedded C program to transfer elements from one location to another for following: i) Internal to internal memory transfer ii) Internal to external memory transfer	Page	1/3

Aim: Write an Embedded C program to transfer elements from one location to another

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXXmicrocontroller kit

Theory: Memory types in microcontrollers Architecture
The microcontrollers units (MCUs) consist of three types of memory.

1. Program Memory
2. Data Memory
3. Data EEPOM

Program Memory type
This is common which have all the microcontroller and its purposes is to store the instructions.it consist of further four different types of memory.

1. ROM (Read only memory)
2. EPROM (Erasable programmable read only memory)
3. OTP (On time programmable)
4. FLASH EEPROM (Electrical erasable programmable read only memory)

ROM
In microcontrollers first type memory is ROM and during the manufacturing process once the program codes are set in ROM that can't be changed after the manufacturing process, therefore it is called read only memory mean just read the code but can't be changed. Due to this reason the microcontrollers which have the ROM memory are considers best for that applications where there is no need of program change only need of program read. These microcontrollers are less expensive as compared to the microcontrollers which have the OTP or FLAS programmable memory and these are ordered in large quantities.

EPROM
The second type is erasable programmable read only and this is used in two different type of packages. When EPROM is used in ceramic package with quartz window then microcontroller can be erased the program many times by using ultraviolet eraser and erase time depends upon the intensity of light. Normally the erase time is in between 5 and 30 minutes. In this the microcontroller can also reprogrammed the program. It is very expensive due to the high cost of the windowed ceramic package.

OTP

The one-time programmable memory used the same type of die as the EPROP windowed packaged devices. Its packaging makes it unique. These microcontrollers are in an opaque plastic packing and its program can't be erase through ultraviolet light. The OTP devices are first transfer to customer side then these are programmed therefore these devices are called one time programmable.

Flash EPROM

This the type which provides the alternate flexibility because its program can be erased electrically and also reprogram in few seconds. It's no need of any ultraviolet light to erase the program. Once the program is erased the program can reprogram with new code. The devices which have the flash memory can also be self-program by using some special sequence of instructions. These devices also contain a small amount non-volatile data EPROM and that can be written thousands of time. In these devices "F" is denoted by part number

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>
/*
 *
 */
void main(void)
{

  int i;
  int source1[] = {0x21,0x22,0x23,0x24,0x25};  // source mem block
  int dest[] = {0x00,0x00,0x00,0x00,0x00};     // destination mem block

  for(i=0; i<=4;i++)
        {                    // counter = 5

    dest[i] = source1[i];                   // source to destination

         }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>
void main(void)
 {

   int temp,i;
   int source1[] = {0x21,0x22,0x23,0x24,0x25};
   int dest[] = {0x99,0x99,0x99,0x99,0x99};
   for(i=0;  i<=4;i++)
          {
     temp = source1[i];
     source1[i] = dest[i];
     dest[i] = temp;

           }
}
```

Input: Elements in a array.

Output: Array after transferred elements

Conclusion: We have implemented transfer of elements from one location to another.

Aim: Write an Embedded C menu driven program for:
i) Multiply 8-bit number by 8-bit number
ii) Divide 8-bit number by 8-bit number

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXXmicrocontroller kit

Theory: Embedded C Programming Language, which is widely used in the development of Embedded Systems, is an extension of C Program Language. The Embedded C Programming Language uses the same syntax and semantics of the C Programming Language like main function, declaration of datatypes, defining variables, loops, functions, statements, etc.

The extension in Embedded C from standard C Programming Language include I/O Hardware Addressing, fixed point arithmetic operations, accessing address spaces, etc.

C examples – with standard arithmetic operators

int i, j, k; // 32-bit signed integers

uint8_t m,n,p; // 8-bit unsigned numbers

i = j + k; // add 32-bit integers

m = n - 5; // subtract 8-bit numbers

j = i * k; // multiply 32-bit integers

m = n / p; // quotient of 8-bit divide

m = n % p; // remainder of 8-bit divide

i = (j + k) * (i – 2); //arithmetic expression

*, /, % are higher in precedence than +, - (higher precedence applied 1st)

Example: j * k + m / n = (j * k) + (m / n)

Program:

```
void main(void)
{
  //static int v_mem[] = 0x55;      @0x0005
  int num1, num2;
  int result,i;

  result = 0;
  num1 = 0x23;
```

```
   num2 = 0x10;

   for(i=1; i<=num2; i++)
   {
      result = result + num1;
   }

   TRISB =0;
   PORTB = result;

}
```

Input: 8-bit number stored in internal memory locations.

Output: Result of 8- bit by 8-bit multiplication and division stored at internal memory locations

Conclusion: Arithmetic operations like multiplication and division of 8-bit number with 8-bit number are studied here

Aim: Write an Embedded C program for sorting the numbers in ascending and descending order.

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXX microcontroller kit

Theory: ALGORITHM:

```
Step I: Initialize the number of elements counter.
Step II: Initialize the number of comparisons counter.
Step III: Compare the elements. If first element < second element goto Step VIII Else go to step V.
Step IV: Swap the elements.
Step V: Decrement the comparison counter.
Step VI: Is count = 0? if yes go to step VIII else go to step IV.
Step VII: Insert the number in proper position.
Step VIII: Increment the number of elements counter.
Step IX: Is count = N? If yes, go to step XI else go to step II
Step X: Store the result.
Step XI: Stop
```

Program:
```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>
void main(void)
 {
   int i,j,temp;
   int num_asc[] = {10,2,5,1,6};

   for(i=0; i<=4; i++)
         {              // point to LHS number
      for(j=i+1;j<=4;j++)              // point to RHS number
      if (num_asc[i] > num_asc[j])
                 {    // if LHS > RHS , change the position
         temp = num_asc[i];
         num_asc[i] =num_asc[j];
         num_asc[j]= temp;
                 }
         }
```

}

Input: Numbers in a array

Output: Sorted array of numbers in ascending or descending order.

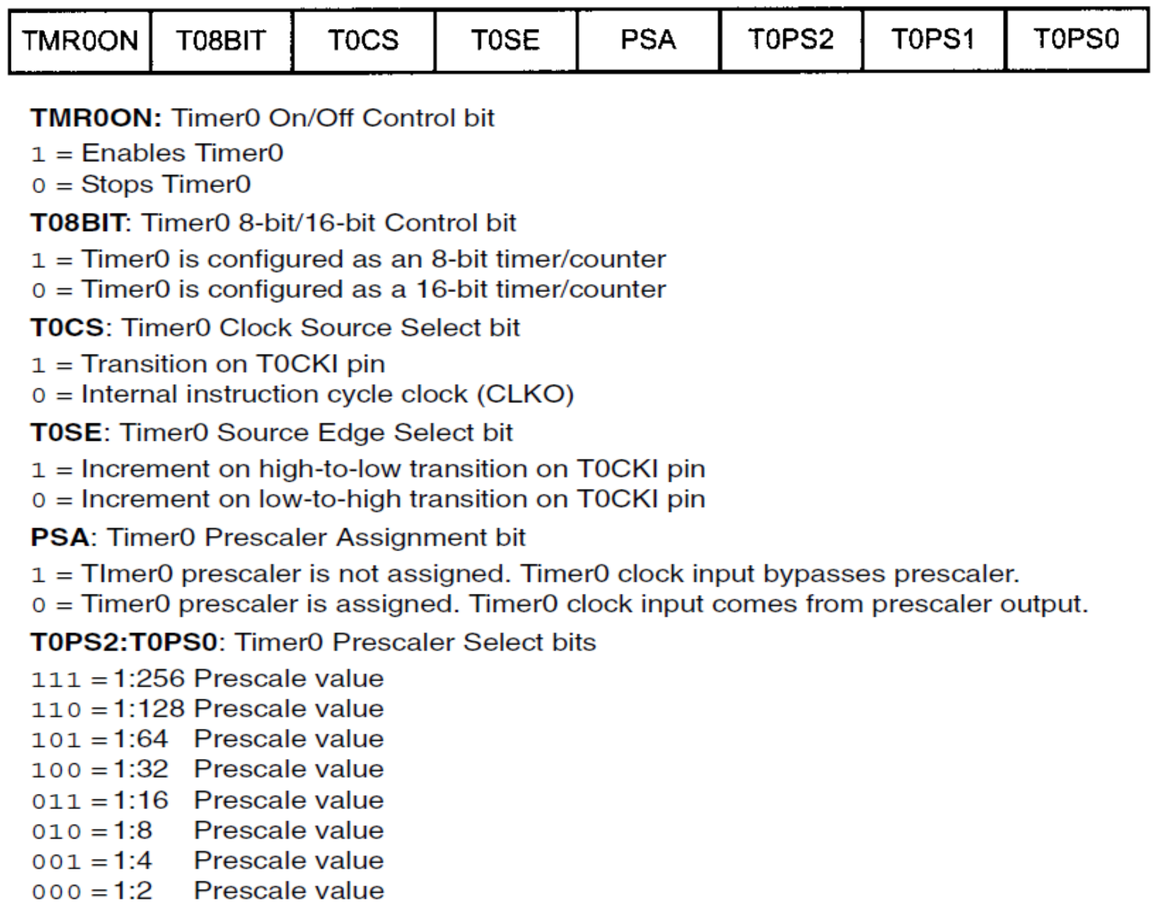
Conclusion: We have implemented sorted array of numbers in ascending or descending order.

Programming Skill Development Lab			
Experiment No: 6	**GROUP B: ASSIGNMENTS** Write an Embedded C program to interface PIC 18FXXX with LED & blinking it using specified delay.	Page	1/3

Aim: Write an Embedded C program to interface PIC 18FXXX with LED & blinking it using specified delay.

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXX microcontroller kit, Proteus, LED

Theory: Timer is used to control the output on a bit of a port.
There are 4 timers in PIC18. Each have corresponding timer registers TMRxH and TMRxL where x is the timer number that ranges from 0 to 3
There is a control register correspond to every timer TxCON. In this program Timer 0 is used. Following is the T0CON register.



TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

TMR0ON: Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

T08BIT: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

T0CS: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

T0SE: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

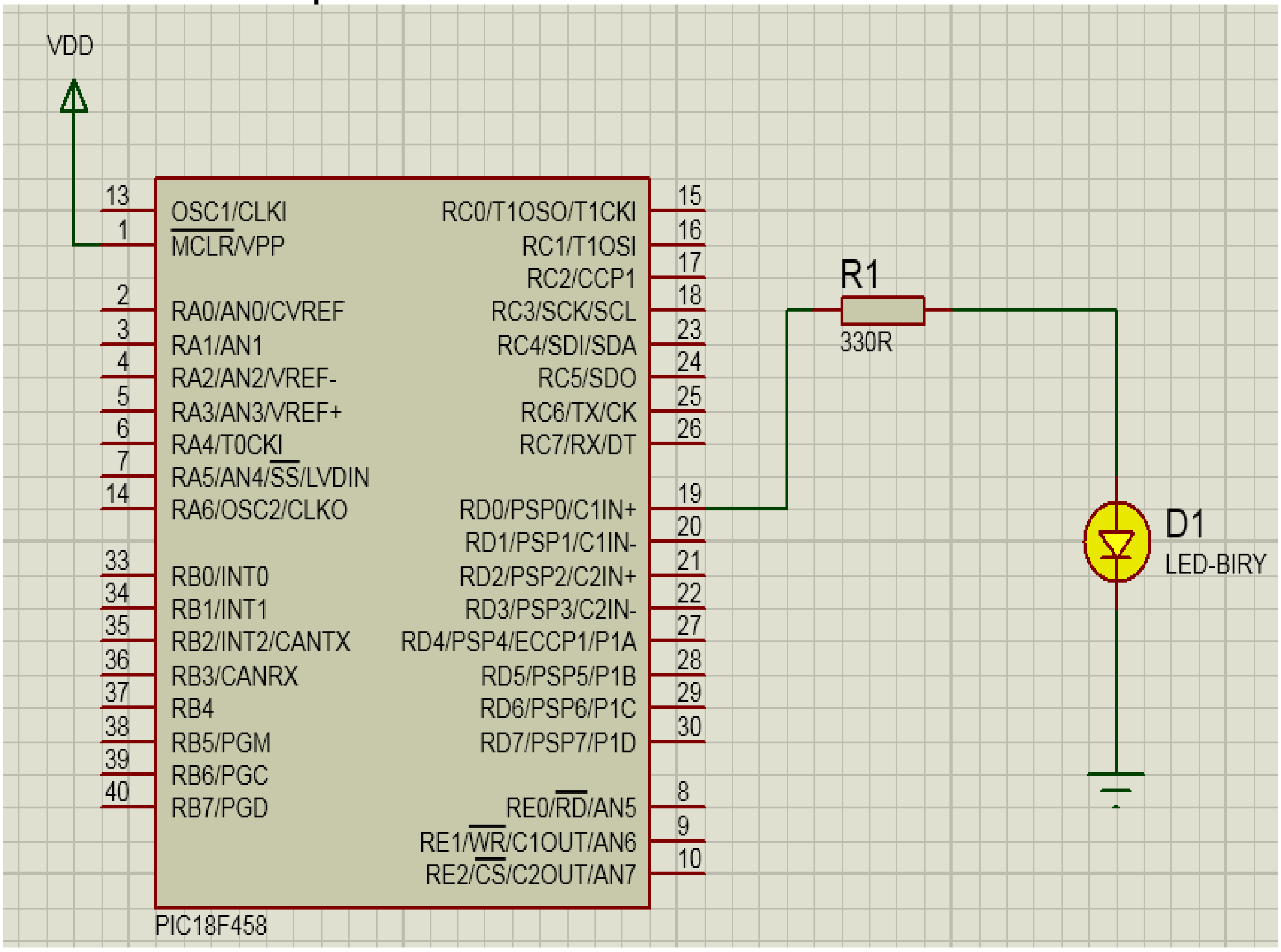
PSA: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

T0PS2:T0PS0: Timer0 Prescaler Select bits

111 =	1:256	Prescale value
110 =	1:128	Prescale value
101 =	1:64	Prescale value
100 =	1:32	Prescale value
011 =	1:16	Prescale value
010 =	1:8	Prescale value
001 =	1:4	Prescale value
000 =	1:2	Prescale value

The program is written such that a LED is connected to a pin of any port (PORTB) and that pin is toggled after a cycle controlled by the timer. This is repeated for infinite time. Thus when the port bit is 0, LED is off and when it is 1, LED is on. So the LED blinks after every cycle.

Hardware/Software Setup:



ALGORITHM:

Step1: Reset TRISB so that PORTB is in output mode

Step2: Reset PORTB

Step3: Call the delay

Step4: Set PORTB

Step5: goto Step3

Program:

```
#include<xc.h>
#include<plib/delays.h>
#include "Config.h"
#ifndef XTAL_FREQ
#define _XTAL_FREQ 20000000
#endif
void delay_ms(unsigned char t);
int main()
{
   TRISD = 0;
   while(1)
   {
      PORTD = 0xFF;
      delay_ms(100);
      PORTD = 0x00;
      delay_ms(100);
   }
}
void delay_ms(unsigned char t)
{
   int i;
   for(i=0;i<t;i++)
      Delay1KTCYx(5);
}
```

Input: Make all the required connections

Output: LED starts blinking with an equal interval

Conclusion: The student is able to understand the working of the timer, LED, and ports.

Programming Skill Development Lab			
Experiment No: 7	**GROUP B: ASSIGNMENTS** Write an Embedded C program for Timer programming ISR based buzzer on/off.	**Page**	**1/2**

Aim: Write an Embedded C program for Timer programming ISR based buzzer on/off.

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXX microcontroller kit, Proteus, Buzzer

Theory: Timer is used to generate the delay for which the buzzer should be on.
In this program Timer 1 is used. Following is the T1CON register.

RD16	--	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
D7	D6 not used	D5	D4	D3	D2	D1	D0

TMR1ON D0 Timer1 ON and OFF control bit
1 = Enable (start) Timer1
0 = Stop Timer1

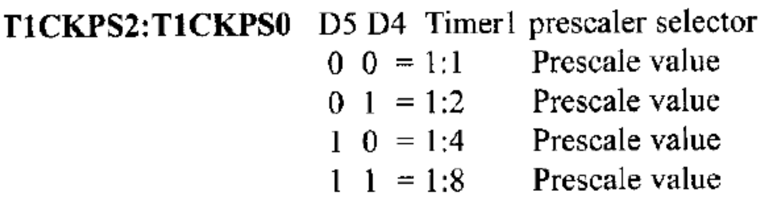
TMR1CS D1 Timer1 clock source select bit
1 = External clock from pin RC0/T1CKI
0 = Internal clock (Fosc/4 from XTAL)

T1SYNC D2 Timer1 synchronization
(used only when TMR1CS = 1 for counter mode
to synchronize external clock input)

If TMR1CS = 0 this bit is not used.

1 = Do not synchronize external clock input
0 = Synchronize external clock input

T1OSCEN D3 Timer1 oscillator enable bit
1 = Timer1 oscillator is enabled.
0 = Timer1 oscillator is shutoff.



T1CKPS2:T1CKPS0	D5 D4	Timer1 prescaler selector	
	0 0	= 1:1	Prescale value
	0 1	= 1:2	Prescale value
	1 0	= 1:4	Prescale value
	1 1	= 1:8	Prescale value

The program is written such that a buzzer is connected to a pin of any port (PORTB) and that pin is set when the switch is pressed. The duration for which the pin should be set is controlled by the timer. This is repeated for infinite time.

ALGORITHM:

```
Step1: Reset TRISB so that PORTB is in output mode
Step2: Reset PORTB
Step3: check the switch is pressed
Step4: if yes then
        Step5: Set PORTB
        Step6: Call the delay
Step7:end if
Step8: goto Step3
```

Input: : Make all the required connections

Output: The buzzer is on when the switch is pressed.

Conclusion: The student is able to understand the working of the timer, buzzer, and ports.

Programming Skill Development Lab			
Experiment No: 8	**GROUP B: ASSIGNMENTS** Write an Embedded C program for External interrupt input switch press, output at relay.	**Page**	**1/4**

Aim: Write an Embedded C program for External interrupt input switch press, output at relay.

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXX microcontroller kit

Theory: Sometimes External devices are connected with **microcontroller**. If that external device has to send some information to microcontroller, then microcontroller needs to know about this situation to get that information. An example of such an external device is the **digital thermometer**. It measures the **temperature** and at the end of measurements transmits results to the microcontroller. Now the purpose of this article to explain the fact that how does the microcontroller knows to get the required information from an external device.

Types of interrupts

There are two methods of communication between the microcontroller and the external device:

- By using Polling
- By using Interrupts

INTERRUPTS

Interrupt is the signal which is sent to the microcontroller to mark the event that requires immediate attention. This signal **requests** the microcontroller to stop to perform the current program temporarily **time** to execute a special code. It means when external device finishes the task imposed on it, the microcontroller will be notified that it can access and receive the information and use it.

INTERRUPT SOURCES in microcontrollers

The request to the microcontroller to stop to perform the current program temporarily can come from various sources:

- Through external hardware devices like pressing specific key on the keyboard, which sends Interrupt to the microcontroller to read the information of the pressed key.
- During execution of the program, the microcontroller can also send interrupts to itself to report an error in the code. For example, division by 0 will causes an interrupt.
- In the multi-processor system, the microcontrollers can send interrupts to each other to communicate. For example, to divide the work between them they will send signals between them.

INTERRUPT TYPES in pic microcontrollers

There are 2 types of interrupts for PIC microcontroller that can cause break.

Software Interrupt: It comes from a program that is executed by microcontroller or we can say that it is generated by internal peripherals of the microcontroller and requests the processor to hold the running of program and go to make an interrupt.
Hardware Interrupt: These interrupts are sent by external hardware devices at certain pins of microcontroller.
Following interrupts sources are present in PIC18F452

- **Timer over interrupt**
- Pins RB0, RB1, RB2 for external hardware interrupts (INT0, INT1, INT2)
- PORTB Change interrupts (any one of the upper four Port B pins. RB4-RB7)
- ADC (**analog-to-digital converter**) Interrupt
- CCP (compare capture pulse-width-modulation) Interrupt
- **Serial communication's USART** interrupts (receive and transmit)
- Reset, Brown-Out Reset, Watch-dog Reset, Power On Reset
- Parallel Port Read/Write Interrupt
- Master Synchronous Serial Port Interrupt
- Data **EEPROM** Write Complete Interrupt

REGISTER CONFIGURATION for external interrupt
These are the registers for interrupt operation and minimum 1 register can be used to control the interrupt operation in PIC18F452 which are:

- RCON (Reset Control Register)
- INTCON, INTCON2, INTCON3 (Interrupt Control Registers)
- PIR1, PIR2 (Peripheral Interrupt Request Registers)
- PIE1, PIE2 (Peripheral Interrupt Enable Registers)

RCON Register:
- Reset control register
- IPEN bit to enable interrupt priority scheme, 1= enable priority level on interrupts
- Other bits used to indicate the cause of reset
  RI (Reset Instruction flag), TO (Watchdog Time Out flag), PD (Power on Detection flag), POR (Power on Reset status) and BOR (Brown Out Reset status bit)

INTCON Register:
- 3 Interrupt control registers INTCON, INTCON2, INTCON3
- Readable and writable register which contains various enable and flag bits
- Interrupt flag bits get set when an interrupt condition occurs
- Contain enable, priority and flag bits for external interrupt, port B pin change and TMR0 overflow interrupt

PIE Register:
- Peripheral Interrupt Enable register
- May be multiple register (PIE1, PIE2), depending on the number of peripheral interrupt sources
- Contain the individual bits to enable/disable Peripheral interrupts for use

PIR Register:
- Peripheral Interrupt Flag register
- May be multiple register (PIR1, PIR2), depending on the number of peripheral interrupt sources
- Contain bits to identify which interrupt occurs (flags)

- Corresponding bits are set when the interrupt occurred

EXTERNAL INTERRUPT registers setting

INTCON registers are just used to configure the external PIC interrupts.

INTCON REGISTER:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE	T0IE	INTE	RAIE	T0IF	INTF	RAIF
bit 7							bit 0

GIE: Global Interrupt Enable
This bit is set high to enable all interrupts of PIC18F452.
1 = Enable all interrupts
0 = Disable all interrupts

PEIE: Peripheral Interrupt Enable
This bit is set high to enable all the peripheral interrupts (Internal interrupts) of the microcontroller.
1 = Enable all peripheral interrupts
0 = Disable all peripheral interrupts

T0IE: TMR0 Overflow Interrupt Enable
This bit is set high to enable the External Interrupt 0.
1 = Enable TMR0 overflow interrupt
0 = Disable TMR0 overflow interrupt

INTE: INT External Interrupt Enable
This bit is set high to enable the external interrupts.
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

RBIE: RB Interrupt Enable
This bit is set high to enable the RB Port Change interrupt pin.
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt
T0IF: TMR0 Overflow Interrupt Flag
1 = TMR0 register has overflowed (it must be cleared in software)
0 = TMR0 register has not overflowed

INTF: INT External Interrupt Flag

1 = The INT external interrupt occurred (it must be cleared in software)
0 = The INT external interrupt did not occur
RBIF: RB Port Change Interrupt Flag
1 = At least one of the RB7:RB4 pins changed the state (must be cleared in software)
0 = None of RB7:RB4 pins have changed the state

ALGORITHM:

In this program, we configure the External Interrupt 0 (INT0). PORT D is used as output port and it is monitored through a set of 8 LEDs. Reset is given through pin 1. Push button is connected to RB0 for external interrupt source.

Step1: Enable the External Interrupt 0 by setting INT0IE bit high (INTCON=0x10).
Step2: Set the interrupt on falling edge by setting the INTEDG0 in INTCON2 to zero (INTCON2=0)
Step3: Set the Global Interrupt Enable in INTCON to high (INTCON.GIE=1)
Step4: Initialize PORTD with certain value (LATD=0xAA)
Step5: Write the ISR (Interrupt Service Routine) for the interrupt
Step6: Clear the INT0IF bit of INTCON (INTCON.INT0IF=0)
Step7: Invert or toggle the value at PORTD (LATD=~LATD)
Step8: go to step3.

Input: Press the push button

Output: The LED's connected to the PORTD are toggled when the button is pressed.

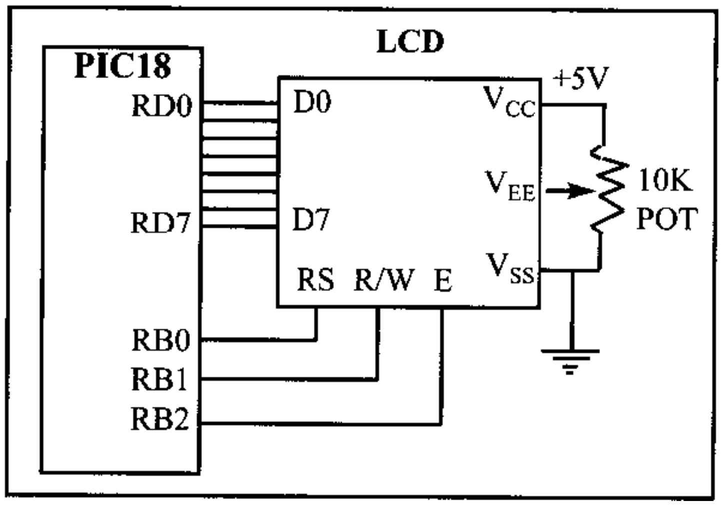
Conclusion: The student is able to understand the working of the interrupts and able to write the ISR (Interrupt Service Routine).

Programming Skill Development Lab			
Experiment No: 9	**GROUP B: ASSIGNMENTS** Write an Embedded C program for LCD interfacing with PIC 18FXXX.	**Page**	**1/3**

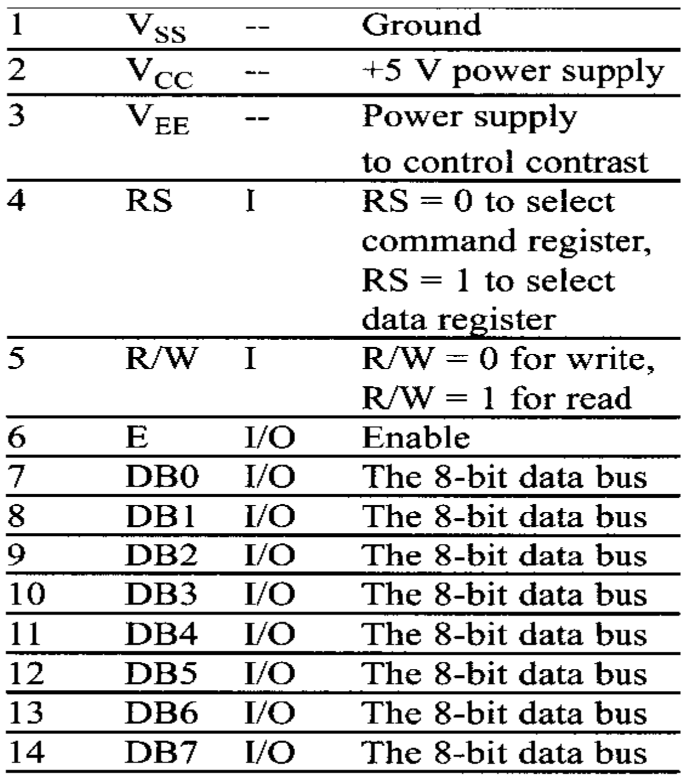
Aim: Write an Embedded C program for LCD interfacing with PIC 18FXXX.

Apparatus: : MPLAB X IDE simulator, XC8 Compiler, PIC18FXXX microcontroller kit, LCD, Proteus

Theory:

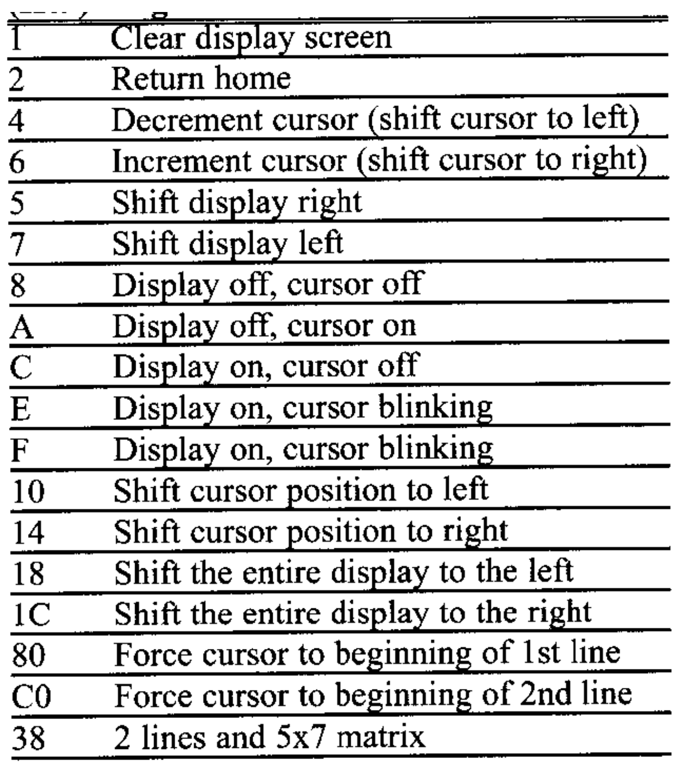


The above figure shows the connection of LCD with PIC18.
The resistor R1 is used for giving the contrast to the LCD. The crystal oscillator of 12 MHz is connected to the OSC1 and OSC2 pins of Pic microcontroller PIC18F4550 for system clock. The capacitor C2 and C3 will act filters to the crystal oscillator. You can use different ports or pins for interfacing the LCD before going to different ports please check the data sheet whether the pins for general purpose or they are special function pins.



1	V_{SS}	--	Ground
2	V_{CC}	--	+5 V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

The above figure shows pin description of LCD.



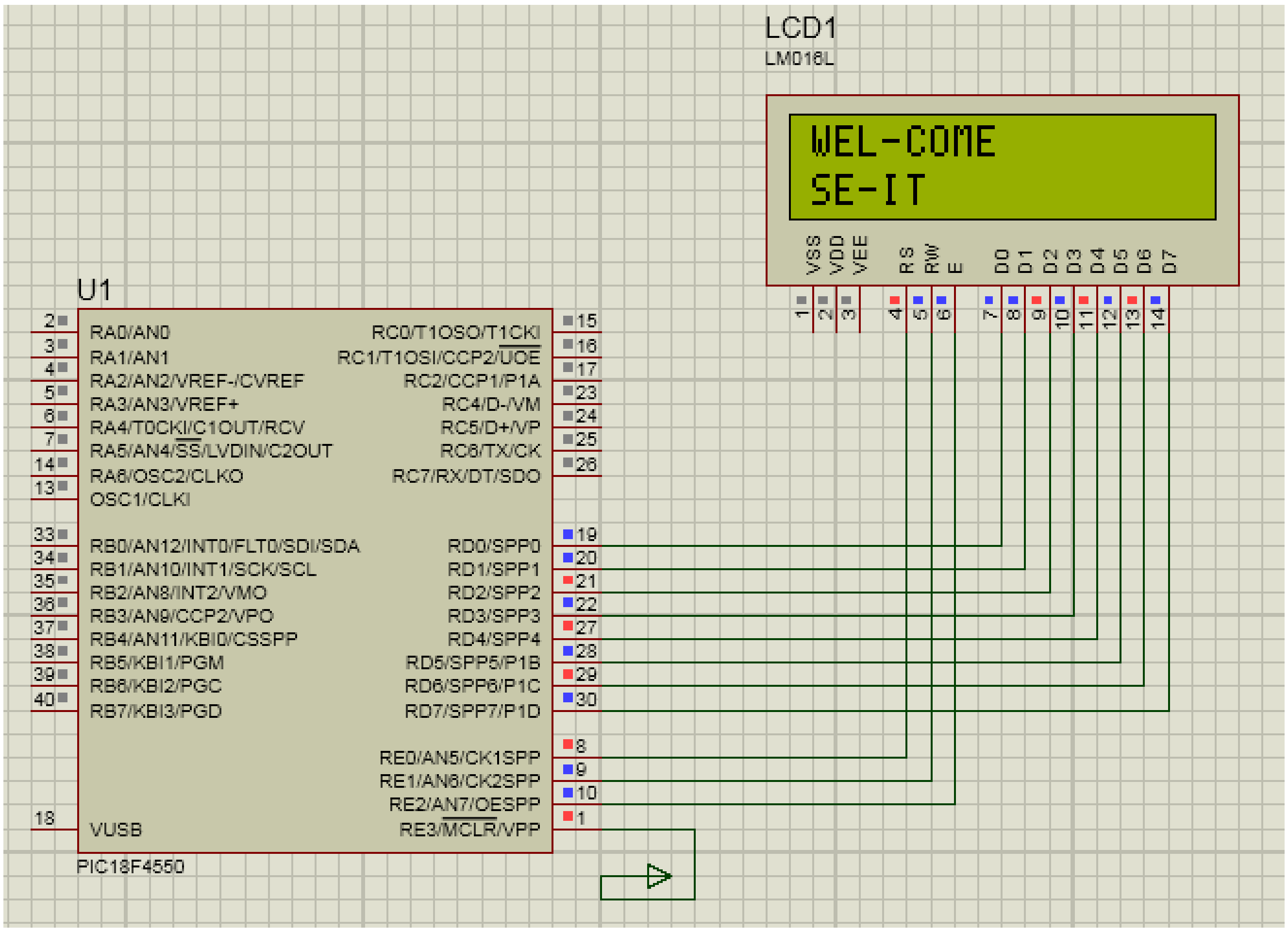
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

The above figure shows the hex codes and their corresponding instruction to LCD register. The characters are sent ot the LCD without checking the busy flag. We need to wait 5-10 ms between issuing each character to the LCD. In programming an LCD we need a long delay for the power-up process.

Initializing the LCD function:

lcdcmd(0x38);//Configure the LCD in 8-bit mode,2 line and 5×7 font
lcdcmd(0x0C);// Display On and Cursor Off
lcdcmd(0x01);// Clear display screen
lcdcmd(0x06);// Increment cursor
lcdcmd(0x80);// Set cursor position to 1st line,1st column

Hardware/Software setup:



Sending command to the LC:

- rs=0; Register select pin is low.
- rw=0; Read/write Pin is also for writing the command to the LCD.

- en=1;enable pin is high.

Sending data to the LCD:

- rs=1; Register select pin is high.
- rw=0; Read/write Pin is also for writing the command to the LCD.
- en=1; enable pin is high.

ALGORITHM:

Step1: Reset TRISD

Step2: Reset TRISE

Step3: store the messages in two character arrays

Step4: store 0x0F in ADCON1 register

Step5: initialize LCD

Step6: call the delay

Step7: write the first string

Step8: call the delay

Step9: move the cursor to the second line in LCD

Step10: write the second string

Program:

```
#include <PIC18f4550.h> //Include Controller specific .h
//#pragma config FOSC = HS //Oscillator Selection
#pragma config WDT = OFF //Disable Watchdog timer
#pragma config LVP = OFF //Disable Low Voltage Programming
//#pragma config PBADEN = OFF //Disable PORTB Analog inputs
//Declarations
#define LCD_DATA PORTD //LCD data port to PORTD
#define ctrl PORTE //LCD control port to PORTE
#define rs PORTEbits.RE0 //register select signal to RE0
#define rw PORTEbits.RE1 //read/write signal to RE1
#define en PORTEbits.RE2 //enable signal to RE2
//Function Prototypes
void init_LCD(void); //Function to initialize the LCD
void LCD_command(unsigned char cmd);//Function to pass command to LCD
void LCD_data(unsigned char data); //Function to write char to LCD
void LCD_write_string(char *str);//Function to write string
void msdelay (unsigned int time); //Function to generate delay
//Start of Main Program
void main(void)
{
char var1[] = "WEL-COME";//Declare message to be displayed
char var2[] = "SE-IT  ";
ADCON1 = 0x0F; //Configuring the PORTE pins as digital I/O
TRISD = 0x00; //Configuring PORTD as output
TRISE = 0x00; //Configuring PORTE as output
init_LCD(); // call function to initialize of LCD
msdelay(50); // delay of 50 milliseconds
LCD_write_string(var1); //Display message on first line
msdelay(150);
LCD_command(0xC0); // initiate cursor to second line
LCD_write_string(var2);//Display message on second line
while (1); //Loop here
} //End of Main

void msdelay (unsigned int time) //Function to generate delay
{
unsigned int i, j;
for (i = 0; i < time; i++)
for (j = 0; j < 710; j++);//Calibrated for a 1 ms delay in MPLAB
}
void init_LCD(void) // Function to initialize the LCD
{
LCD_command(0x38); // initialization of 16X2 LCD in 8bit mode
msdelay(15);
LCD_command(0x01); // clear LCD
```

```
msdelay(15);
LCD_command(0x0C);  // cursor off
msdelay(15);
LCD_command(0x80);  // go to first line and 0th position
msdelay(15);
}
void LCD_command(unsigned  char cmd) //Function  to pass command to LCD
{
LCD_DATA = cmd; //Send data on LCD data bus
rs = 0; //RS = 0 since command to LCD
rw = 0; //RW = 0 since writing to LCD
en = 1; //Generate High to low pulse on EN
msdelay(15);
en = 0;
}
void LCD_data(unsigned char data)//Function  to write data to the LCD
{
LCD_DATA = data; //Send data on LCD data bus
rs = 1; //RS = 1 since data to LCD
rw = 0; //RW = 0 since writing to LCD
en = 1; //Generate High to low pulse on EN
msdelay(15);
en = 0;
}
//Function to write string to LCD
void LCD_write_string( char *str)
{
int i = 0;
while (str[i] != '\0') //Check for end of the string
{
LCD_data(str[i]); // sending data on LCD byte by byte
msdelay(15);
i++;
}
}
```

Input: Make all the connections and start

Output: Two input strings are displayed on the LCD.

Conclusion: The student is able to understand the working of the PIC18 with LCD and able to write the codes for the initialization of LCD, writing to LCD etc.

Programming Skill Development Lab			
Experiment No: 10	**GROUP C: ASSIGNMENTS** Write an Embedded C program for Generating PWM signal for servo motor/DC motor.	Page	1/4

Aim: Control speed of the DC motor and rotation using PIC18F family.

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXXmicrocontroller kit

Theory:
There are different kinds of controls of dc motor.

1. Direction control of dc motor both in clock wise and in anti clock wise direction
2. Speed control of DC motor by changing the input voltage
3. Speed control of DC motor by PWM technique
4. Speed control of DC motor by PID control
5. Speed control of DC motor using microcontroller

In this project, we will discuss only two methods of control of DC motor

1. Direction control of DC motor using PIC microcontroller
2. speed control of dc motor using PIC microcontroller

Direction control of dc motor using PIC microcontroller:

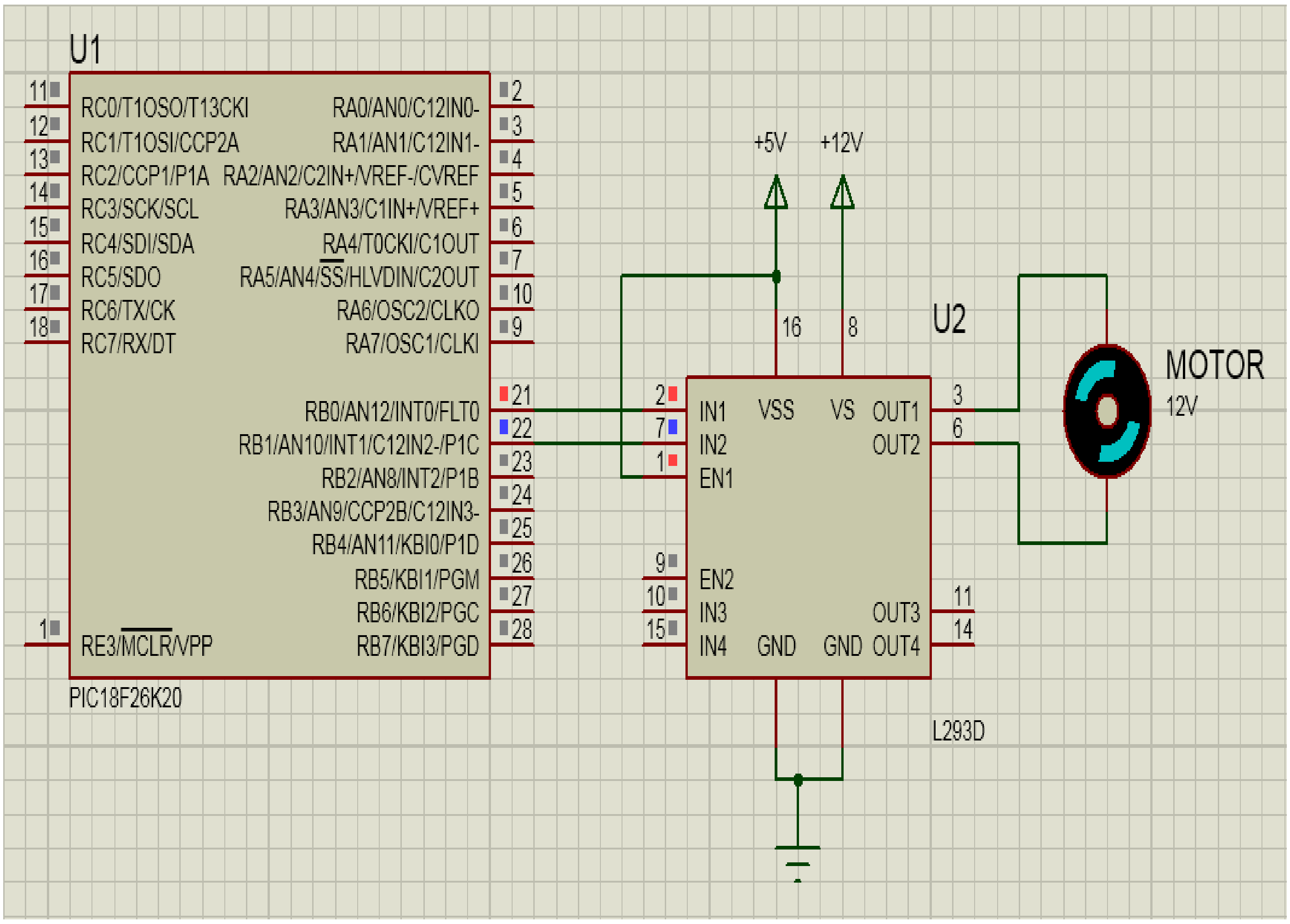
The following components are used in direction control of dc motor

1. PIC microcontroller PIC18F458
2. L293D
3. DIODES 1N4004
4. DC MOTOR
5. BATTERT +12V
6. PUSH SWITCH
7. 10K RESISTOR
8. 8MHZ CRYSTAL
9. CAPACITOR 22PF

MOTOR DRIVER IC L293D:

We can't drive a DC Motor (depends) directly with a Microcontroller, as DC Motors requires high current and high voltage than a Microcontroller can handle. Microcontrollers usually operates at +5 or +3.3V supply and it I/O pin can provide only up to 25mA current. Commonly used DC Motors requires 12V supply and 300mA current, moreover interfacing DC Motors directly with Microcontrollers may affect the working of Microcontroller due to the Back EMF of the DC Motor. Thus it is clear that, it not a good idea to interface DC Motor directly with microcontrollers The solution to the above problem is used H-BRIDGE. It is a special circuit, by using the 4 switches we can control the direction of DC Motor. Depending upon our power requirements we can make our own H-bridge using Transistors/MOSFETs as switches. It is better to use ready made ICs, instead of making our own H-bridge.

INTERFACING MOTOR WITH PIC MICROCONTROLLER:



PREPARED BY APPROVED BY CONTROLLED COPY STAMP MASTER COPY STAMP

Muneshwar R.N.

L293D and L293 are two such ICs. These are dual H-bridge motor drivers, by using one IC we can control two DC Motors in both clock wise and counter clockwise directions. The L293D can provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V while L293 can provide up to 1A at same voltages. Both ICs are designed to drive inductive loads such as dc motors, bipolar stepping motors, relays and solenoids as well as other high-current or high-voltage loads in positive-supply applications. All inputs of these ICs are TTL compatible and output clamp diodes for inductive transient suppression are also provided internally. These diodes protect our circuit from the Back EMF of DC Motor. In both ICs drivers are enabled in pairs, with drivers 1 and 2 are enabled by a high input to 1,2EN and drivers 3 and 4 are enabled by a high input to 3,4EN. When drivers are enabled, their outputs will be active and in phase with their inputs. When drivers are disabled, their outputs will be off and will be in the high-impedance state.

```
#include "mcc_generated_files/mcc.h"
//This function creates seconds delay. The argument specifies the delay time in seconds
void Delay_Seconds(unsigned char z)
{
   unsigned char x,y;
   for(y=0;y<z;y++)
   {
      for(x=0;x<100;x++)__delay_ms(10);
   }
}
void main(void)
{
while (1)
         {
      //Turn motor clockwise
  IN1_SetHigh();
  IN2_SetLow();
  Delay_Seconds(5);//5 seconds delay

  //Stop motor
  IN1_SetLow();
  IN2_SetLow();
  Delay_Seconds(2);//2 seconds delay

  //Turn motor anticlockwise direction
  IN1_SetLow();
  IN2_SetHigh();
  Delay_Seconds(5);//5 seconds delay

  //Stop motor
  IN1_SetHigh();
  IN2_SetHigh();
  Delay_Seconds(2);//2 seconds delay
        }
}
```

Input: DC Motor is connected to micro-controller.

Output: Motor spins in clock wise and anti – clock wise direction

Conclusion: In this project we learnt how to use dc motor in line follower robot and how to interface dc motor with microcontroller. The main thing, We learnt from this project how to develop logic to program microcontroller. We wrote both the programs and these two methods of dc motor control will be help full for me in control projects.

Programming Skill Development Lab **GROUP C: ASSIGNMENTS**			
Experiment No: 11	Write an Embedded C program for PC to PC serial communication using UART.	**Page**	**1/7**

Aim: Serial communication between PIC18F4550 and PC

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXXmicrocontroller kit our own H-bridge.

Introduction

Several devices such as GPS, GSM, RFID, sensors, etc need to communicate with the PIC microcontroller for transmitting or receiving information. To communicate with the PIC microcontroller, several communication protocols are used such as RS232, SPI, I2C, CAN, etc. Basically, a protocol is a set of rules agreed by both, the sender and the receiver, on -

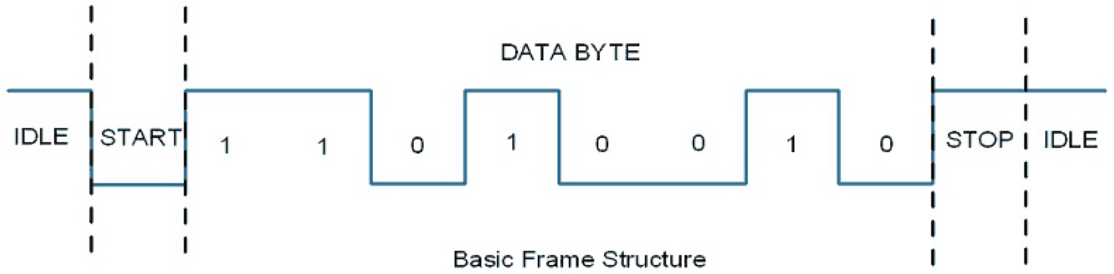
- How the data is packed?
- How many bits constitute a character?
- When the data begins and ends?

PIC18F4550 has an in-built USART module which is useful for serial communication. With the help of USART, we can send/receive data to a computer or other devices. USART is also used in interfacing PIC with various modules like Wi-Fi (ESP8266), Bluetooth, GPS, GSM, etc.

We will see how the communication is established between PIC microcontroller and PC through USART using RS232 protocol. We will also see how to communicate with laptops, which do not have an RS232 DB9 port, and instead use a USB port.

Let us start with the serial communication using PIC18F4550.

Asynchronous Communication: PIC18F4550 has a built-in asynchronous receiver-transmitter. Asynchronous means each character (data byte) is placed in between the start and stop bits. The start bit is always 0 (low) and the stop bit is always 1 (high).



Basic Frame Structure

Bit Rate & Baud Rate: The rate of data transfer in serial data communication is stated in bps (bits per second). Another widely used terminology for bps is baud rate; means, a number of changes in signal per second. Here the signal is in bits, therefore bit rate = baud rate.

Interface: Although there are many pins in the DB9 connector, we do not need all. We only use pins RX, TX, and GND.

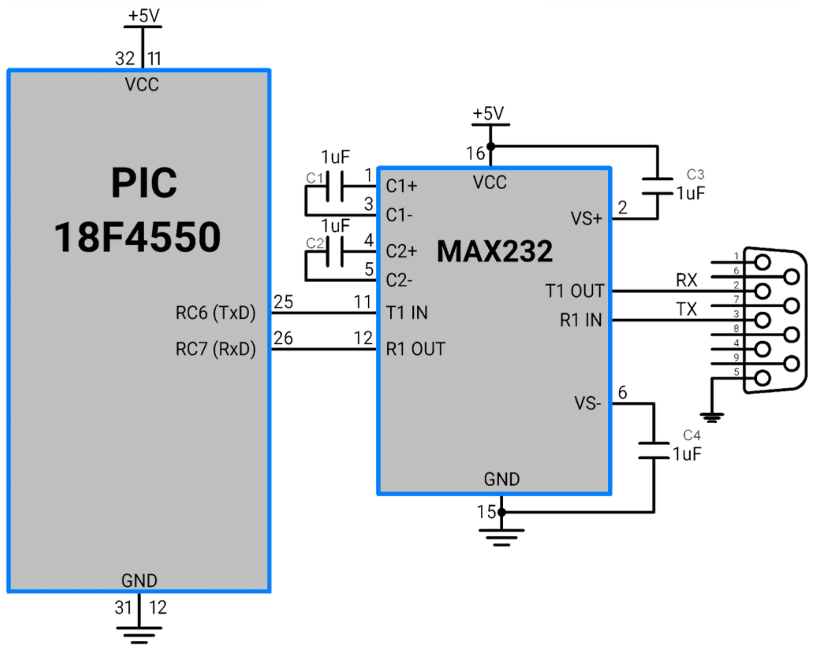
Level Conversion

Convert PIC18F TTL levels to RS232 levels and vice-versa

PC has RS232 levels whereas the PIC microcontroller has TTL levels. The RS232 has different voltage levels for logic 0 and 1. To make it compatible with the PIC TTL voltage levels, we have to use a MAX232 IC.

TTL	
LOGIC 0	0 V
LOGIC 1	5 V

RS232	
LOGIC 0	+3 to +25 V
LOGIC 1	-3 to -25 V



Baud Rate Calculation

How to Calculate the Baud Rate in PIC18F4550?

$$Desired\ Baud\ Rate = \frac{Fosc}{64 * (X + 1)}$$

a value that will be loaded into the SPBRG register (16-bit) of the PIC18F4550 to get the desired baud rate. The value of SPBRG for the desired baud rate is calculated as,

$$SPBRG = \frac{Fosc}{64 * Desired\ Baud\ Rate} - 1$$

E.g.

Suppose, Fosc = 8 MHz and Baud Rate = 9600 bps

Then, **SPBRG=((8 MHz)/(64 ×9600))-1**

SPBRG = (8 MHz / 64 × 9600) - 1

Therefore, SPBRG = 12

The above formula depends on the **BRGH** bit in the **TXSTA** register.

Let’s see the TXSTA register, which is used for transmission setting, in detail as follows

TXSTA: Transmit Status and Control Register

7	6	5	4	3	2	1	0
CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D

TXSTA Register

TXEN: Transmit Enable Bit

1 = Enable the transmission

0 = Disable the transmission

BRGH: High Baud Rate select bit

0 = Low speed

$$SPBRG = \frac{Fosc}{64 * Desired\ Baud\ Rate} - 1$$

1 = High speed

$$SPBRG = \frac{Fosc}{16 * Desired\ Baud\ Rate} - 1$$

Load the calculated value directly to the SPBRG register.

CSRC bit is not used for asynchronous communication.

TX9 : 9th Transmit Enable Bit

0 = Select 8-bit transmission

1 = Select 9-bit transmission

SYNC : USART Mode Select Bit

0 = Asynchronous mode

1 = Synchronous mode

SENDB : Send Break Character bit

1 = Send Sync Break on next transmission (cleared by hardware upon completion)

0 = Sync Break transmission completed

TRMT : Transmit Shift Register Status Bit

0 = TSR full

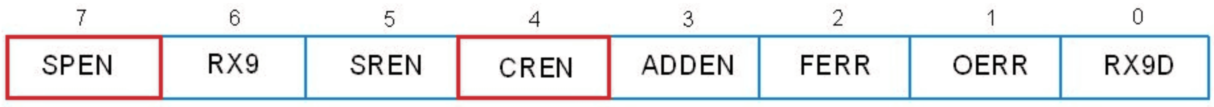
1 = TSR empty

TX9D: 9th bit of transmitting data

Can be Address / Data bit or a parity bit.

In PIC18F4550, the RCSTA register is used for serial data receive settings.

RCSTA: Receive Control and Status Register



7	6	5	4	3	2	1	0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

RCSTA Register

SPEN: Serial Port Enable

1= Enable Serial Port for communication

0 = Disable Serial Port for communication

RX9 : 9-bit Receive Enable bit

1 = Enable 9-bit reception

0 = Enable 8-bit reception

Generally, we use 8-bit reception

SREN: Single Receive Enable bit

Not used

CREN: Continuous Receive Enable bit

1 = Enable receiver for continuous reception of data byte

0 = Disable receiver

ADDEN: Address Detect Enable bit

Asynchronous mode 9-bit (RX9 = 1)

1 = Enable address detection, enable interrupt, and load the receive buffer when the RSR bit is set.

0 = Disable address detection, all bytes are received and the ninth bit can be used as a parity bit.

Asynchronous mode 9-bit (RX9 = 0)

Don't care (any 0 or 1)

FERR: Framing Error bit

1 = Framing error (can be updated by reading the RCREG register and receiving the next valid byte)

0 = No framing error

OERR: Overrun Error bit

1 = Overrun error can be cleared by clearing bit CREN.

0 = No overrun error

RX9D: 9th bit of the Receiving Data

This can be address/data bit or a parity bit and must be calculated by user firmware.

Data Buffer and Interrupt Flag for Serial Communication

For transmitting data and reception of data **TXREG** and **RCREG** 8-bit data registers are allocated in PIC18F4550 respectively.

Transmit Register

Receive Register

- When we have to transmit data, we directly copy that data to the TXREG register. After completing the transmission of 8-bit data, the **TXIF** interrupt flag is generated.

- This **TXIF** (transmit interrupt flag) is located in the **PIR1** register. TXIF flag is set when the 8-bit data is transmitted. Then, the buffer is ready to receive another data for transmission.
- Also, **RCIF** (receive interrupt flag) is located in the **PIR1** register. When this flag is set, it indicates that the complete data byte is received by the **RCREG** register. Read the **RCREG** register immediately. Now, the RCREG register is ready to receive another data.
- When the RCIF flag is not set, the PIC microcontroller has to wait for the reception of the complete data byte.

Steps for Programming PIC18F4550 USART

Initialization

1. Initialize the Baud Rate by loading a value into the SPBRG register.

2. Then set bit SPEN in the RCSTA for enabling Serial Port.

3. Then set bit BRGH in the TXSTA for low or high speed.

4. Also clear bit SYNC in the TXSTA register for asynchronous communication.

5. Set bit TXEN in the TXSTA register to enable transmission.

6. Set bit CREN in the RCSTA register to enable reception.

Input: UART circuit

Output: Data transmission between PC to Microcontroller.

Conclusion: Here we have learnt UART communication.

Programming Skill Development Lab GROUP C: ASSIGNMENTS			
Experiment No: 12	Write an Embedded C program for Temperature sensor interfacing using ADC & display on LCD.	**Page**	**1/5**

Aim: LM35 temperature interfacing with PIC18F4550.

Apparatus: MPLAB X IDE simulator, XC8 Compiler, PIC18FXXXmicrocontroller kit our own H-bridge.

Introduction

LM35 is a temperature sensor that can measure temperature in the range of -55°C to 150°C.

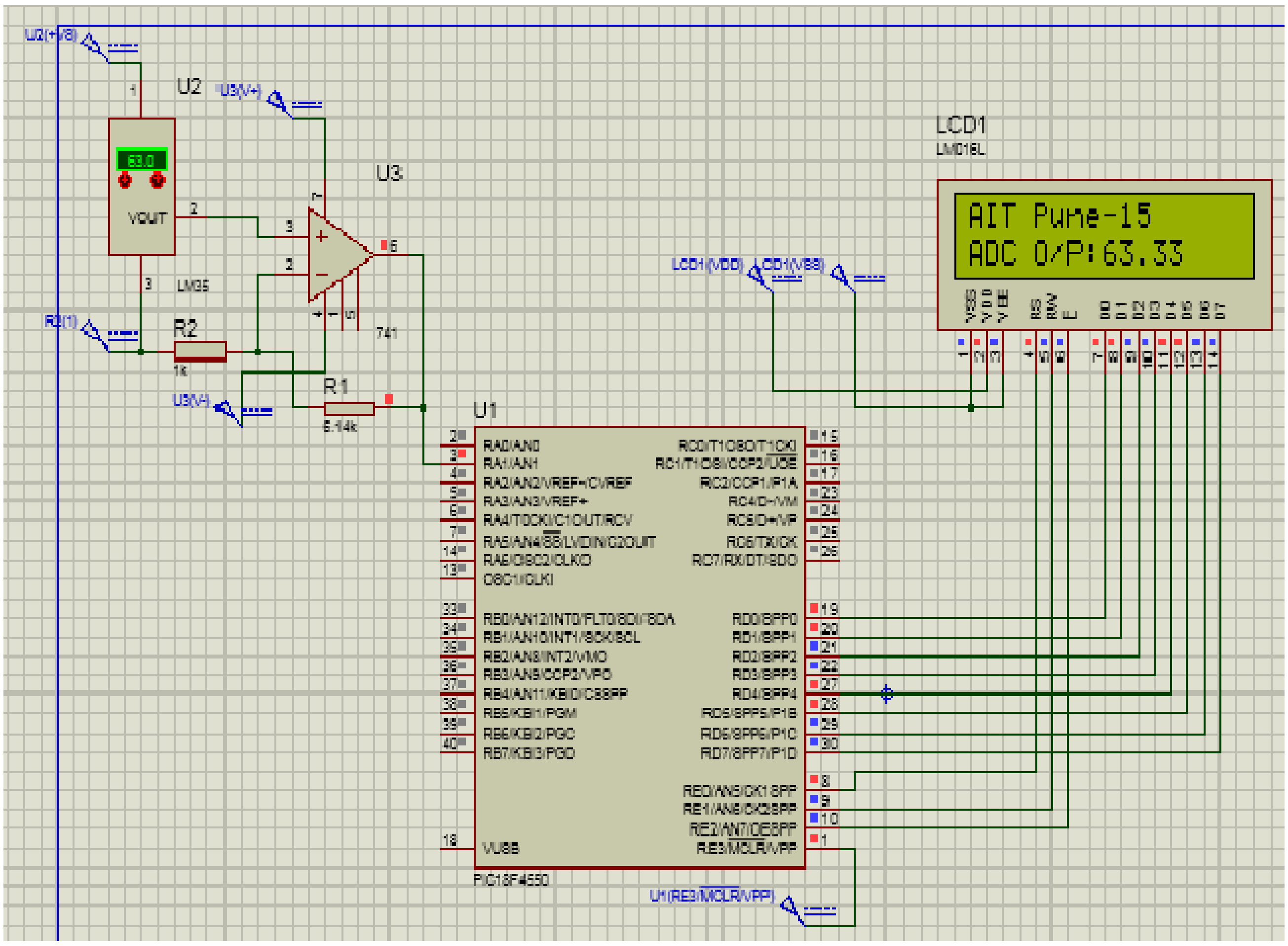
It is a 3-terminal device that provides an analog voltage proportional to the temperature. The higher the temperature, the higher is the output voltage.

The output analog voltage can be converted to digital form using ADC so that a microcontroller can process it.

For more information about LM35 and how to use it, refer to the topic LM35 Temperature Sensor in the sensors and modules section.

For information about ADC in PIC18F4550 and how to use it, refer to the topic ADC in PIC18F4550 in the PIC inside section.

Interfacing Diagram



LM35 Temperature Sensor Interfacing with PIC18F4550

Program:

//Declarations for LCD Connection

```
 #define LCD_DATA   PORTD      //LCD data port
 #define en         PORTEbits.RE2 // enable signal
 #define rw         PORTEbits.RE1 // read/write signal
 #define rs         PORTEbits.RE0    // register select signal
  //Function Prototypes
 void ADC_Init(void);    //Function to initialize the ADC
 unsigned int  Get_ADC_Result(void);   //Function to Get ADC result
 void Start_Conversion(void); //Function to Start of Conversion
 void msdelay (unsigned int time); //Function to generate delay
 void init_LCD(void);    //Function to initialize the LCD
 void LCD_command(unsigned char cmd); //Function to pass command to LCD
 void LCD_data(unsigned char data);//Function to write character to LCD
 void LCD_write_string(static char *str); //Function to write string
//Start of main program
 void main()
  {
  char msg1[] = "AIT Pune-15";
  char msg2[] = "ADC O/P:";
  unsigned char i, Thousands,Hundreds,Tens,Ones,dot;
  unsigned int adc_val;
dot= 46;
    ADCON1 = 0x0F;       //Configuring the PORT pins as digital I/O
    TRISD = 0x00;       //Configuring  PORTD as output
    TRISE = 0x00;       //Configuring  PORTE as output

  ADC_Init();   // Init ADC peripheral
  init_LCD();   // Init LCD Module
  LCD_write_string(msg1); // Display Welcome Message
  LCD_command(0xC0);  // Goto second line, 0th place of LCD
  LCD_write_string(msg2); // Display Message "ADC O/P"

   while(1)
  {
     Start_Conversion(); //Trigger conversion
     adc_val= Get_ADC_Result(); //Get the ADC output by polling GO bit
     adc_val=adc_val*6.84;
     LCD_command (0xC8);  //Goto 9th place on second line of LCD

     i = adc_val/1000 ;  //Get the thousands place
     Thousands = i + 0x30;  // Convert it to ASCII
     LCD_data (Thousands);  // Display thousands place

     i = (adc_val%1000)/100; //Get the Hundreds place
     Hundreds = i + 0x30;  // Convert it to ASCII
     LCD_data (Hundreds);  //Display Hundreds place
     LCD_data(dot);
     i = ((adc_val%1000)%100)/10; //Get the Tens place
     Tens = i + 0x30;   // Convert it to ASCII
     LCD_data (Tens);   //Display Tens place

     i = adc_val%10 ;   //Get the Ones place
     Ones = i + 30;    // Convert it to ASCII
     LCD_data (i + 0x30);  //Display Ones place
     msdelay(300);   //Delay between conversions.
      }
       }

  //Function Definitions
   void ADC_Init()
              {
      ADCON0=0b00000100; //A/D Module is OFF and Channel 1 is selected
              ADCON1=0b00001110; // Reference as VDD & VSS, AN0 set as analog pins
              ADCON2=0b10001110; // Result is right Justified
                          //Acquisition Time 2TAD
                     //ADC Clk FOSC/64
              ADCON0bits.ADON=1;  //Turn ON ADC module
          }
void Start_Conversion()
 {
   ADCON0bits.GO=1;
        }
//Poll till the conversion complete
 unsigned int Get_ADC_Result()
  {
   unsigned int ADC_Result=0;
   while(ADCON0bits.GO);
    ADC_Result=ADRESL;
    ADC_Result|=((unsigned int)ADRESH) << 8;
    return ADC_Result;
      }
void msdelay (unsigned int time) //Function to generate delay
{
  unsigned int i, j;
  for (i = 0; i < time; i++)
  for (j = 0; j < 710; j++);//Calibrated for a 1 ms delay in MPLAB
    }
void init_LCD(void)  // Function to initialise the LCD
 {
  LCD_command(0x38);     // initialization of 16X2 LCD in 8bit mode
  msdelay(15);
  LCD_command(0x01);     // clear LCD
  msdelay(15);
  LCD_command(0x0C);      // cursor off
  msdelay(15);
  LCD_command(0x80);     // go to first line and 0th position
  msdelay(15);
     }
void LCD_command(unsigned char cmd) //Function to pass command to the LCD
 {
   LCD_DATA = cmd;   //Send data on LCD data bus
   rs = 0;   //RS = 0 since command to LCD
   rw = 0;   //RW = 0 since writing to LCD
   en = 1;   //Generate High to low pulse on EN
   msdelay(15);
   en = 0;
     }
void LCD_data(unsigned char data)//Function to write data to the LCD
 {
   LCD_DATA = data; //Send data on LCD data bus
   rs = 1;   //RS = 1 since data to LCD
   rw = 0;   //RW = 0 since writing to LCD
   en = 1;   //Generate High to low pulse on EN
   msdelay(15);
   en = 0;
    }
 //Function to write string to LCD
void LCD_write_string(static char *str)
 {
   int i = 0;
   while (str[i] != 0)
    {
      LCD_data(str[i]);     // sending data on LCD byte by byte
      msdelay(15);
      i++;
       }
        }
```

Input: LM35 Temperature sensor.

Output: Temperature display on LCD panel.

Conclusion: Thus we studied Temperature sensor interfacing using ADC & display on LCD.

Programming Skill Development Lab GROUP D: ASSIGNMENTS			
Experiment No: 13	Study of Arduino board and understand the OS installation process on Raspberry-pi.	**Page**	**1/3**

Aim: To study operating systems for platforms such as Raspberry-Pi/Beagle board/Arduino.

1) **Raspberry-Pi**: - The Pi can run the official Raspbian OS, Ubuntu Mate, Snappy Ubuntu Core, the Kodi-based media centers OSMC and LibreElec, the non-Linux based Risc OS (one for fans of 1990s Acorn computers). It can also run Windows 10 IoT Core, which is very different to the desktop version of Windows, as mentioned below.

- **OS which install on Raspberry-Pi**: Raspbian, Ubuntu MATE, Snappy Ubuntu, Pidora, Linutop, SARPi, Arch Linux ARM, Gentoo Linux, etc.
- **How to install Raspbian on Raspberry-Pi:**

Step 1: Download Raspbian

Step 2: Unzip the file. The Raspbian disc image is compressed, so you'll need to unzip it. The file uses the ZIP64 format, so depending on how current your built-in utilities are, you need to use certain programs to unzip it.

Step 3: Write the disc image to your microSD card. Next, pop your microSD card into your computer and write the disc image to it. The process of actually writing the image will be slightly different across these programs, but it's pretty self-explanatory no matter what you're using. Each of these programs will have you select the destination (make sure you've picked your microSD card!) and the disc image (the unzipped Raspbian file). Choose, double-check, and then hit the button to write.

Step 4: Put the microSD card in your Pi and boot up. Once the disc image has been written to the microSD card, you're ready to go! Put that sucker into your Raspberry Pi, plug in the peripherals and power source, and enjoy. The current edition to Raspbian will boot directly to the desktop. Your default credentials are username **pi** and password **raspberry.**

2) BeagleBone Black:- The **BeagleBone Black** includes a 2GB or 4GB on-board eMMC flash memory chip. It comes with the Debian distribution factory pre-installed. You can flash new operating systems including Angstrom, Ubuntu, Android, and others.

1. **Os which install on BeagleBone Black:** Angstrom, Android, Debian, Fedora, Buildroot, Gentoo, Nerves Erlang/OTP, Sabayon, Ubuntu, Yocto, MINIX 3

 How to install Debian on BeagleBone Black:

Step 1: Download Debian img.xz file.

Step 2: Unzip the file.

Step 3: Insert your MicroSD (uSD) card into the proper slot. Most uSD cards come with a full-sized SD card that is really just an adapter. If this is what you have then insert the uSD into the adapter, then into your card reader.

Step 4: Now open Win32 Disk imager, click the blue folder icon, navigate to the debian img location, and double click the file. Now click Write and let the process complete. Depending on your processor and available RAM it should be done in around 5 minutes.

Step 5: Alright, once that's done, you'll get a notification pop-up. Now we're ready to get going. Remove the SD adapter from the card slot, remove the uSD card from the adapter. With the USB cable disconnected insert the uSD into the BBB.

Step 6: Now, this next part is pretty straight forward. Plug the USB cable in and wait some more. If everything is going right you will notice that the four (4) leds just above the USB cable are doing the KIT impression. This could take up to 45 minutes, I just did it again in around 5 minutes. Your mileage will vary. Go back and surf reddit some more.

Step 7: If you are not seeing the leds swing back and forth you will need to unplug the USB cable, press and hold down the user button above the uSD card slot (next to the 2 little 10 pin ICs) then plug in the USB cable. Release the button and wait. You should see the LEDs swinging back and forth after a few seconds. Once this happens it's waiting time. When all 4 LEDs next to the USB slot stay lit at the same time the flash process has been completed.

Step 8: Remove the uSD card and reboot your BBB. You can reboot the BBB by removing and reconnecting the USB cable, or hitting the reset button above the USB cable near the edge of the board.

Step 9: Now using putty, or your SSH flavor of choice, connect to the BBB using the IP address

192.168.7.2. You'll be prompted for a username. Type root and press Enter. By default, there is no root password. I recommend changing this ASAP if you plan on putting your BBB on the network. To do this type password, hit enter, then enter your desired password. You will be prompted to enter it again to verify.

3) Arduino: - The Arduino itself has no real operating system. You develop code for the Arduino using the Arduino IDE which you can download from Arduino - Home. Versions are available for Windows, Mac and Linux. The Arduino is a constrained microcontroller.
Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. You are literally writing the "firmware" when you write the code and upload it. It's both good and its bad.

Input: kits of Raspberry – pi, Arduino and Beagle board.

Output: OS is installed on Raspberry – Pi.

Conclusion: Thus, we have studied of how to install operating systems for platforms such as Raspberry-Pi/Beagle board/Arduino.

Programming Skill Development Lab			
Experiment No: 14	Write simple program using Open source prototype platform like Raspberry-Pi/Beagle board/Arduino for digital read/write using LED and switch Analog read/write using sensor and actuators.	**Page**	**1/4**

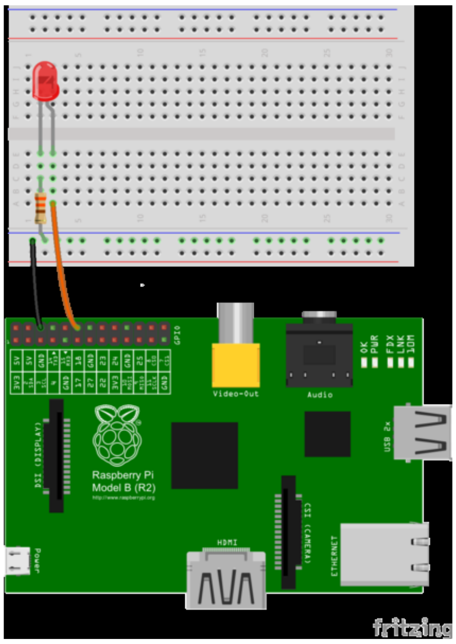
Aim: To get knowledge for communicating with objects using sensors and actuators.

An application of blinking LEDs using Raspberry Pi

In this assignment, we will learn how to interface an LED with Raspberry Pi. To implement this, we will gather the requirements and follow the instructions given below.

Requirements:

1. Raspberry Pi3 Kit
2. Breadboards
3. LEDs
4. Jumper wires



Instructions to Glow LED:

Use GPIO diagram of Raspberry Pi, which will give you understanding of GPIO structure.

1. Connect GPIO 18 (Pin No. 12) of Raspberry Pi to the Anode of the LED through connecting jumper wires and Breadboard.

2. Connect Ground of Raspberry Pi to Cathode of the LED through connecting wires and breadboard.

3. Now power up your Raspberry Pi and boot.

4. Open terminal and type *nano blinkled.py*

5. It will open the nano editor. Use following pseudo code in the python to blink LED and save

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
while True:
GPIO.output(18, GPIO.HIGH)
time.sleep(1)
GPIO.output(18, GPIO.LOW)
time.sleep(1)
```

6. Now run the code. Type *python blinkled.py*

7. Now LED will be blinking at an interval of 1s. You can also change the interval by modifying time.sleep in the file.

8. Press Ctrl+C to stop LED from blinking.

An application to read the environment temperature and humidity & display it.

In this assignment, we will learn how to find environment temperature and humidity using DHT11 sensor and Raspberry Pi.

Temperature and Humidity Sensor- DHT11

- It's a very basic and slow sensor, easy to use for small scale.
- The DHT sensors are made of two parts, a capacitive humidity sensor and thermistor.
- There is also a very basic chip inside that does some analog to digital conversion and spits out a digital signal with the temperature and humidity.

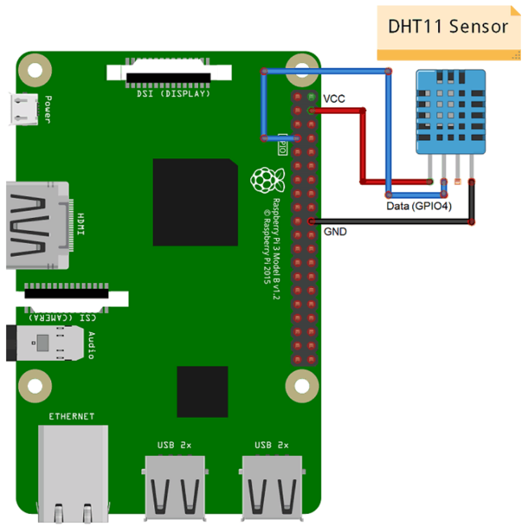
- The digital signal is fairly easy to read using any microcontroller.
- These sensors are frequently used in remote weather stations, soil monitors, and home environment control systems.

Features of DHT11 Sensor: -

- Ultra-low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20–80% humidity readings with 5% accuracy
- Good for 0–50°C temperature readings ±2°C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

Requirements:

1. Raspberry Pi3 Kit
2. Breadboards
3. DHT11 Sensor
4. Jumper wires



Algorithm:

1. Connect GPIO 18 (Pin No. 12) of Raspberry Pi to the Data pin of DHT11 sensor through connecting jumper wires and Breadboard.

2. Connect Power (5V) and Ground of Raspberry Pi to DHT11 sensor through connecting wires and breadboard.

3. Now power up your Raspberry Pi and boot.

4. Download Adafruit_Python_DHT.tar.gz package

5. Open terminal and type *tar -xvzf Adafruit_Python_DHT.tar.gz* command to extract Adafruit package.

6. In the same terminal, type *nano temp.py*

7. It will open the nano editor. Use following pseudo code in the python to blink LED and save

```
import Adafruit_DHT as dht
import time
while True:
hum, temp = dht.read_retry(11,18)
print "Temp: ", temp
print "Hum: ", hum
time.sleep(1)
```

8. Now run the code. Type *python temp.py*

9. Now temperature and humidity will be displayed at an interval of 1s. You can also change the interval by modifying time.sleep in the file.

10. Press Ctrl+C to stop displaying temperature and humidity.

Input: kits of Raspberry – pi, Arduino and Beagle board, LED and DHT11.

Output: LED will glow.

Conclusion: Thus, we have studied how to monitor LEDs and measure temperature and humidity in the environment using DHT11 sensor and Raspberry Pi 3.