214455: Programming Skill Development Laboratory

SEIT (2019 Course)

Semester - II

Teaching Scheme		Examination Scheme	
Practical :	2 Hrs. / Week	Term work :	25 Marks
		Practical	25 Marks

Sinhgad Institutes

LABORATORY MANUAL V 3.0

DEPARTMENT OF INFORMATION TECHNOLOGY

Sinhgad College of Engineering, Pune

2024-2025

VISION

To provide excellent Information Technology education by building strong teaching and research environment.

MISSION

1) To transform the students into innovative, competent and high quality IT professionals to meet thegrowing global challenges.
2) To achieve and impart quality education with an emphasis on practical skills and social relevance.
3) To endeavor for continuous up-gradation of technical expertise of students to cater to the needs ofthe society.
4) To achieve an effective interaction with industry for mutual benefits.

PROGRAM EDUCATIONAL OBJECTIVES

The students of Information Technology course after passing out will

1. Possess strong fundamental concepts in mathematics, science, engineering and Technologyto address technological challenges.

2. Possess knowledge and skills in the field of Computer Science and Information Technology for analyzing, designing and implementing complex engineering problems of any domain with innovative approaches.

3. Possess an attitude and aptitude for research, entrepreneurship and higher studies in the field of Computer Science and Information Technology.

4. Have commitment to ethical practices, societal contributions through communities and life-long learning.

5. Possess better communication, presentation, time management and teamwork skills leading to responsible & competent professionals and will be able to address challenges inthe field of IT at global level.

PROGRAM OUTCOMES

The students in the Information Technology course will attain:

PO1	Engineering knowledge	An ability to apply knowledge of mathematics, computing, science, engineering and technology.
PO2	Problem analysis	An ability to define a problem and provide a systematic solution with the help of conducting experiments, analyzing the problem and interpreting the data.
PO3	Design / Development of Solutions	An ability to design, implement, and evaluate a software or a software/hardware system, component, or process to meet desired needs within realistic constraints.
PO4	Conduct Investigations of Complex Problems	An ability to identify, formulates, and provides systematic solutions to complex engineering/Technologyproblems.
PO5	Modern Tool Usage	An ability to use the techniques, skills, and modern engineering technology tools, standard processes necessary for practice as a IT professional.
PO6	The Engineer and Society	An ability to apply mathematical foundations, algorithmic principles, and computer science theory inthe modeling and design of computer-based systems with necessary constraints and assumptions.
PO7	Environment and Sustainability	An ability to analyze and provide solution for the local and global impact of information technology on individuals, organizations and society.
PO8	Ethics	An ability to understand professional, ethical, legal, security and social issues and responsibilities.
PO9	Individual and Team Work	An ability to function effectively as an individual or as a team member to accomplish a desired goal(s).
PO10	Communication Skills	An ability to engage in life-long learning and continuing professional development to cope up with fast changes in the technologies/tools with the help of electives, professional organizations and extra-curricular activities.
PO11	Project Management and Finance	An ability to communicate effectively in engineering community at large by means of effective presentations, report writing, paper publications, demonstrations.
PO12	Life-long Learning	An ability to understand engineering, management, financial aspects, performance, optimizations and time complexity necessary for professional practice.

Compliance Document Control

Reference Code	SCOE-IT / Lab Manual Procedures
Version No.	3. 0
Compliance Status	Complete
Revision Date	2nd January 2025
Security Classification	Department Specific
Document Status	Definitive
Review Period	Yearly

	Author	Authorizer
Signature		
Name	Mrs.M.S.Bhosale	Dr.S.R.Ganorkar
Designation	Assistant Prof.	Head of Department

Document History

Revision No.	Revision Date	Reason For Change
1.0	17th January 2021	University syllabus modification - course 2019
2.0	27th January 2023	According to Oral point of view.
3.0	2nd January 2024	Experiment No.10 added in Manual.

Summary of Changes to PSDL Laboratory Manual

Sr. No	Changes	Change type
1	Theory content added in PC to PC serial communication experiment	Text.
2	Experiment No.10 added in Manual.	Whole Experiment.

PREFACE

Introduction of microcontroller was a turning point in the era of electronic product design. Ability of programs to have decision making capabilities which is not possible with conventional discreet logic was a major breakthrough in designing intelligent products. Today microprocessors & microcontrollers dominate industrial scene & consumer applications. PIC microcontroller, ARM & other programmable peripherals from Intel and microchip are the ideal vehicles to make the students understand microprocessor architecture & integrated systems. In order to achieve these objectives of understanding microcontroller architecture & embedded system design architecture embedded C language programming: “Processor Architecture” theory course is designed. The objective of this laboratory is to understand all these concepts by practically trying out programming and experiments. It also aims at developing analytical skills and problem solving abilities of the students. It is not necessary to mention that, performing things practically further reinforces the concepts learned in the classroom. This laboratory is organized into 10 assignments, divided into 4 groups. After completion of all designed experiments, students will be familiar with microcontroller based designing and will be prepared to understand micro-controllers for embedded applications. They will also be familiar with embedded C programming.

In this laboratory theory comes alive and practical hands-on skills are learnt; a balance is struck between theory and practice. This laboratory manual is prepared by referring to various standard books which help the students to perform the experiments. Students are not expected to copy the contents of the manual as it is. They must understand the concepts given in the manual and write journal on their own. The manual is prepared as per Savitribai Phule Pune University (SPPU) syllabus and accordingly the practical assignments are discussed in the manual. However, students can go beyond this set and perform extra practical assignments.

Savitribai Phule Pune University, Pune
Second Year Information Technology (2019 Course)
214455: Programming Skill Development Lab

Teaching Scheme:	**Credit Scheme:**	**Examination Scheme:**
Theory(TH) :02hrs/week	**01**	**PR: 25Marks**
		TW: 25Marks

Prerequisites: Computer Organization and Architecture

Course Objectives:

1. To learn embedded C programming and PIC18FXXXmicrocontrollers.
2. To learn interfacing of real-world input and output devices to PIC18FXXX microcontroller

Course Outcomes:

On completion of this course student will be able to --

CO1: Apply concepts related to embedded C programming.

CO2: Develop and Execute embedded C program to perform array addition, block transfer, sorting operations

CO3: Perform interfacing of real-world input and output devices to PIC18FXXX microcontroller.

CO4: Use source prototype platform like Raspberry-Pi/Beagle board/Arduino.

Guidelines for Instructor's Manual

The faculty member should prepare the laboratory manual for all the experiments and it should be made available to students and laboratory instructor/Assistant. The instructor's manual should include prologue, university syllabus, conduction & Assessment guidelines, topics under consideration- concept, objectives, outcomes, algorithm, sample test cases etc.

Guidelines for Student's Lab Journal

1. The laboratory assignments should be submitted by students in the form of journal. The Journal consists of Certificate, table of contents, and write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software & Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept, circuit diagram, pin configuration, conclusion/analysis).

2. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of program listing to journal may be avoided.

3. Use of Digital media like shared drive containing students' programs maintained by lab In-charge is highly encouraged.

4. Practical Examination will be based on the term work submitted by the student in the form of journal.

5. Candidate is expected to know the theory involved in the experiment.

6. The practical examination should be conducted if the journal of the candidate is completed in all respects and certified by concerned faculty and head of the department.

7. All the assignment mentioned in the syllabus must be conducted.

Guidelines for Lab /TW Assessment

1. Examiners will assess the term work based on performance of students considering the parameters such as timely conduction of practical assignment, methodology adopted for

INDEX

SCHEDULE

Sr. No.	Title	No. Of Hrs.	Week
1	Write an Embedded C program to add array of n numbers.	2	1
2	Write an Embedded C menu driven program for 8-bit Multiplication &Division	2	2
3	Write an Embedded C program for sorting the numbers in ascending and descending order.	2	3
4	Write an Embedded C program to interface PIC 18FXXX with LED & blinking it using specified delay.	2	4
5	Mid-term Mock & Partial Submission	2	5
6	Write an Embedded C program for Timer programming ISR based buzzer on/off.	2	6
7	Write an Embedded C program for LCD interfacing with PIC 18FXXX.	2	7
8	Write an Embedded C program for Generating PWM signal for servo motor/DC motor.	2	8
9	Write an Embedded C program for PC-to-PC serial communication using UART.	2	9
10	Study of Arduino board and understand the OS installation process on Raspberry-pi.	2	10
11	Write simple program using Open-source prototype platform like Arduino for digital read/write using LED and switch, Analog read/write using sensor and actuators.	2	11
12	Final Mock & submission	2	12

Assignment No. 1: Array Addition

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

Objectives:

1. To study basics of Embedded C programming i.e. Software development tools required commands to use them.
2. To study steps in Embedded C programming.

Apparatus:
PC, MPLABX IDE.

Theory:

PIC18F4550 is 8-bit microcontroller developed by 'MICROCHIP' based on RISC
architecture. This microcontroller has total of 33 I/O pins. These I/O pins are distributed among five I/O ports for interfacing various I/O devices. I/O ports are multiplexed with an alternate function from the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

The following table shows names and number of I/O pins of these 5 ports:

Port Name	No. of Pins	Pins
Port A	6	RA0-RA5
Port B	8	RB0-RB7
Port C	8	RC0-RC7
Port D	8	RD0-RD7
Port E	3	RE0-RE2

Each port has three registers for its operation. These registers are:
• **TRISx** register (data direction register), 1 : input, 0 : output
• **PORTx** register (reads the levels on the pins of the device)
• **LAT**x register (output latch)

An array is a contiguous block of list of data in memory. Each element of the list must be the same type and use the same number of bytes of memory for storage. Because of these properties, arrays allow efficient access of the data by its position (or index) in the array.

1. ARRAY Declaration
A group of variables of the same type is called an array. Elements of an array are called components, while their type is called the main type. An array is declared by specifying its name, type and the number of elements it will comprise:

```
component_type array_name [number_of_components];
```

2. **For LOOP declaration**

The **for** loop looks as follows:

```
for(initial_expression; condition_expression; change_expression) {
    operation;
    ...
}
```

As we want to output the result of array addition on port B, PORTB must be programmed as output. So, direction register TRISB=0. Assign the sum to PORTB

Procedure:

STEPS TO BE FOLLOWED FOR PROGRAMMING WITH PIC

1. 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 OpenProject

2. Adjusting application for Boot loader.

Step 1: Go to Project window and Right Click on the Project folder.
Go to Properties.
Select XC8 linker.
In Option categories ->Select memory model.In Code offset section type 800
Click Apply button and OK button.

3. Compiling Project.

Step1: Go to project window.

- Right Click on the project folder and select Build or Clean and Build.

How to see the Result:

1. Write the program
2. Debug the project – Check for the errors
   If No errors – Build successful will appear on the OUTPUT window
3. Go to Debug window --- Discreet debug operation
   - build for debugging
   - Launch for debugger option
   - For single stepping press F8
4. To see the result: click on Window tab on Navigation bar
   - ■ Click on PIC Memory Views
   - ■ Select option required
     - ■ Window will appear go to required location andobserve the result by doing single stepping

Algorithm:

1. Declare the array of 'n' numbers.
2. Initialize sum = 0
3. i=1
4. sum = sum + number(i)
5. i=i+1
6. if n<5, go to step 4
7. Configure PORT B as output port
8. Assign Sum to PORTB register.
9. End

Input: Array of ‘n’ numbers
Output: Sum of ‘n’ numbers

Conclusion:

Program is executed in MPLAB IDE and output is observed in SFR, LATB.

Assignment No. 2: Multiplication & Division

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:
PC, MPLABX IDE.

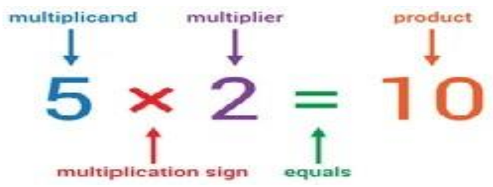
Theory:

Registers of PIC 18FIC4550:

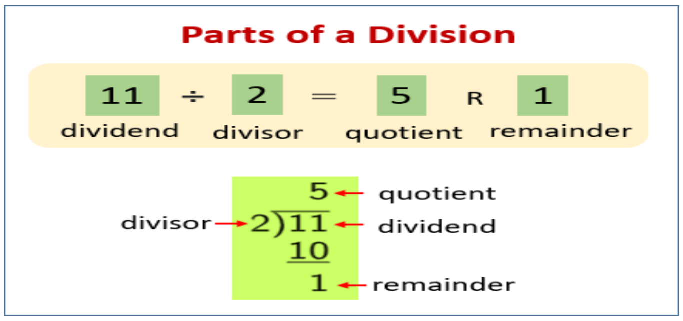
The memory of PIC is divided into series of registers. Each of register has its own address & memory location. According to type of working & usage register in PIC are classified as-

- Special Function Registers (SFRS): Used for control & status of controller & peripheral function
- WREG (Working register acts as an accumulator): Used to perform arithmetic & logical operations
- Status register that stores flags:
- Indicate operation that is done by ALU
- Register hold memory address
- Bank select register (BSR): 4-bit register that hold program in direct addressing data memory
- File select register (FSR): 16-bit register used as memory pointer in indirect addressing data memory
- Program Counter (PC): 21-bit register that hold program memory address while executing the program. This means PIC 18F can access program address 000000H to IFFFFFH total 21 bytes of code
- Stack Pointer (SP): PIC18F has 5-bit stack pointer It is used to access to stack

Logic of Multiplication using successive addition:
In successive addition method, one number is accepted, and other number is taken as a counter. The first number is added with itself, till the counter decrements to zero.
Eg: Here 5 is repeated 2 times, we can write this addition as (5 +5=10).



Logic of Division using successive subtraction:
Division is just counting how many times you can subtract on number (**divisor**) from another number (**dividend**). The number of times you can subtract is called the **quotient** and any number smaller the divisor remaining is called the **remainder**.



Condition to be checked	Dividend= dividend-divisor	New dividend = dividend-divisor	Q
As dividend > divisor, sub	11-2	9	0+1=1
As dividend > divisor, sub	9-2	7	1=1=2
As dividend > divisor, sub	7-2	5	2+1=3
As dividend > divisor, sub	5-2	3	3+1=4
As dividend > divisor, sub	3-2	1	4+1=5
Now dividend < divisor, stop do not sub	1	Remainder= 1	Quotient =5

Algorithm

Declare the Multiplicand & Multiplier**,** Dividend & Divisor

if choice =1 perform multiplication operation else if choice=2 perform division operation

Multiplication

1) Get the first number, multiplicand
2) Initialize the second number (multiplier) as a counter
3) Initialize Result = 0
4) Result= Result + first number
5) Decrement counter
6) If counter>0, then repeat step 4
7) Configured PORT B as output port
8) Assign result to PORTB
9) Terminate program

Division

1) Initialize dividend, divisor
2) Initialize quotient=0, remainder=0, and i=0
3) for(i=0; dividend>=divisor; i++)
4) Dividend=Dividend - divisor
5) Quotient = Quotient +1
6) If dividend > divisor, go to step 4
7) remainder = dividend
8) Configured PORT B and C as output ports
9) Assign Quotient to PORTB
10) Assign remainder to PORTC
11) Terminate program

Input: multiplicand, multiplier dividend, divisor

Output: product, quotient, remainder

Conclusion:

Assignment No. 3: Sorting

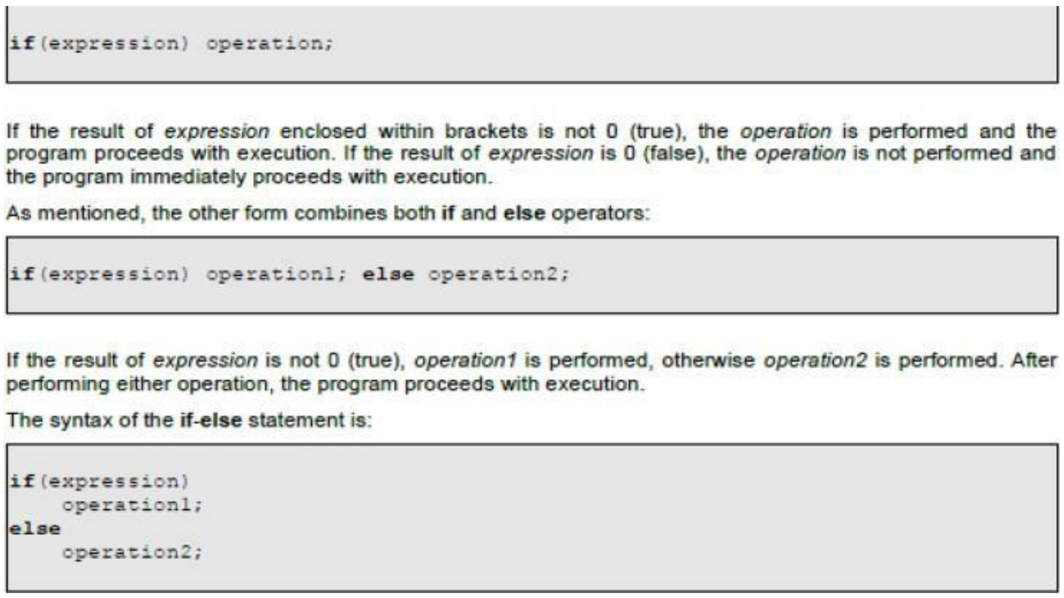
Aim:
Write an Embedded C program for sorting the numbers in ascending and descending order.

Apparatus:
PC, MPLABX IDE.

Theory:
Bubble Sort is a simple algorithm, which is used to sort a given set of 'n' elements provided in form of an array with 'n' number of elements. Bubble Sort compares all elements one by one and sort them based on their values. If the given array is to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will swap both the elements, and then move on to compare the second and the third element, and so on. If we have total **n elements**, then we need to repeat this process for **n-1 times**. It is known as **bubble sort**, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises to the water surface. Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

1. Conditional Operator:

The conditional operator can appear in two forms - as **if** and **if-else** operator. Here is an example ofthe **if** operator



```
if(expression) operation;
```

If the result of *expression* enclosed within brackets is not 0 (true), the *operation* is performed and the program proceeds with execution. If the result of *expression* is 0 (false), the *operation* is not performed and the program immediately proceeds with execution.

As mentioned, the other form combines both **if** and **else** operators:

```
if(expression) operation1; else operation2;
```

If the result of *expression* is not 0 (true), *operation1* is performed, otherwise *operation2* is performed. After performing either operation, the program proceeds with execution.

The syntax of the **if-else** statement is:

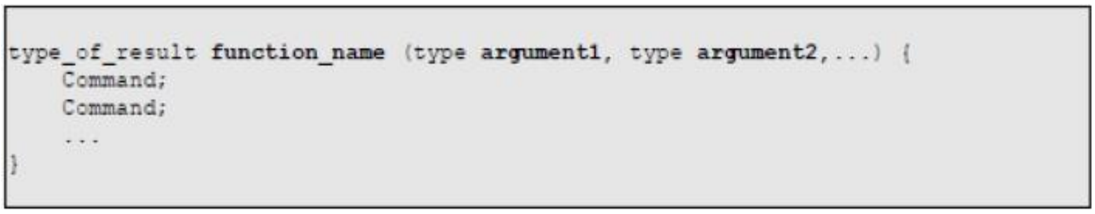
```
if(expression)
    operation1;
else
    operation2;
```

2. FUNCTIONS Declaration

Declaration contains the following elements:

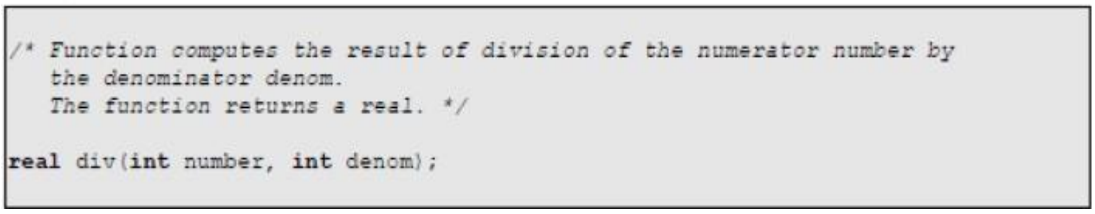
- Function name
- Function body
- List of parameters
- Declaration of parameters

• Type of function result. This is how a function looks like



```
type_of_result function_name (type argument1, type argument2,...) {
    Command;
    Command;
    ...
}
```

Example:



```
/* Function computes the result of division of the numerator number by
   the denominator denom.
   The function returns a real. */

real div(int number, int denom);
```

Following are the steps involved in bubble sort for sorting a given array in ascending order:

1. Starting with the first element (index = 0), compare the current element with the next element of the array.
2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the next element. Repeat Step1.

Algorithm: Ascending order

1. Declare the array num of 'n' numbers.
2. i=0,
3. j=j+1
4. if num[i]>num[j] then exchange their position
5. i=i+1
6. j=j+1
7. if i<=n, repeat steps 5,6,7
8. end

Algorithm: Descending order

1. Declare the array num of 'n' numbers.
2. i=0,
3. j=j+1
4. if num[i]>num[j] then exchange their position
5. i=i+1
6. j=j+1
7. if i<=n, repeat steps 5,6,7
8. end

Input: The unsorted order array

Output: Sorted Array in ascending & Descending order

Conclusion: Program is executed in MPLAB IDE and output is observed in File register window as shown above.

Assignment No. 4: LED Interfacing

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

Apparatus:

PIC18F4550 Development board, adaptor, USB cable.

Software used:

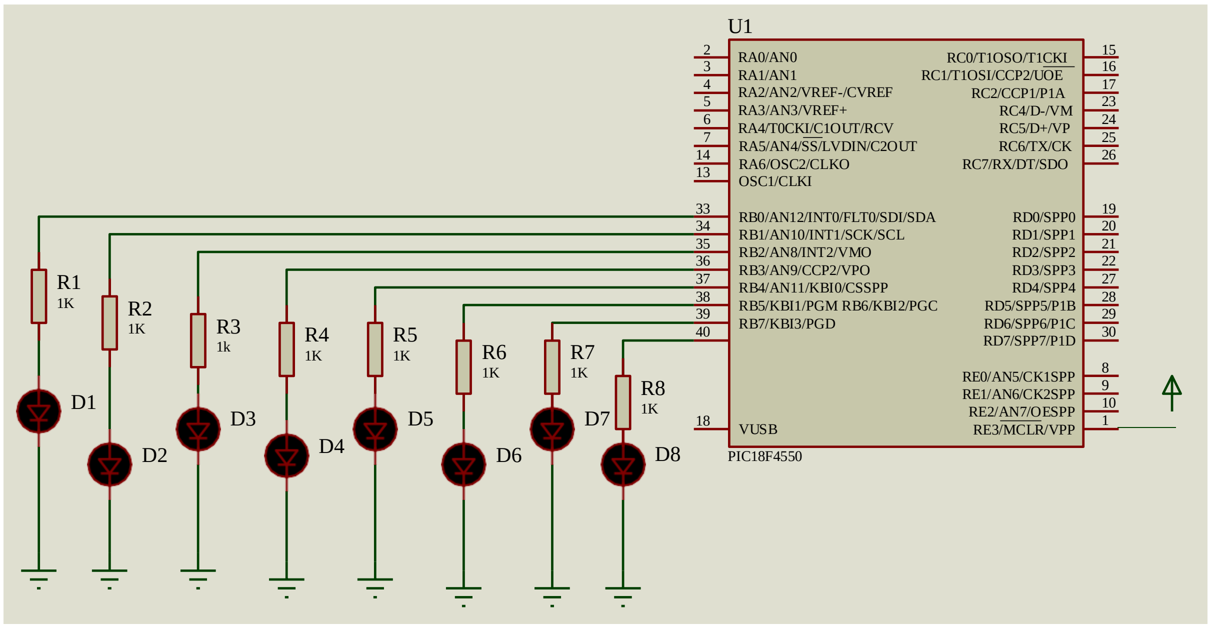
- MPLABX IDE
- XC8 Compiler
- PICLoader

Theory:

A light-emitting diode (LED) can be interfaced with PIC18FXXX Microcontroller for various purpose. The Port Pins are configured in the output direction. So, once the Logic "1" is the at O/P pin, then an LED will be turned "ON". And when Logic "0" is given the LED will be turned "OFF". Since the current sinking and sourcing capacities in the PIC18 family is about 25mA, we can drive LED using PORT pins.

Interfacing Diagram:

LEDs are connected to pin0 to pin7 of port B (RB0 to RB7).



Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.
Step2: Write the program in C language for interfacing LEDs to PIC18F4550. **(in program properties make sure to add the 0x800 offset)**
Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to thePC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

Executable Flashing Tool (PICLoader.exe).

- Connect the USB Cable to the PIC18F Board.
- Double Click the PICloader.exe.
- Go to Programs -> Settings.
- Select the USB or serial com port. Click OK.

Step3: Go to Programs -> Break/Reset Mode or Press F3.
Step4: Press the Reset Switch on the Micro-PIC18F Board.
Step5: Go to Programs ->Bootloader Mode or
Press F4.
Step6: Select Hex file :
File -> Open -> Browse to location.
Project folder -> dist -> default->production. - Double click on every
Step7: Go to Programs -> Write Device or Press F6
Step8: Press Reset on the board to Run the program

Step6: Press reset button and execute the program.

Result: Check if the LEDs are blinking. You can change the delay and vary the blinking rate.

Algorithm:

1. Program port B as output.
2. Send FFH on port B to Turn ON the LEDs.
3. Call delay
4. Send 00H on port B to Turn OFF the LEDs.
5. Repeat steps 2 to 4.
6. End

Conclusion:

LED is interfaced to PIC micro controller and observed the blinking LEDs.

FAQs:

Which are the RISC feature included in PIC microcontroller?
What is the need of delay in flashing the LED?
What is the role of TRIS register in programming the I/O ports?

Assignment No. 5: Timer

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

Apparatus:

PIC18F4550 Development board, adaptor, USB cable.

Software used:

- MPLABX IDE
- XC8 Compiler
- PICLoader

Theory:

Timers are common features of most microcontrollers. In simplified terms a timer is just a register whose value keeps increasing (or decreasing) by a constant rate without the help of the CPU. The CPU can read or write this register any time. If reads it find out how much time has elapsed. The Timer register can have the following bit length.

- 8-bit timers – These can count between 0-255
- 16-bit timers – These can count between 0-65535

A timer has a clock source, for example of the clock source of 10KHz is input to a timer then one increment will take 100 microseconds. This clock source can be obtained from the CPU clock. The CPU clock of popular MCU ranges from 1 MHz to 20 MHz, this can be sometimes too fast. To help us out there is a thing called prescaler in the MCU. The job of prescaler is to divide the CPU clock to obtain a smaller frequency. The PIC Micro that we will use in this example has the following prescaler division factors available: 256, 128, 64, 32, 16, 8, 4, 2, 1.

Timers are also called Counters this is because they can be used to count

external events. **Overflow:**

An overflow occurs when a timer register has already counted the maximum value it can count. At overflow the counter value become 0 again. For example, when an 8-bit timer has the value 255 and receive another clock that will set it to 0 and generate an overflow. An overflow can trigger an interrupt and the ISR can handle it.

Interrupts:

Interrupts, as the name suggests *interrupts the* normal execution and Requests and urgent attention of CPU. Interrupts are situations that the CPU can’t predict when they will happen, they can happen any time, so the CPU does not wait for them. So, the CPU keeps on doing its normal job unless and interrupt occurs. But as soon as the device interrupts to CPU, the CPU save its current state (so that it can resume) and jumps to the ISR (interrupt service routine) immediately. Where we can process the command or put it in a FIFO queue (to process latter). The ISR is generally kept very small and fast. As soon as the ISR ends, the CPU restores its saved state and resumes where it left. In this way CPU does not miss any data byte.

Sources of interrupts in PIC18:

There are many sources of interrupts in the PIC18. Following are some of the most widely used interrupts in the PIC18:

- ***Internal peripheral sources:***
  - Interrupt set aside for each of the Timers, Timer0, Timer1, Timer2, and so on.
  - Serial communication's USART has two interrupts, one for receive and another for transmit. ADC (analog to digital converter).
  - CCP (compare capture pulse width modulation, PWM)

- ***External sources:***
  - Three Interrupts are set aside for External hardware interrupts. Pins RB0 (PORTB.0), Pins RB1 (PORTB.1), and Pins RB2 (PORTB.2) are for INT0, INT1 and INT2 (Can be used to connect external interrupting sources: Keypads or switches).
  - The PORTB-Change interrupt (Change in logic levels of pins RB4-RB7)

TIMER interrupts – They are also very common in MCUs. Today's MCUs comes with very sophisticated timers that can do lots of magic for you. They have related interrupts. In most simple situation they can act like alarm clocks that can interrupt the CPU at predefined intervals. If you toggle a i/o pin in response to these alarms (interrupts), what you have is a frequency generator.

Interrupts management:

In general, each interrupt source has following related bits:

Enable Bit – They are suffixed with IE (Interrupt Enable), example TMR0IE stands for TIMER0 Interrupt Enable. It can be used to enable/disable the related interrupt. When set to '1′ it enables the interrupt.

Flag Bit – It is set automatically by the related hardware when the interrupt condition occurs. It is generally suffixed with IF (Interrupt Fag). When it is set to '1' we know that interrupt has occurred. For example, when TMR0IF is set by TIMER0, it indicates that TIMER0 has overflowed.

Priority Bit - We can leave this for now to keep things simple. We won't be using interrupt priority feature of PIC18.

The interrupts must be enabled (unmasked) by the software for the uC to respond to them. In global level there are following bits to control the interrupts globally:

GIE (Global Interrupt Enable) - The D7 bit of INTCON (interrupt control) register enables/disables interrupts globally.

PEIE (Enable/disable peripheral interrupts)- For some of the peripheral interrupts such as TMR1IF, TMR2IF and TXIF, we must enable the PEIE flag in addition to the GIE bit.

Interrupt Vector

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microcontroller runs the interrupt service routine. Generally, in most microprocessors, for every interrupt there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called the *interrupt vector table*. In the case of the PIC18, there are only two locations for the interrupt vector table, locations 0008 and 0018, as shown in Table.

Table 11-1: Interrupt Vector Table for the PIC18

Interrupt	ROM Location (Hex)
Power-on Reset	0000
High Priority Interrupt	0008 (Default upon power-on reset)
Low Priority Interrupt	0018 (See Section 11.6)

It is the address where the CPU jumps when an interrupt occurs. In PIC18 this address is **0008H**. CPU jumps to this address for any interrupt. The ISR or Interrupt service routine is placed in this address. The ISR must determine the source of interrupt from the flag bit (described above).

void interrupt ISR() is the main service routine where the CPU jumps for any interrupt (TIMER, ADC, INTx etc.). As soon as we enter the ISR we check for the source of interrupt. This done using the "**if(TMR0IE && TMR0IF)".**

The above line check is the source of interrupt was TIMER0 (so it checks if TMR0IF is set or not). It also checks TMR0IE (TIMER0 interrupt enable bit) to make sure this interrupt is enabled or not. It executes the TIMER0 ISR only if both the conditions are true. We must clear the flag; this is necessary otherwise the interrupt is triggered again.

Timer Calculations:

To generate a delay of 1ms on RA5 using Timer0 ISR. Assume XTAL = 48 MHz:

- Given: Fosc = 48 MHz, No prescaler
- fTimer = Fosc/4=48 MHz /4 =12 MHz
- Timer clock Period = 1/12 MHz = 0.083 us
    - required delay = 1ms
- Count = desired delay/Timer period = 1 ms/0.0833 us = $(12000)_{10}$
    - Value to be loaded in Timer register is: 65536 – count = 65536 – 12000= 53536= D120H
- TMR = 0xD120
- TMRH = 0XD1
- TMRL = 0x20

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for interfacing Buzzer to PIC18F4550, using Timer ISR. (in program properties make sure to add the 0x800 offset)

Step3: Build the program and create hex file. In case of errors correct program andrebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to thePC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

Step6: Press reset button and execute the program.

Result: Check if the buzzer is sounding ON/OFF and the ISR is getting executed with the specified timer delay. You can change the delay and vary the sounding rate.

Algorithm:

Main Program

1. Define RA5 pin of port A as buzzer pin

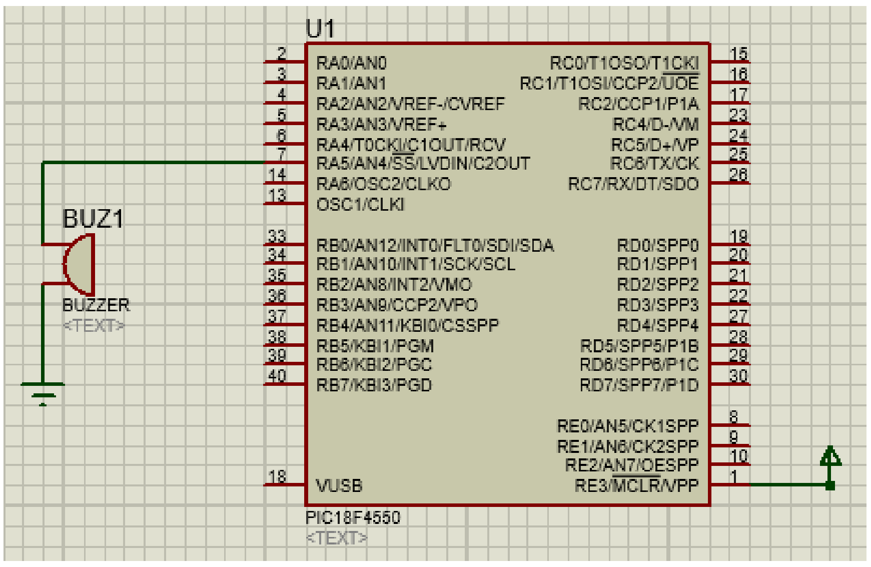
2. Initialize counter
3. Program buzzer pin RA5 of port A as output.
4. Enable Global Interrupt
5. Enable Peripheral Interrupt
6. Enable Timer1 Overflow Interrupt
7. Clear Timer1 flag bit
8. Set T1CON register for the specifications: Enable 16-bit TMR1 register, no pre-scale, internal clock, timer OFF
9. Load initial values in TMR1L & TMR1H register to generate delay of 1 ms
10. Start Timer1
11. End

ISR Routine

1. Timer1 flag bit = 1 then reload initial values in TMR1L & TMR1H register to generate delay of 1 ms
2. Increment counter to generate delay of 1Sec
3. Toggle buzzer pin
4. reset counter
5. Clear timer1 overflow flag bit to 0

Interfacing Diagram:

The buzzer is connected to **RA5** pin.



Conclusion:

What are the advantages of interrupt method to generate delay?
What is the value of TMR0L and TRM0H to generate delay of 100msec if Fosc=48MHz and prescaler value is 256? Q.3 Explain T0CON register.

Assignment No. 6: LCD Interfacing

Aim: Write an Embedded C program to interface PIC18F4550 to 16x2 Character LCD.

Objectives:

1. Draw and explain interfacing of 16x2 LCD with PIC18FXXX microcontroller in 8-bit mode
2. Write an Embedded C Program to display word "Hello".

Apparatus:

PIC18F4550 Development board, adaptor, USB cable.

Software used:

- MPLABX IDE
- XC8 Compiler
- PicLoader/Proteus Simulation

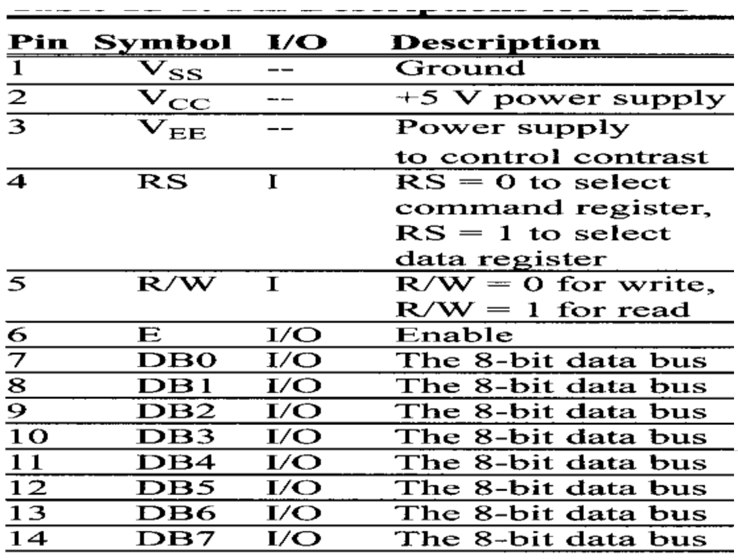
Theory:

Liquid Crystal Display (LCD) is very commonly used electronic display module and having a wide range of applications such as calculators, laptops, mobile phones etc. Now a days LCD modules are available which have built-in drivers for LCD and interfacing circuitry to interface them to microcontroller system. These LCD modules allow display of characters as well as numbers.

They are available in 16x2, 20x1, 20x1, 20x4 and 40x2 sizes. 16×2 Liquid Crystal Display which will display the 32 characters at a time in two rows (16 characters in one row). Each character in the display of size 5×7pixel matrix. LCD can be interfaced with microcontroller in 4 Bit or 8 Bit mode. These differs in how data is sent to LCD. In 8-bit mode to write a character, 8-bit ASCII data is sent through the data lines D0 – D7 and E (high to low pulse), this pin is used by LCD to latch information available at its data pins. LCD commands which are also 8 bits are written to LCD in similar way. The LCDs have a parallel interface, meaning that the microcontroller must manipulate several interface pins at once to control the display. The interface consists of the following pins:

There are 16 pins in the LCD module, the pin configuration is given below.

Table: Pin Description for LCD



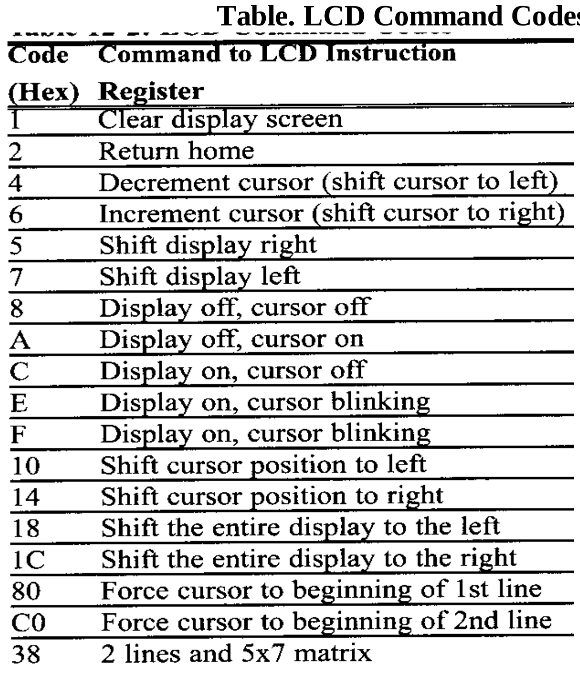
Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5 V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit 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

All the pins are clearly understandable by their name and functions, except the control pins, so they are explained below:
RS: RS is the register select pin. We need to set it to 1, if we are sending some data to be displayed on LCD. And we will set it to 0 if we are sending some command instruction like clear the screen (hex code 01).
RW: This is Read/write pin, we will set it to 0, if we are going to write some data on LCD. And set it to 1, if we are reading from LCD module. Generally, this is set to 0, because we do not have need to read data from LCD. Only one instruction "Get LCD status", need to be read sometimes.
E: This pin is used to enable the module when a high to low pulse is given to it. A pulse of 450 ns should be given. That transition from HIGH to LOW makes the module ENABLE. There are some preset command instructions in LCD, we have used them in our program below to prepare the LCD (in lcd_init() function). Some important command instructions are given below:

LCD Commands

The process of controlling the display involves putting the data that form the image of what youwant to display into the data registers, then putting instructions in the instruction register.

Table. LCD Command Codes



Code (Hex)	Command to LCD Instruction Register
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

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.
Step2: Write the program in C language for interfacing 16x2 LCD to PIC18F4550.
(in program properties make sure to add the 0x800 offset)
Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.
Step4: Prepare the experimental setup by connecting the MicroPIC18F board to the PC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.
Step6: Press reset button and execute the program.
Result: Check if the characters are getting printed on the LCD screen.

Algorithm:

1. Configuration:
   Oscillator: Hs, Power ON timer Watchdog timer: OFF, Debug: OFF, Low vol. protect: OFF
2. Program Port A and Port B as output port.
3. Select function set for : 16 x 2 , 8 bit mode, 2line 5x7 Dot matrix for each character
4. Give delay of 50ms
5. Make display ON cursor ON
6. Give delay of 15ms
7. Clear display
8. Give delay of 15ms
9. Increment cursor and shift right
10. Give delay of 15ms
11. Force cursor to first row first position on LCD
12. Display first character
13. Give delay of 50ms
14. Display second character
15. Give delay of 50ms
16. End

Algorithm for function lcdcmd(unsigned char value) (Function to issue command to LCD)

1. Send command to PORT A
2. Select command register by making rs=0
3. Give write signal to LCD by making r/w=0
4. Enable LCD by making E=1
5. Wait for 1 ms delay
6. Disable LCD by making E=0
7. End

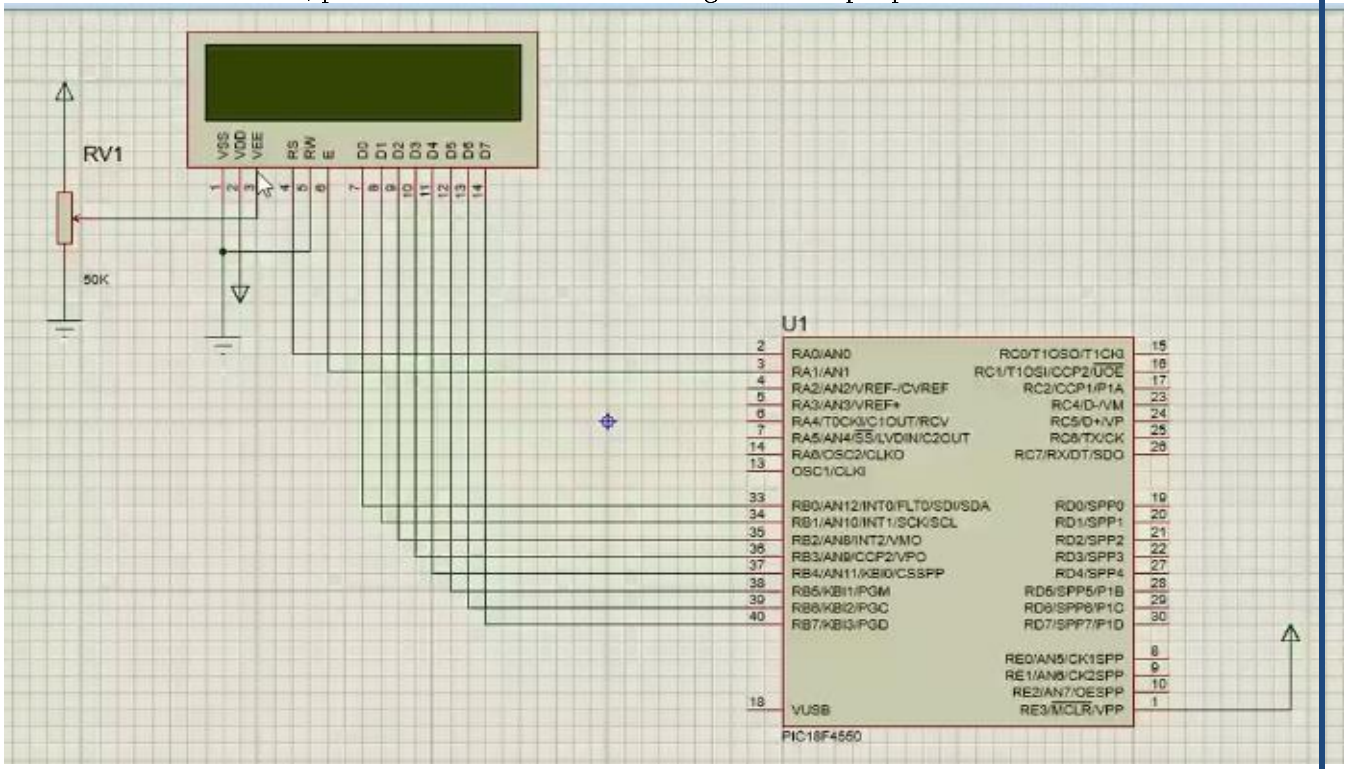
Algorithm for function lcddata(unsigned char value) (Function to display character)

1. Send ASCII code of character to display to PORT B
2. Select data register by making rs=1
3. Give write signal to LCD by making r/w=0
4. Enable LCD by making E=1
5. Wait for 1 ms delay
6. Disable LCD by making E=0
7. End

Interfacing Diagram:

- The figure shows the interfacing of a 16 character x 2 line LCD module with the PIC 18. Crystal oscillator is connected between OSC1 and OSC2 pin which provides clock frequency to microcontroller. Pin 1, MCLR is connected to reset circuit with push button to reset the microcontroller. Pin 1 of LCD is grounded. Pin 2 is connected to VCC, and pin 3 VEE is connected to potentiometer to adjust contrast of the LCD.
- As shown in the Fig. the 8 bit data bus DB0-DB7of LCD are connected to the PORTB of PIC 18Fxxx and control lines Pin 4 (RS) is connected RA0, pin 6 (E) to RA1 and pin

5 (R/W) grounded. So various commands and data can be given to LCD through port A and Port B. So, port A and Port B are to be configured as output ports.



Conclusion:

Observed displayed message by interfacing LCD to PIC micro controller.

FAQs:

1. Give the status of RS, EN and R/W when sending data character ‘Z’ to LCD.
2. How does the busy flag help in making the LCD program more efficient?
3. Write a program for 8-bit LCD to display “SCOE” on 1st line 5th position & “IT” on 2nd 2nd line 6th position.

Assignment No. 7: PWM signal for DC motor

Aim:

Write an Embedded C program to interface PIC18F4550 to DC motor and varying speed using PWM signal generation.

Objective:

- To understand and program the PWM module of PIC18F458 microcontroller.

Apparatus:

MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

Software used:

- MPLABX IDE
- XC8 Compiler
- PicLoader/Proteus Simulation

Theory:

Pulse Width Modulation (PWM) is a technique by which the width of a pulse is varied while keeping the frequency of the wave constant. The applications such as motor speed control, for encoding messages in telecommunication systems and for controlled switching in switch mode power supplies and for sound synthesis in audio amplifiers etc.

Duty Cycle:

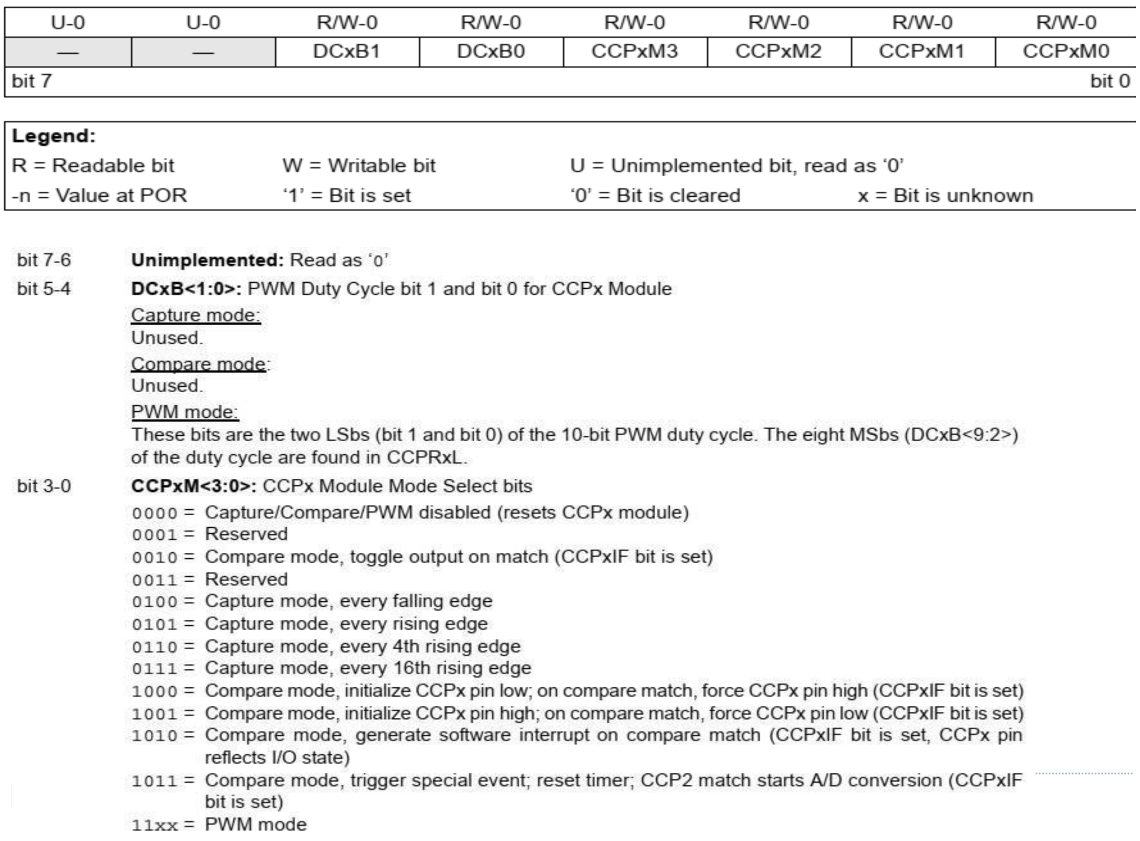
A period of a pulse consists of an ON cycle (5V) and an OFF cycle (0V). The fraction for which the signal is ON over a period is known as a duty cycle. E.g. A pulse with a period of 10ms will remain ON (high) for 2ms.Therefore, the duty cycle will be, D = 2ms / 10ms = 20%

Through the PWM technique, we can control the power delivered to the load by using the ON-OFF signal. The PWM signals can be used to control the speed of DC motors and to change the intensity of the LED. Moreover, it can also be used to generate sine signals.

In PIC18F458, only Timer2 can be used for PWM generation. TMR2 is a 8-bit Timer2 register which is used to hold the count.

Make the RC1(CCP2) & RC2(CCP1) pin as output for PWM generation

PWM module is associated with a control register (generically, CCPxCON) and a data register (CCPRx).



U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

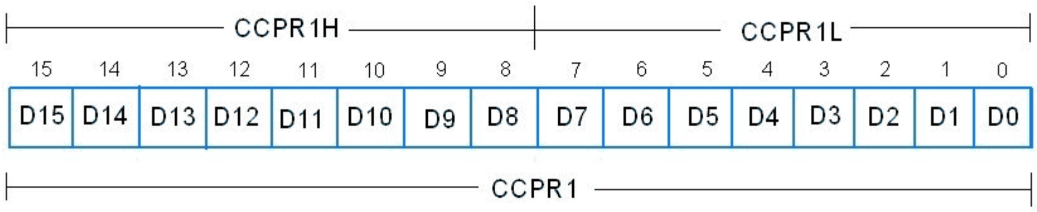
bit 7-6 **Unimplemented:** Read as ‘0’

bit 5-4 **DCxB<1:0>:** PWM Duty Cycle bit 1 and bit 0 for CCPx Module
Capture mode:
Unused.
Compare mode:
Unused.
PWM mode:
These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The eight MSbs (DCxB<9:2>) of the duty cycle are found in CCPRxL.

bit 3-0 **CCPxM<3:0>:** CCPx Module Mode Select bits

0000 =	Capture/Compare/PWM disabled (resets CCPx module)
0001 =	Reserved
0010 =	Compare mode, toggle output on match (CCPxIF bit is set)
0011 =	Reserved
0100 =	Capture mode, every falling edge
0101 =	Capture mode, every rising edge
0110 =	Capture mode, every 4th rising edge
0111 =	Capture mode, every 16th rising edge
1000 =	Compare mode, initialize CCPx pin low; on compare match, force CCPx pin high (CCPxIF bit is set)
1001 =	Compare mode, initialize CCPx pin high; on compare match, force CCPx pin low (CCPxIF bit is set)
1010 =	Compare mode, generate software interrupt on compare match (CCPxIF bit is set, CCPx pin reflects I/O state)
1011 =	Compare mode, trigger special event; reset timer; CCP2 match starts A/D conversion (CCPxIF bit is set)
11xx =	PWM mode

- In the CCP module, there is a 16-bit register which is split into two 8-bit registers - **CCPR1H** and **CCPR1L**.



- The CCP module uses Timer2 and its associated period register, PR2 (PR2 is a readable and writable register), for the PWM time-base, which means that the frequency of the PWM is a fraction of the Fosc, the crystal frequency. PR2 register is an 8-bit register that is used to load a count for a period of the pulse (TPWM).
- *It uses the PR2 register to set the PWM period using the formula:*

PR2= (Fosc/ Fpwm x 4× N) - 1
Where,
N: TMR2 prescaler value. Can be set to 1, 4 or 16 by programming Timer2 control register (T2CON)
Fpwm: PWM frequency

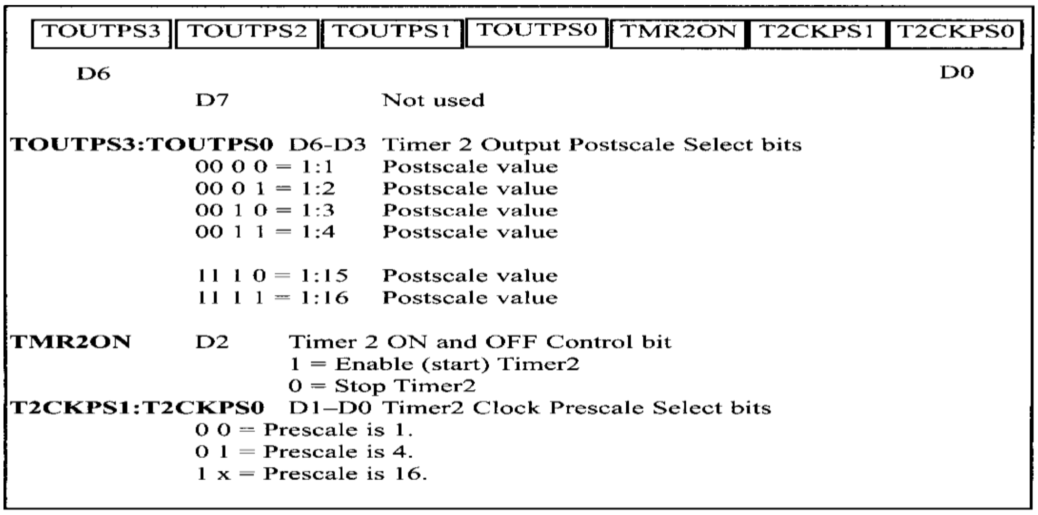
- In reality, CCPRL1 is the main register for the duty cycle and the lower 2 bits of **DCB2:DCB1 are for the decimal point portion of the duty cycle** and are set as follows:

CCP1CON<5:4>		Decimal point
DC1B2	**DC1B1**	
0	0	0
0	1	0.25
1	0	0.50
1	1	0.75

Now, let us see how to set a value for the CCPR1L which decides the duty cycle of a pulse. It must be noted that the value for the duty cycle register of the CCPRL1 is always some % of the PR2 (period) register.

- For example, if **PR2 is 95** and we need a 1**0 % duty cycle then** CCPRL1 is given by,
  (CCPR1L: CCP1 CON < 5:4>) = PR2 x Duty Cycle
  = (95) x (10/100) = 9.5 **= 0FH= 0x09**
  In this case, CCPR1L=09H & **DCB2:DCB1= 10.**

T2CON (Timer2 Control Register):



TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

D6 D0

D7 Not used

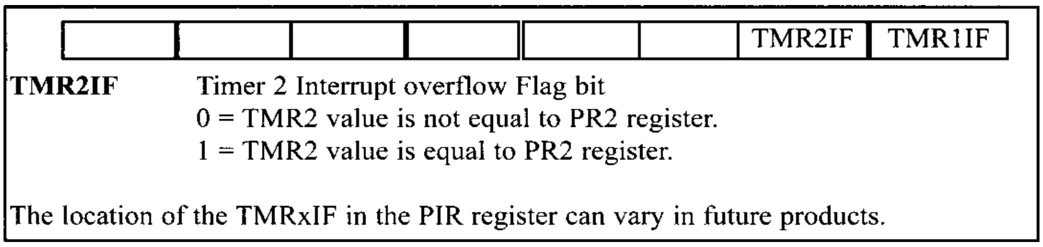
TOUTPS3:TOUTPS0 D6-D3 Timer 2 Output Postscale Select bits
00 0 0 = 1:1 Postscale value
00 0 1 = 1:2 Postscale value
00 1 0 = 1:3 Postscale value
00 1 1 = 1:4 Postscale value

11 1 0 = 1:15 Postscale value
11 1 1 = 1:16 Postscale value

TMR2ON D2 Timer 2 ON and OFF Control bit
1 = Enable (start) Timer2
0 = Stop Timer2

T2CKPS1:T2CKPS0 D1–D0 Timer2 Clock Prescale Select bits
0 0 = Prescale is 1.
0 1 = Prescale is 4.
1 x = Prescale is 16.

PIR1 (Peripheral Interrupt Flag Register1):



						TMR2IF	TMR1IF

TMR2IF Timer 2 Interrupt overflow Flag bit
0 = TMR2 value is not equal to PR2 register.
1 = TMR2 value is equal to PR2 register.

The location of the TMRxIF in the PIR register can vary in future products.

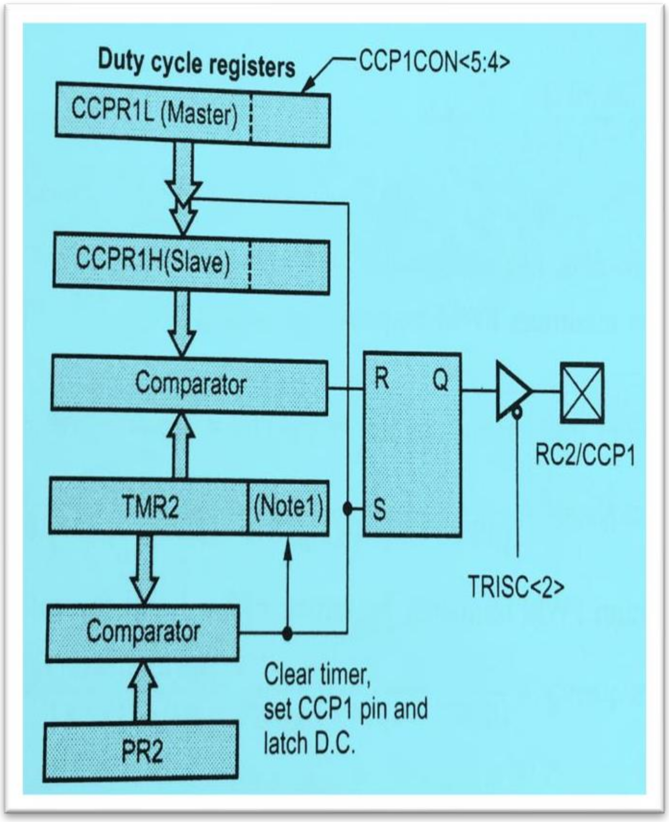
Working of CCP module in PWM mode

- The role of CCPR1H in the process of creating the duty cycle must be noted. A copy of the duty cycle value in register CCPR1L is given to CCPRIH as soon as we start Timer2.

- Timer2 goes through the following stages in creating the PWM:

(a) The CCPR1L is loaded into CCPR1H and the CCP1 pin goes HIGH to start the beginning of the period.
(b) As TMR2 counts up, the TMR2 value is compared with both the CCPR1H and PR2 registers.
(c) When the TMR2 and CCPR1H (which is the same as CCPR1L) values are equal, the CCP pin is forced low (R i/p of F/F = 1). That ends the duty cycle portion of the period.
(d) The TMR2 keeps counting up until its value matches the PR2. At that point, (S i/p of F/F = 1). the CCP pin goes high, indicating the end of one period and the beginning of the next one. It also clears Timer2 for the next round.

The CCPR1L is loaded into CCPR1H, and the process continues. See Figures.



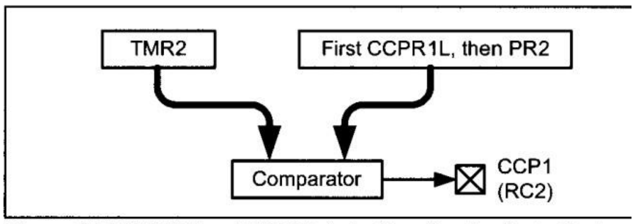


Figure 15-16. TMR2 and PR2 Role in Creating the Duty Cycle

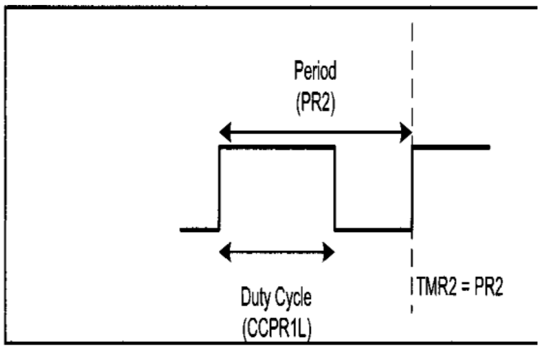


Figure 15-17. TMR2 Relation to CCPR1L and PR2 in PWM

PWM Programming steps:

- Following steps are needed to setup the PWM module in PIC18F458:

1. Set the PWM period by writing to the PR2 register.
2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
3. Set the CCP pin as output, by clearing the TRISC<2> bit.

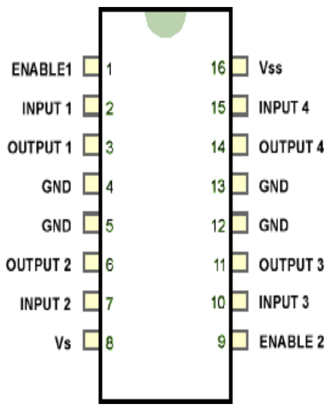
4. Using the T2CON register, set the TMR2 prescale value
5. Clear Timer2 (TMR2) register.
6. Configure the CCP1CON register for PWM and set DC1B2: DC1B1 bits for the decimal portion of the duty cycle.
7. Start Timer2 by writing to T2CON register.

DC Motor Control using PWM mode of CCP

- DC Motor Driver is a L293D based motor driver interface board.
- The main aim of interfacing DC motor with any microcontroller is to control the direction and speed of a DC motor.
- But due to high voltage and current requirement of DC motors, it cannot be interfaced directly with microcontrollers.
- For to interface DC motor with any microcontroller, we need a motor driver. Motor driver is basically a current amplifier which takes a low-current signal from the microcontroller and gives out a proportionally higher current signal which can control and drive a motor.
- L293D is a dual H-Bridge motor driver IC. With one L293D IC we can interface two DC motors which can be controlled in both clockwise and counter clockwise direction.

L293D:

- The **PWM mode** is used to control the speed of DC motors.
- Higher the duty cycle of the PWM signal higher is the speed of DC motor.
- The PIC18 microcontroller will generate PWM signals of different duty cycle using its PWM mode of CCP module and will give the PWM signal to the two enable pins of the L293D.
- Here, use the DC Motor Driver to control the speed of two DC motor.
- In this way, the microcontroller will run the DC motor in forward direction with different speeds.



Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for interfacing DC motor to PIC18F4550 and varying speed using PWM . **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program andrebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to the PC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

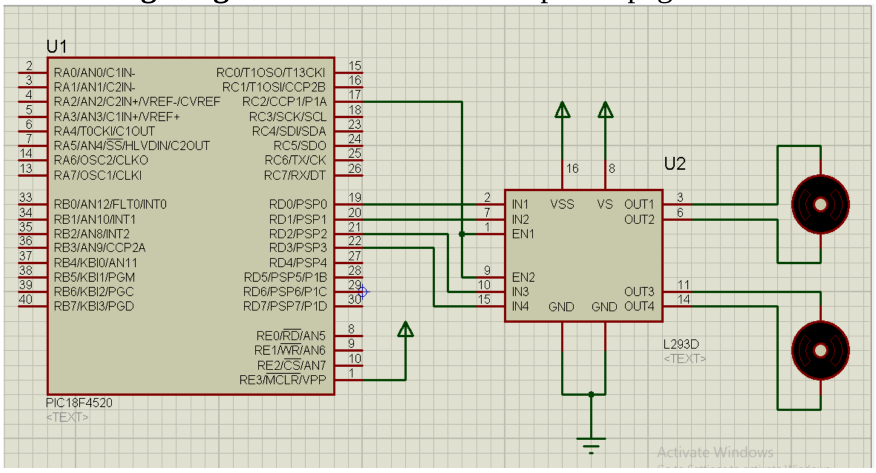
Step6: Press reset button and execute the program.

Result: Check if the DC motor speed varies.

Algorithm:

- START
- Configure RC0, RC2 pins as output pins
- Select PWM mode of operation by writing into CCP1CON
- Load initial value into PR2 period register
- Configure T2CON register for Prescaler = 16; Timer2 OFF
- Write value into CCPR1L (Duty cycle register) to generate 25% duty cycle waveform on pin RC2
- Provide some delay
- Write value into CCPR1L (Duty cycle register) to generate 50% duty cycle waveform on pin RC2
- Provide some delay
- Write value into CCPR1L (Duty cycle register) to generate 75% duty cycle waveform on pin RC2
- Provide some delay
- Write value into CCPR1L (Duty cycle register) to generate 100% duty cycle waveform on pin RC2
- END

Interfacing Diagram: To be drawn on separate page



Conclusion: Motors speed varies by changing the duty cycle of PWM signal.

FAQs:

Q.1 Write a program to rotate DC motor clockwise if sw=0, and anti- clockwise if sw=1.

Assignment No. 8: PC to PC serial communication

Aim: Write an Embedded C program for PC to PC serial communication using UART.

Theory:

- The microcontroller is parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices.
- However, in many situations, parallel data transfer is impractical. For example, parallel data transfer over a long is very expensive.
- Hence, a serial communication is widely used in long distance communication. In serial data communication, 8-bit data is converted to serial bits using a parallel in serial out shift register and then it is transmitted over a single data line.
- The data byte is always transmitted with least significant bit first.
- Serial ports are a type of computer interface for serial communication that complies with the RS-232 standard.
- They are 9-pin connectors that relay information, incoming or outgoing, one byte at a time.
- Each byte is broken up into a series of eight bits, hence the term serial port.
- Serial ports are controlled by a special chip call a UART (Universal Asynchronous Receiver Transmitter)
- Serial communication is classified into three types of communication.

a. Simplex communication link: In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data.

b. Half duplex communication link: In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time.

c. Full duplex communication link: If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.

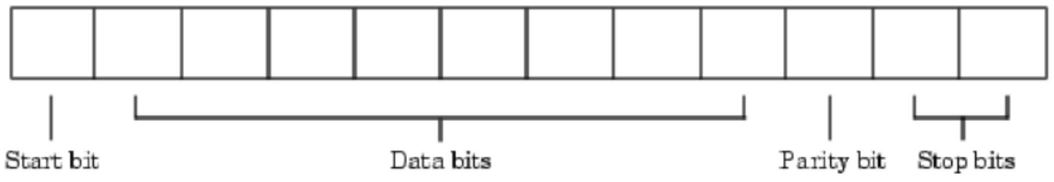
- **Serial data communication uses two types of communication:**
  - a. Synchronous serial data communication:
    - In this transmitter and receiver are synchronized. It uses a common clock to synchronize the receiver and the transmitter. First the synch character is sent and then the data is transmitted. This format is generally used for high speed transmission. In Synchronous serial data communication a block of data is transmitted at a time.
  - b. Asynchronous Serial data transmission:
    - In this, different clock sources are used for transmitter and receiver. In this mode, data is transmitted with start and stop bits. A transmission begins with start bit, followed by data and then stop bit. For error checking purpose parity bit is included just prior to stop bit. In Asynchronous serial data communication a single byte is transmitted at a time.

Baud Rate:

- The rate at which the bits are transmitted is called baud or transfer rate.
- The baud rate is the reciprocal of the time to send one bit. In asynchronous transmission, baud rate is not equal to number of bits per second.
- This is because; each byte is preceded by a start bit and followed by parity and stop bit.
- For example, in synchronous transmission, if data is transmitted with 9600 baud, it means that 9600 bits are transmitted in one second.
- For bit transmission time = 1 second/ 9600 = 0.104 ms.
  The serial data format :
  It includes one start bit, between five and eight data bits, and one stop bit. A parity bit and an additional stop bit might be included in the format as well.
- The format for serial port data is often expressed using the following notation: Number of

data bits - parity type - number of stop bits. For example, 8-N-1 is interpreted as eight data bits, no parity bit, and one stop bit, while 7-E-2 is interpreted as seven data bits, even parity, and two stop bits.

- The data bits are often referred to as a character because these bits usually represent an ASCII character. The remaining bits are called framing bits because they frame the data bits.



PIN Description:

Signal	Pin No.	Symbol
TXD	25	RC6/TX/CK
RXD	26	RC7/RX/DT/SDO

The pins of the Enhanced USART are multiplexed with PORTC. To configure RC6/TX/CK and RC7/RX/DT/SDO as an EUSART:
• bit SPEN (RCSTA<7>) must be set (= 1)
• bit TRISC<7> must be set (= 1)
• bit TRISC<6> must be set (= 0)

- **The operation of the Enhanced USART module is controlled through three registers:**

SFR	Description	Access	Reset Value	Address
TXSTA	Transmit Status & Control Register	Read/Write	0x02	0xFAC
RCSTA	Receive Status & Control Register	Read/Write	0x00	0xFAB
BAUDCON	Baud rate control register	Read/Write	0x00	0xFB8
SPBRGH	Baud rate generator register higher byte	Read/Write	0x00	0xFB0
SPBRG	Baud rate generator register lower byte	Read/Write	0x00	0xFAF
TXREG	Transmission Register	Write	0x00	0xFAD
RCREG	Receive Register	Read	0x00	0xFAE

Transmit Status & Control Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN[1]	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

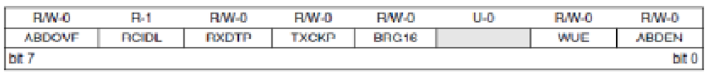
Bit No.	Control Bit	Description
Bit 7	CSRC	Clock Source Select bit Asynchronous mode: Don't care. Synchronous mode: 1 = Master mode; 0 = Slave mode
Bit 6	TX9	9-Bit Transmit Enable bit 1 = Selects 9-bit Tx; 0 = Selects 8-bit Tx
Bit 5	**TXEN**	**Transmit Enable bit** **1 = Transmit enabled ;** 0 = Transmit disabled
Bit 4	**SYNC**	**EUSART Mode Select bit** 1 = Synchronous mode; **0 = Asynchronous mode**
Bit 3	SENDB	Asynchronous mode: Send Break Character bit Synchronous mode: Don't care.
Bit 2	**BRGH**	**High Baud Rate Select bit** **Asynchronous mode: 1 = High speed;** 0 = Low speed Synchronous mode: Unused in this mode.
Bit 1	**TRMT**	**Transmit Shift Register Status bit** **1 = TSR empty; 0 = TSR full**
Bit 0	TX9D	9th bit of Transmit Data Can be address/data bit or a parity bit.

Receive Status & Control Register

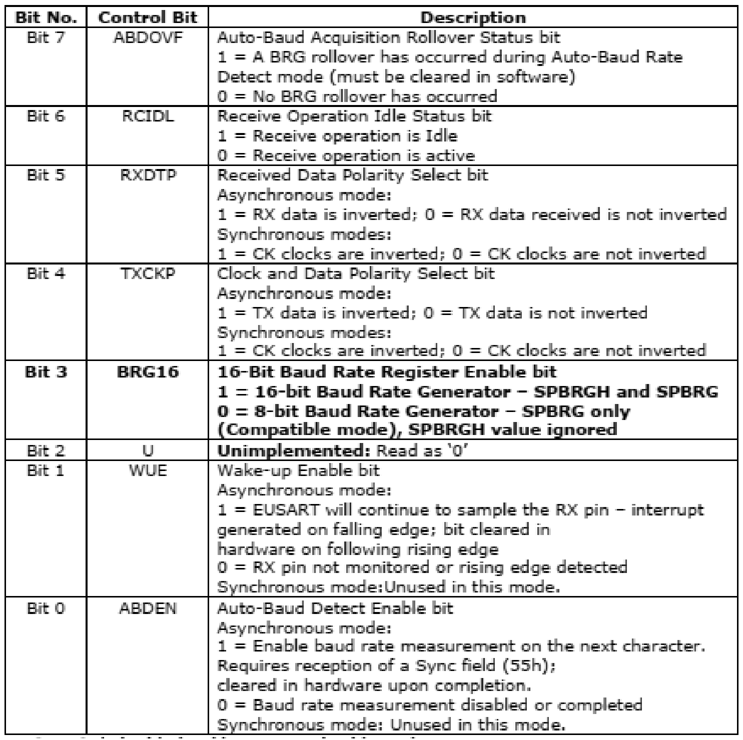
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Bit No.	Control Bit	Description
Bit 7	**SPEN**	**Serial Port Enable bit** **1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)** **0 = Serial port disabled (held in Reset)**
Bit 6	RX9	9-Bit Receive Enable bit 1 = Selects 9-bit reception; 0 = Selects 8-bit reception
Bit 5	SREN	Single Receive Enable bit Asynchronous mode: Don't care. Synchronous mode – Master: 1 = Enables single receive 0 = Disables single receive This bit is cleared after reception is complete. Synchronous mode – Slave: Don't care.
Bit 4	**CREN**	**Continuous Receive Enable bit** **Asynchronous mode:** **1 = Enables receiver** **0 = Disables receiver** Synchronous mode: 1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN) 0 = Disables continuous receive
Bit 3	ADDEN	Address Detect Enable bit Asynchronous mode 9-bit (RX9 = 1): 1 = Enables address detection, enables interrupt and loads the receive buffer when RSR<8> is set 0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit Asynchronous mode 9-bit (RX9 = 0): Don't care.
Bit 2	FERR	Framing Error bit 1 = Framing error (can be updated by reading RCREG register and receiving next valid byte) 0 = No framing error
Bit 1	OERR	Overrun Error bit 1 = Overrun error (can be cleared by clearing bit CREN) 0 = No overrun error
Bit 0	RX9D	9th bit of Received Data This can be address/data bit or a parity bit and must be calculated by user firmware.

Baud rate control register (BAUDCON)



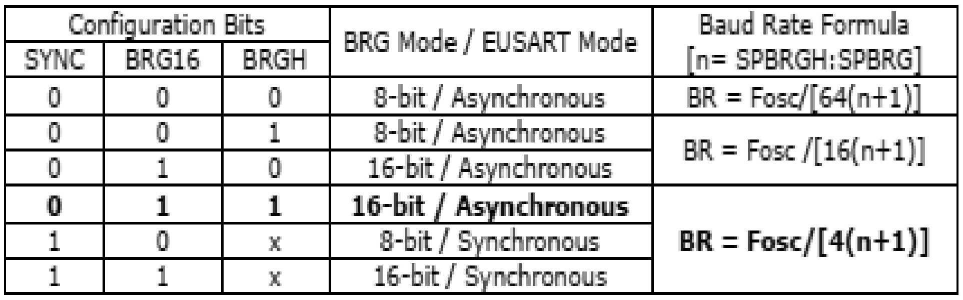
R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16		WUE	ABDEN
bit 7							bit 0



Bit No.	Control Bit	Description
Bit 7	ABDOVF	Auto-Baud Acquisition Rollover Status bit 1 = A BRG rollover has occurred during Auto-Baud Rate Detect mode (must be cleared in software) 0 = No BRG rollover has occurred
Bit 6	RCIDL	Receive Operation Idle Status bit 1 = Receive operation is Idle 0 = Receive operation is active
Bit 5	RXDTP	Received Data Polarity Select bit Asynchronous mode: 1 = RX data is inverted; 0 = RX data received is not inverted Synchronous modes: 1 = CK clocks are inverted; 0 = CK clocks are not inverted
Bit 4	TXCKP	Clock and Data Polarity Select bit Asynchronous mode: 1 = TX data is inverted; 0 = TX data is not inverted Synchronous modes: 1 = CK clocks are inverted; 0 = CK clocks are not inverted
Bit 3	**BRG16**	**16-Bit Baud Rate Register Enable bit** **1 = 16-bit Baud Rate Generator – SPBRGH and SPBRG** **0 = 8-bit Baud Rate Generator – SPBRG only (Compatible mode), SPBRGH value ignored**
Bit 2	U	**Unimplemented:** Read as '0'
Bit 1	WUE	Wake-up Enable bit Asynchronous mode: 1 = EUSART will continue to sample the RX pin – interrupt generated on falling edge; bit cleared in hardware on following rising edge 0 = RX pin not monitored or rising edge detected Synchronous mode:Unused in this mode.
Bit 0	ABDEN	Auto-Baud Detect Enable bit Asynchronous mode: 1 = Enable baud rate measurement on the next character. Requires reception of a Sync field (55h); cleared in hardware upon completion. 0 = Baud rate measurement disabled or completed Synchronous mode: Unused in this mode.

Baud Rate Generator (BRG)

- The BRG is a dedicated 8-bit, or 16-bit, generator that supports both the Asynchronous and Synchronous modes of the EUSART. By default, the BRG operates in 8-bit mode.
- Setting the BRG16 bit (BAUDCON<3>) selects 16-bit mode. The SPBRGH:SPBRG register pair controls the period of a free-running timer.
- In Asynchronous mode, bits BRGH (TXSTA<2>) and BRG16 (BAUDCON<3>) also control the baud rate.
- In Synchronous mode, BRGH is ignored.
- The table shows the formula for computation of the baud rate for different EUSART modes which only apply in Master Mode (internally generated clock). Given the desired baud rate and FOSC, the nearest integer value for the SPBRGH:SPBRG registers can be calculated using the formulas in Table.
- From this, the error in baud rate can be determined. It may be advantageous to use the high baud rate (BRGH = 1), or the 16-bit BRG to reduce the baud rate error, or achieve a slow baud rate for a fast oscillator frequency.
- Writing a new value to the SPBRGH:SPBRG registers causes the BRG timer to be reset (or cleared). This ensures the BRG does not wait for a timer overflow before outputting the new baud rate.



Configuration Bits			BRG Mode / EUSART Mode	Baud Rate Formula [n= SPBRGH:SPBRG]
SYNC	BRG16	BRGH		
0	0	0	8-bit / Asynchronous	BR = Fosc/[64(n+1)]
0	0	1	8-bit / Asynchronous	BR = Fosc /[16(n+1)]
0	1	0	16-bit / Asynchronous	
0	**1**	**1**	**16-bit / Asynchronous**	**BR = Fosc/[4(n+1)]**
1	0	x	8-bit / Synchronous	
1	1	x	16-bit / Synchronous	

Baud Rate Calculation

- Mode selection

  a. BRG Mode = 16-bit by setting BRG16 bit in BAUDCON Register

  b. EUSART mode = Asynchronous by clearing SYNC bit in TXSTA Register

  c. Formulae Desired Buad Rate = Fosc / [4(SPBRGH:SPBRG+1)] selected by setting BRG16 bit in TXSTA Register

  For Fosc = 20MHz, Baud rate = 9600

  SPBRGH:SPBRG = [Fosc/4(Baud rate)]-1

  = [20x106/(4x9600)] – 1

  = [519]10 = [0207]16

  % Error = (Calculated baud rate - Desired Baud Rate)/Desired Baud Rate

  = (Fosc / [4(SPBRGH:SPBRG+1)] – 9600)/Desired Baud Rate

  = ([20x106/[4x(519+1)]] – 9600)/Desired Baud Rate

  = (9615 – 9600)/9600

  % Error = 0.16 %

  Therefore to reduce error we have selected

  – SPBRGH = 0x02 and SPBRG = 0x08

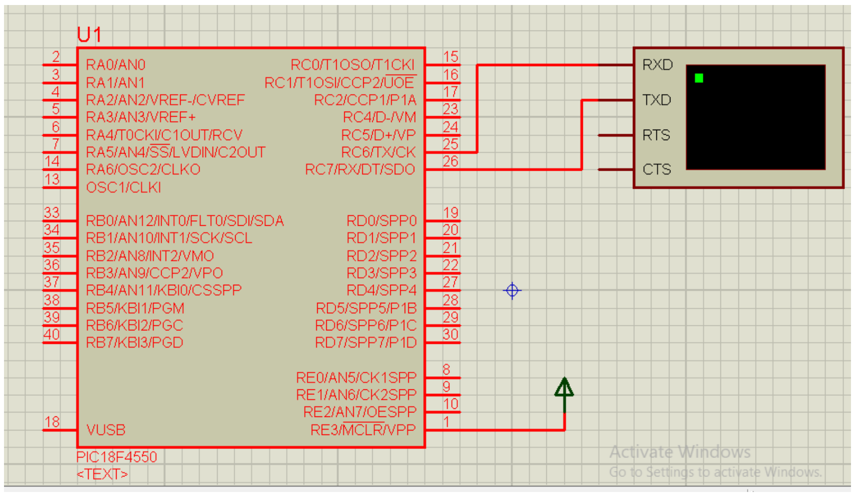
To set up an Asynchronous Transmission

1. Initialize the SPBRGH:SPBRG registers for the appropriate baud rate. Set or clear the BRGH and BRG16 bits, as required, to achieve the desired baud rate.
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If the signal from the TX pin is to be inverted, set the TXCKP bit.
4. If interrupts are desired, set enable bit TXIE.
5. If 9-bit transmission is desired, set transmit bit TX9.
6. Enable the transmission by setting bit TXEN which will also set bit TXIF.
7. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
8. Load data to the TXREG register (starts transmission).
9. If using interrupts, ensure that the GIE and PEIE bits in the INTCON register (INTCON<7:6>) are set.

To set up an Asynchronous Reception

1. Initialize the SPBRGH:SPBRG registers for the appropriate baud rate. Set or clear the BRGH and BRG16 bits, as required, to achieve the desired baud rate.
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If the signal at the RX pin is to be inverted, set the RXDTP bit.
4. If interrupts are desired, set enable bit RCIE.
5. If 9-bit reception is desired, set bit RX9.
6. Enable the reception by setting bit CREN.
7. Flag bit, RCIF, will be set when reception is complete and an interrupt will be generated if enable bit, RCIE, was set.
8. Read the RCSTA register to get the 9th bit (if enabled) and determine if any error occurred during reception.
9. Read the 8-bit received data by reading the RCREG register.
10. If any error occurred, clear the error by clearing enable bit CREN.
11. If using interrupts, ensure that the GIE and PEIE bits in the INTCON register (INTCON<7:6>) are set.

Interfacing Diagram:



Algorithm:

1.Calculate the SPBRGH: SPBRG value for desired baud rate. Load the calculated value in SPBRGH: SPBRG.
2. Configure Port pin RC6 as output and port pin RC7 as input.
3. Set BRG Mode = 8-bit by setting BRG16 bit in BAUDCON Register
4. Set EUSART mode = Asynchronous by clearing SYNC bit in TXSTA Register
5. Enable the Serial Port by setting SPEN bit in RCSTA Register
6. Enable the Transmission by setting TXEN bit in TXSTA Register
7. Check TRMT bit in TXSTA register for TXREG empty. Load data to the TXREG register if TRMT = 1;
8. Repeat the step 6 until complete message transmitted.
9. Enable the continuous reception by setting CREN bit in RCSTA register.
10.Poll the RCIF bit in PIR1 register to check data is received.
11.Read the 8-bit received data by reading the RCREG register.
12.Do the control action as per data received.
13.Repeat the steps from 10.

Conclusion:

Assignment No. 9: Study of Arduino, Raspberry-Pi

Aim: Study of Arduino, Raspberry-Pi, Beagle board.

Theory:

A) Arduino Uno Board

Arduino Uno is a microcontroller board developed by Arduino.cc which is an open-source electronics platform mainly based on AVR microcontroller Atmega328.
First Arduino project was started in Interaction Design Institute Ivrea in 2003 by David Cuartielles and Massimo Banzi with the intention of providing a cheap and flexible way to students and professional for controlling a number of devices in the real world.
It allows the designers to control and sense the external electronic devices in the real world.
There are many versions of Uno boards available, however, Arduino Nano V3 and Arduino Uno arethe most official versions. When nature and functionality of the task go complex, Mirco SD card can be added in the boards to make them store more information.

Processor:

1. 8-bit AVR microcontroller Atmega328
2. 2KB SRAM and 1KB of EEPROM.
3. 13KB of flash memory.

Features of Arduino Uno Board

1. Arduino Uno comes with USB interface i.e. USB port is added on the board to develop serial communication with the computer. Apart from USB, battery or AC toDC adopter can also be used to power the board. It is an open source platform where anyone can modify and optimize the board based on the number of instructions and task they want to achieve.
2. This board comes with a built-in regulation feature which keeps the voltage under control when the device is connected to the external device.
3. There are 14 I/O digital and 6 analog pins incorporated in the board that allows the external connection with any circuit with the board. These pins provide the flexibility and ease of use to the external devices that can be connected through these pins.
4. The 6 analog pins are marked as A0 to A5 and come with a resolution of 10bits. These pins measure from 0 to 5V, however, they can be configured to the high range using analog Reference() function and AREF pin.
5. 13KB of flash memory is used to store the number of instructions in the form of code.
6. Only 5 V is required to turn the board on, which can be achieved directly using USBport or external adopter, however, it can support external power source up to 12 V which can be regulated and limit to 5 V or 3.3 V based on the requirement of the project.

Pin Description

I/O digital and analog pins placed on the board which operates at 5V. These pins come with standard operating ratings ranging between 20mA to 40mA. Internal pull-up resistors are used in theboard that limits the current exceeding from the given operating conditions. However, too much increase in current makes these resisters useless and damages the device.

LED: Arduino Uno comes with built-in LED which is connected through pin 13. Providing HIGH value to the pin will turn it ON and LOW will turn it OFF.

Vin: It is the input voltage provided to the Arduino Board. Vin and Power Jack support a voltage ranges between 7V to 20V. This pin is used to supply voltage. If a voltage is provided through power jack, it can be accessed through this pin.

5V: This board comes with the ability to provide voltage regulation. 5V pin is used to provide output regulated voltage. The board is powered up using three ways i.e. USB, Vin pin of the board or DC power jack.

GND: These are ground pins. More than one ground pins are provided on the board which can be used as per requirement.

RESET: This pin is incorporated on the board which resets the program running on the board. Instead of physical reset on the board, IDE comes with a feature of resetting the board through programming.

IOREF: This pin is very useful for providing voltage reference to the board. A shield is used toread the voltage across this pin which then selects the proper power source.

PWM: PWM is provided by 3, 5, 6, 9, 10, 11 pins. These pins are configured to provide 8-bit output PWM.

SPI: It is known as Serial Peripheral Interface. Four pins 10(SS), 11(MOSI), 12(MISO), 13(SCK) provide SPI communication with the help of SPI library.

AREF: It is called Analog Reference. This pin is used for providing a reference voltage to the analog inputs.

TWI: It is called Two-wire Interface. TWI communication is accessed through Wire Library. A4 and A5 pins are used for this purpose.

Serial Communication: The Atmega328 placed on the board provides serial communication using pins like Rx and Tx. It is carried out through these pins called Pin 0 (Rx) and Pin 1 (Tx).

Rx pin is used to receive data while Tx pin is used to transmit data.

External Interrupts: Pin 2 and 3 are used for providing external interrupts. An interrupt is called by providing LOW or changing value.

(B) <u>Raspberry Pi Board</u>

Raspberry Pi is a credit card sized bargain micro-Linux machine. The goal behind creating Raspberry Pi was to create a low cost device that would improve programming skills and hardware understanding for students.Raspberry Pi is open hardware with the exception of its primary chip, theBroadcomm SoC which runs the main components of the board – CPU, graphics, memory, USB controller etc.

Processor:

1. The Model A+ is the low-cost variant of the Raspberry Pi. It has 256MB RAM, one USB port,40 GPIO pins and no Ethernet port.
2. The Model B+ is the final revision of the original Raspberry Pi. It has 512MB RAM, four USB ports, 40 GPIO pins, and an Ethernet port.
3. The Pi 2 uses a 900MHz quad-core ARM Cortex-A7 CPU and has 1GB RAM.
4. The Pi 3 Model B was launched in February 2016; it uses a 1.2GHz 64-bit quad-core ARMCortex-A53 CPU, has 1GB RAM, integrated 802.11n wireless LAN, and Bluetooth 4.1.
5. Pi Zero is half the size of a Model A+, with a 1Ghz single-core CPU and 512MB RAM, andmini-HDMI and USB On-The-Go ports.
6. Operating Systems: Raspbian RaspBMC, Arch Linux, Rise OS, OpenELEC Pidora
7. Video Output: HDMI Composite RCA
8. Supported Resolutions: 640x350 to 1920x1200, including 1080p, PAL & NTSC standards
9. Power Source: Micro USB

Hardware required to setup your Raspberry Pi

1. A Raspberry Pi
2. An HDMI or composite video capable television or monitor
3. An HDMI or composite video cable
4. An SD card that is compatible with your Raspberry Pi - http://elinux.org/Raspberry Pi_SD_cards has a list of SD cards you should use
5. A USB keyboard and mouse (Bluetooth keyboard/mouse work for latest model but with minor connectivity issues)
6. Standard Ethernet cable
7. Micro USB power supply (that can provide at least 700mA at 5V)
8. A 3.5 mm stereo audio cable if your project requires Raspberry Pi to be connected to external speakers.

Steps to setup Raspberry Pi

Each of the following steps is detailed in the subsequent slides.

Step 1 : SD Card Setup

Step 2 : Raspberry Pi cabling

Step 3 : Booting your Raspberry Pi for the first time

Step 4 : Load GUI environment to your Raspberry Pi

Sep 5 : Setup a network connection

Step 6: After completing this setup, you will have your device powered up and working as full-fledged Linux box running Debian.

Raspberry Pi cabling

1. Push SD card into the SD card slot.
2. Plug the HDMI cable into the HDMI output of the Raspberry Pi and connect to the

TV/monitor.

3. Turn on monitor and switch to the HDMI port.
4. Insert the network cable and connect to the router.
5. Connect the keyboard and mouse via USB ports.
6. Plug the power supply into the micro USB.
7. The device is now ready for the next steps.

Voltages

Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V),which are not configurable.

Outputs

A GPIO pin designated as an output pin can be set to high (3V3) or low (0V).

Inputs

A GPIO pin designated as an input pin can be read as high (3V3) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software. The GPIO pins can be used with a variety of alternative functions; some are available on all pins, others on specific pins such as

1. PWM (pulse-width modulation)
2. Software PWM available on all pins
3. Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
4. SPI
5. SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
6. SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17);CE2 (GPIO16)
7. I2C
8. Data: (GPIO2); Clock (GPIO3)
9. EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
10. Serial
11. TX (GPIO14); RX (GPIO15)

Difference between Arduino and Raspberry Pi

Sr. No.	Raspberry Pi	Arduino
1	It is a mini computer with Raspbian OS.It can run multiple programs at a time.	Arduino is a microcontroller, which is a part of the computer. It runs only one program again and again.
2	It is difficult to power using a batterypack.	Arduino can be powered using a battery pack.
3	It requires complex tasks like installing libraries and software for interfacing sensors and other components	It is very simple to interface sensors and other electronic components to Arduino.

4	It is expensive	It is available for low cost.
5	Raspberry Pi can be easily connected to the internet using Ethernet port and USB Wi-Fi dongles.	Arduino requires external hardware to connect to the internet and this hardware is addressed properly using code.
6	Raspberry Pi did not have storage on board. It provides an SD card port.	Arduino can provide onboard storage.
7	Raspberry Pi has 4 USB ports to connect different devices.	Arduino has only one USB port to connectto the computer.
8	The processor used is from ARM family.	Processor used in Arduino is from AVR family Atmega328P
9	This should be properly shutdown otherwise there is a risk of files corruption and software problems.	This is a just plug and play device. If power is connected it starts running the program and if disconnected it simply stops.
10	The Recommended programming language is python but C, C++, Python,ruby are pre-installed.	Arduino uses Arduino, C/C++

(C)<u>Beagle Bone Board</u>

The Beagle Board is a low-power open-source single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. Beagle Bone Black is a low-cost, open source, community-supported development platform for ARM CortexTM-A8 processor developers and hobbyists.

Processor:

1. AM335x 1GHz ARM® Cortex-A8
2. 512MB DDR3 RAM
3. 4GB 8-bit eMMC on-board flash storage
4. 3D graphics accelerator
5. NEON floating-point accelerator
6. 2x PRU (PRU-Programmable Real Time Unit) 32-bit microcontrollers

Connectivity

1. USB client for power & communications
2. USB host
3. Ethernet
4. HDMI
5. 2x 46 pin headers

Software Compatibility

1. Debian
2. Android
3. Ubuntu
4. Cloud9 IDE on Node.js w/ BoneScript library
5. plus, much more

Difference between Raspberry Pi and BeagleBone Black

Sr. No.	**BeagleBone Black**	**Raspberry PI**
1	Better interfaces with external sensors	Better graphics capabilities
2	Faster, newer, better supported processor	Better audio capabilities
3	Includes an OS out of the box	Better community support
4	Runs Angstrom, Ubuntu and other OS’s	Mainly runs Raspbian OS

Applications

The devices discussed above come in wide range of applications. A larger number of people are using these boards for developing sensors and instruments that are used in scientific research. Following are some main applications of the board.

1. Embedded System
2. Security and Defense System
3. Digital Electronics and Robotics
4. Parking Lot Counter
5. Weighing Machines
6. Traffic Light Count Down Timer
7. Medical Instrument
8. Emergency Light for Railways
9. Home Automation
10. Industrial Automatio

Experiment No.10

Aim:- Write simple program using Open source prototype platform like Arduino for digital read / Write using LED .

Apparatus: Bread Board, Arduino Board ,LED, Connecting Wires, USB cable, Power Supply Connector

Theory:-

Build the circuit

Here is the circuit.

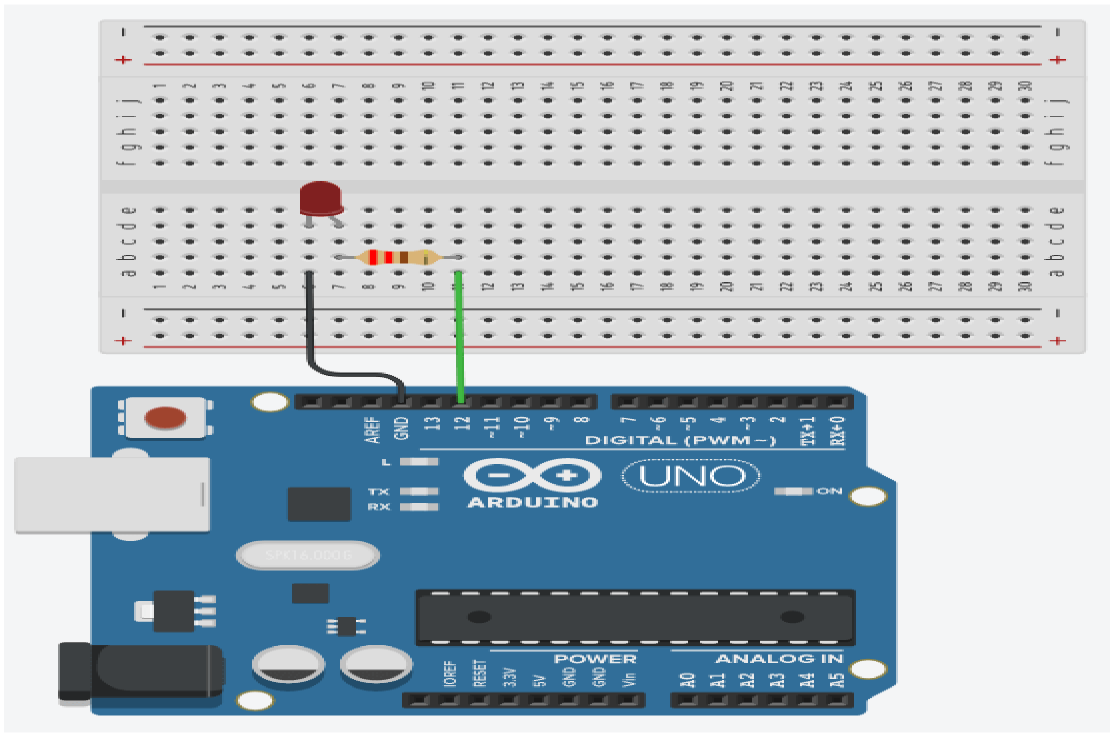


Fig1: Hardware Connection

1) First make sure that the Arduino is powered off (no USB cable plugged to anything Check the LED, you will see that one of the leg is shorter than the other one.
2) Plug the shorter leg of the LED to a hole on the breadboard. Connect that leg to a GND pin of the Arduino, using a black cable if possible (convention for GND).
3) Plug the longer leg of the LED to a different hole, on a different and independent line of the breadboard.
4) Add a 220 Ohm resistor between this longer leg and a digital pin of the Arduino, using an additional colored wire (no red, no black) for convenience. Here we choose digital pin 12 on the Arduino Uno.

Note:

you could have chosen any of the digital pins ranging from 0-13, and also any of the analog pins, which you can use as digital pins. So, for the Arduino Uno, you get 20 possibilities.

Why a 220 Ohm resistor for the LED:

Basically, you want to limit the current that goes through the LED, so you will avoid damaging it and also damaging the pin on the Arduino board. The max value can depend, but let's agree on 20mA max, which is 0.02A.
With the Ohm's law, you get the relation between resistance, voltage, and current:
Voltage = Resistance * Current.
If you modify the order you get: Resistance = Voltage / Current.
So, if you want a max current of 0.02A, for a 5V voltage, you would need a 5V/0.02A = 250 Ohm resistor.
The thing is that, for LEDs, there is something more to take into account. This is the voltage drop of the LED. This voltage drop is going to be different for different colors, but let's approximate it to 2V. So, the computation for the resistor becomes (5V – 2V)/0.02A = 150 Ohm.
With those values, you'd need a resistor which is at least 150 Ohm, so you can make sure the current stays under 20mA. The thing is that, finding a 150 Ohm resistor is not that common. 220 Ohm is more common, and it's a higher value, so no problem. In fact, you could also use a 330 Ohm resistor and it will work, just with a slightly lower brightness, because the current will be lower. You could go as high as 1kOhm, for example if you want to add many LEDs, in order to reduce the overall current usage.
One thing to understand here: there's no point in getting an exact resistor value. Any approximation that is above the minimum "safety" computation is fine.

Arduino code to power on an LED

Power on the LED with digitalWrite()
Let's start with the most simple thing. Let's say you just want to power on the LED when the Arduino program starts. This code will help us understand the setup and how digital pins work.

```
#define LED_PIN 12
void setup()
{
pinMode(LED_PIN, OUTPUT);
digitalWrite(LED_PIN, HIGH);
}
void loop() {}
```

This code is very short and will just power on the LED on pin 12. Let's analyze it.

#define LED_PIN 12
First, and this is a best practice, we create a define for the pin number. This way, anytime we need to use

this digital pin, we can write LED_PIN instead of the hard-coded number. In the future, if you ever need to use a different digital pin (for example 11), then you just need to change the number here and it will update it everywhere in your program.

```
void setup()
{
pinMode(LED_PIN, OUTPUT);
```

We enter the void setup() function, which will be executed once at the beginning of the program. Before we can actually use a digital pin, we need to set a mode. Basically you have 2 modes: output (if you want to control a component), or input (if you want to read some information from a component). Here, we want to control the LED, so we choose output. For that, we use the pinMode() function with 2 parameters:

- Pin number: here we use the LED_PIN that we previously defined.
- Mode: for output mode, we have to use OUTPUT (all uppercase).

After the execution of this line, the digital pin 12 will be set as output, and we can control the LED.

```
digitalWrite(LED_PIN, HIGH);
}
```

Now, to control the LED, it's very simple. We need to use the digitalWrite() function with 2 parameters:

- Pin number: again, we use the defined LED_PIN.
- State: we have 2 choices here. HIGH to power on the LED, and LOW to power it off.

So, after this line, the LED will be powered on.

Note: it's important to call pinMode() before digitalWrite(), otherwise the LED won't be powered on because the mode is not set.

```
void loop() {}
```

After the void setup() function is finished, we enter the void loop() and this function will be executed again and again, until you power off the Arduino.

Here we didn't write anything in the void loop() because we just want to power on the LED and that's it.

Output: We observe the status of LED

Conclusion: Program is executed in Arduino IDE and output is observed on LED.