

Embedded systems

with PIC Microcontroller

Sidharth Mehta

# Table of Contents

1. INTRODUCTION TO EMBEDDED SYSTEM
   1. Brief History
      1. Different Microcontroller Architectures
   2. Difference Between Processors and Controllers
   3. Applications of Embedded System
2. INTRODUCTION TO MICROCONTROLLER
   1. Classification of Microcontrollers
      1. Classification on the basis of bits
      2. Market Scenario
   2. Introduction to PIC High Range, Mid-Range & Low Range Controllers
   3. Basic Architecture Description of PIC16F887
      1. Pin Diagram
      2. Memory Organization
   4. Introduction to General Microcontroller Terms
      1. Program Counter(PC)
      2. Accumulator (or Working Register)
      3. Reset
      4. Clock Cycle, Machine Cycle, Instruction Cycle
      5. Interrupts
      6. Stack
   5. Brief Introduction to Internal Features
      1. General Purpose Input-Output PORTs
      2. Interrupt
      3. Timers
      4. Analog to Digital Convertors
      5. USART
      6. EEPROM
      7. Device Protection features - Watchdog Timer, BOR, Power up Timer
3. DESCRIPTION OF DEVELOPMENT TOOLS
   1. MPLAB IDE
      1. Demo of Project Making & sample programs
   2. Proteus - ISIS
      1. Brief introduction to simulator tool
      2. Demo of some small logic circuits
   3. Programming tools
      1. PICkit2
4. EMBEDDED C PROGRAMMING
   1. LEDs
      1. LED Introduction
      2. LED Interfacing with Microcontroller
      3. LED programming
   2. Linear Switches
      1. Introduction to Switches
      2. Interfacing of Switches with Microcontroller
      3. Switches Programming
   3. Seven Segment Display(SSD)
      1. Introduction to SSD
      2. Types of SSD (CC & CA)
      3. Logics of displaying Numeric values on SSD
      4. Interfacing of SSD with Microcontroller
      5. SSD Programming
   4. Keypad Matrix
      1. Concept of Keypad Matrix
      2. Interfacing of Keypad with Microcontroller
      3. Programming of Keypad Matrix
   5. Multi Seven Segment Display(MSSD)
      1. Concept of MSSD
      2. Interfacing of MSSD with Microcontroller
      3. Programming of MSSD
   6. Liquid Crystal Display(LCD)
      1. Types of LCD (Character & Graphical)
      2. Introduction to LCD Driver IC(HD44780) & Character LCD Module
      3. Introduction to Pin Diagram of Character LCD
      4. Interfacing of Character LCD with Microcontroller
      5. Programming of Character LCD
   7. Timers / Counters
      1. Timer0
      2. Timer1
      3. Programming concept of Timers to Generate delays
5. ADVANCED INTERNAL PERIPHERALS
   1. Analog to Digital Converter(ADC)
      1. Types of ADC
      2. Introduction to PIC internal ADC and its SFRs
      3. Programming Concept of ADC
   2. Universal Asynchronous Receiver Transmitter(UART)
      1. UART and its SFRs
      2. Programming concept of Serial Transmitter & Receiver using UART
   3. EEPROM
      1. EEPROM SFRs
      2. Programming concept of EEPROM Writing & Reading
   4. Interrupts
      1. Polling Vs. Interrupt
      2. Types of Interrupts
      3. Interrupts in PIC16F887
      4. Programming Concept of Interrupts

# 1. INTRODUCTION TO EMBEDDED SYSTEM

* Brief History
* Difference Between Processors and Controllers
* Applications of Embedded System

Contents

## 1.1 Brief History

An **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system.

One of the very first recognizably modern embedded systems was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project as it employed the then newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits.

Since these early applications in the 1960s, embedded systems have come down in price and there has been a dramatic rise in processing power and functionality. An early microprocessor for example, the Intel 4004, was designed for calculators and other small systems but still required external memory and support chips. In 1978 National Engineering Manufacturers Association released a "standard" for programmable microcontrollers, including almost any computer-based controllers, such as single board computers, numerical, and event-based controllers.

### 1.1.1 Different Microcontroller Architectures

Architecture is a conceptual design and operational structure of a system. There are two main architectures considered in microcontroller designs. Harvard and von Neumann architectures. The difference is the way data and programs are accessed and stored.

Harvard Architecture: In this architecture, physically separate memories with their own dedicated buses are used for instructions and data. This allows faster execution and data flow happens in parallel with each other.

Von Neumann: Contrast to Harvard Architecture, in this the controller can either read an instruction or read/write data from/to the memory. Both cannot happen at the same time as they use the same bus for instruction and data transfer.

## 1.2 Difference Between Processors and Controllers

|  |  |
| --- | --- |
| Microprocessor | Microcontroller |
| * Microprocessor is an IC which has only the CPU inside them i.e. only the processing powers. | * Microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip. |
| * Microprocessor find applications where tasks are unspecific like developing software, games, websites, photo editing, creating documents etc. In such cases the relationship between input and output is not defined. | * Microcontrollers are designed to perform specific tasks. Specific means applications where the relationship of input and output is defined. Depending on the input, some processing needs to be done and output is delivered. |
| * They need high amount of resources like RAM, ROM, I/O ports etc. Microprocessor cannot be used stand alone. They need other peripherals like RAM, ROM, buffer, I/O ports etc. and hence a system designed around a microprocessor is quite costly. | * Since the applications are very specific, they need small resources like RAM, ROM, I/O ports etc. and hence can be embedded on a single chip. This in turn reduces the size and the cost. |
| * E.g. Intel’s Pentium, Core i5, AMD Phenom etc. | * E.g. Atmel Atmega, Microchip PIC etc. |

## 1.3 Applications of Embedded System

Embedded systems are commonly found in consumer, cooking, industrial, automotive, medical, commercial and military applications.

Telecommunications systems employ numerous embedded systems from telephone switches for the network to cell phones at the end user. Computer networking uses dedicated routers and network bridges to route data.

Consumer electronics include MP3 players, mobile phones, videogame consoles, digital cameras, GPS receivers, and printers. Household appliances, such as microwave ovens, washing machines and dishwashers, include embedded systems to provide flexibility, efficiency and features. Advanced HVAC systems use networked thermostats to more accurately and efficiently control temperature that can change by time of day and season. Home automation uses wired- and wireless-networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.

Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Various electric motors — brushless DC motors, induction motors and DC motors — use electric/electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles increasingly use embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems include anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.

Medical equipment uses embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging (PET, SPECT, CT, and MRI) for non-invasive internal inspections. Embedded systems within medical equipment are often powered by industrial computers.

Embedded systems are used in transportation, fire safety, safety and security, medical applications and life critical systems, as these systems can be isolated from hacking and thus, be more reliable. For fire safety, the systems can be designed to have greater ability to handle higher temperatures and continue to operate. In dealing with security, the embedded systems can be self-sufficient and be able to deal with cut electrical and communication systems.

Embedded Wi-Fi modules provide a simple means of wirelessly enabling any device which communicates via a serial port.

# 2. INTRODUCTION TO MICROCONTROLLER

* Classification of Microcontrollers
* Introduction to PIC High Range, Mid-Range & Low Range Controllers
* Basic Architecture Description of PIC16F887
* Introduction to General Microcontroller Terms
* Brief Introduction to Internal Features

Contents

## **2.1** Classification of Microcontrollers

Microcontrollers can be classified on the basis of bits, memory, architecture and instruction set.

### 2.1.1 Classification on basis of bits:

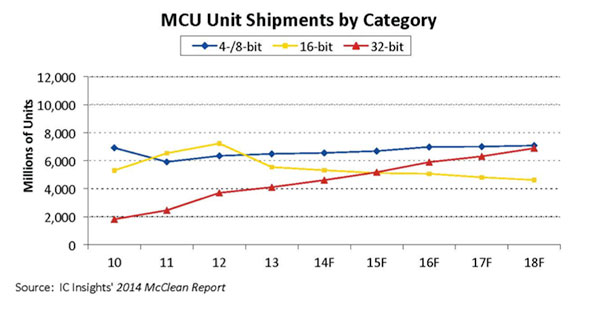
On the basis of bits, the microcontrollers can be classified as:

1. 4 bit
2. 8 bit
3. 16 bit
4. and 32 bit microcontrollers.

Here the no of bits refers to the size of data bus.

### 2.1.2 Market scenario

Here is a self-explanatory bar graph that shows the market cap of each microcontroller classified on basis of bits.



Source IC Insights, McCLean reports

Fig 2.1 MCU Shipments by category

## 2.2 Introduction to PIC High Range, Mid-Range & Low Range Controllers

PIC microchips are designed with a Harvard architecture, and are offered in various device families. The baseline and mid-range families use 8-bit wide data memory, and the high-end families use 16-bit data memory. The latest series, PIC32MZ is a 32-bit MIPS-based microcontroller. Instruction words are in sizes of 12-bit (PIC10 and PIC12), 14-bit (PIC16) and 24-bit (PIC24 and dsPIC).

The contents of this text cover the PIC16F887 controller of the mid-range family.

## 2.3 Basic Architecture Description of PIC16F887

### 2.3.1 Pin Diagram

The PIC16F887 controller is available in 40 pin PDIP, 44 pin QFN and TQFP packages.

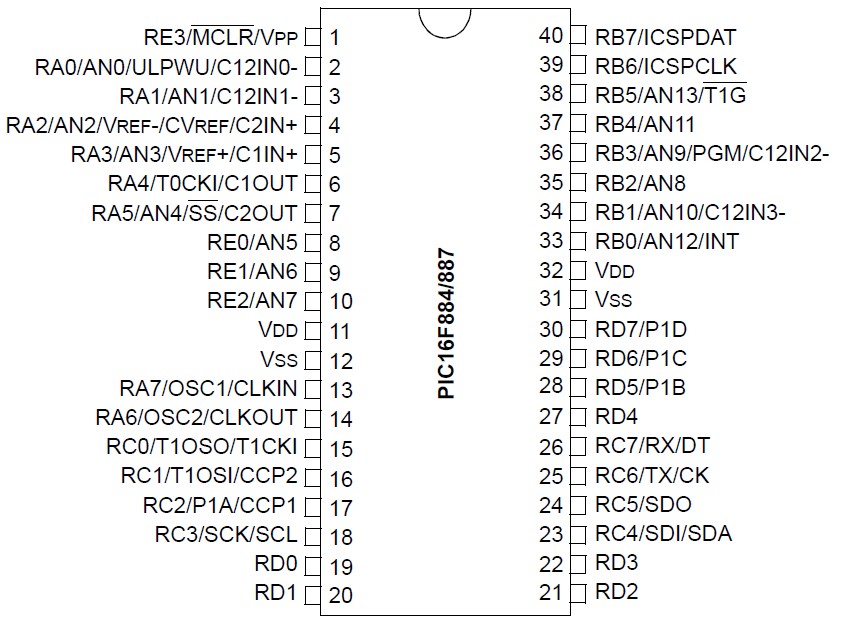


Fig 2.2 PIC16F887 in PDIP package

### 2.3.2 Memory organization

The controller has an 8k\*14 program memory space. The data memory is partitioned into 4 banks which contain General purpose registers (GPR) and Special function registers (SFR).

A register is a set of D flip flops each having size of 4/8/16/32 bit depending upon the type of controller which is meant for temporary storage of data.

The special function registers are used by the CPU and peripheral functions for controlling the desired operation of the device. The special function registers can be classified as core and peripheral. The SFRs will be used later on in the text.

## 2.4 Introduction to General Microcontroller Terms

### 2.4.1 Program Counter(PC)

It is a processor register that indicates where a computer is in its program sequence/ the address of the next instruction to be executed. The size of PC is 13 bit in the controller and in general the size of PC depends upon the size of address bus.

### 2.4.2 Accumulator (or Working Register)

It is an intermediate register between the CPU and other registers. Its size is of 8 bits; this is so because of the size of data bus being 8 bits. It is used to perform arithmetic as well as logical operations on data. The Accumulator is called as the working register (WREG) in PIC architecture.

### 2.4.3 Reset

A reset signal brings the microcontroller to its initial state. A Power On Reset (POR) is provided on applying power to the controller. If this signal is not provided the microcontroller will start executing opcodes from any random location

### 2.4.4 Clock Cycle, Machine Cycle, Instruction Cycle

A **clock signal** is a particular type of signal that oscillates between a high and a low state. A clock signal is produced by a clock generator. Although more complex arrangements are used, the most common clock signal is in the form of a square wave with a 50% duty cycle, usually with a fixed, constant frequency. The four steps of Machine cycle are:

1. **Fetch** - Retrieve an instruction from the memory.
2. **Decode** - Translate the retrieved instruction into a series of computer commands.
3. **Execute** - Execute the computer commands.
4. **Store** - Send and write the results back in memory.

A **machine cycle** is a set of steps performed by the computer processor for each machine language instruction received. The machine cycle is a 4 process cycle that includes reading and interpreting the machine language, executing the code and then storing that code.

An **instruction cycle** (sometimes called a **fetch–decode–execute** cycle) is the basic operational process of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction dictates, and carries out those actions. This cycle is repeated continuously by a computer's central processing unit (CPU), from boot-up to when the computer is shut down.

### 2.4.5 Interrupts

An **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt service routine, ISR to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities. There are two types of interrupts: hardware interrupts and software interrupts.

### 2.4.6 Stack

A part of the RAM is used for the stack. Before the microcontroller starts to execute a subroutine, or when an interrupt occurs, the address of first next instruction being currently executed is pushed onto the stack.  In that way, upon subroutine or interrupt execution, the microcontroller knows from where to continue regular program execution.

## 2.5 Brief Introduction to Internal Features

### 2.5.1 General Purpose Input-Output PORTs

**General-purpose input/output (GPIO)** is a generic pin on an integrated circuit or computer board whose behavior—including whether it is an input or output pin—is controllable by the user at run time.

There are 5 GPIO ports on the PIC16F887 controller with PORT A, B, C and D having 8 pins each and PORT E having 4 pins.

### 2.5.2 Timers

The controller has 3 timers, namely Timer 0, Timer 1 and Timer 2. The Timer 0 and Timer 2 are 8 bit timers whereas the Timer 1 is 16 bit in size.

### 2.5.3 Analog to Digital Convertors

An analog to digital convertor (ADC) is used to convert analog data into digital form. The PIC16F887 have 14 analog pins which are multiplexed to a 10-bit ADC.

### 2.5.4 USART

USART stands for universal synchronous/asynchronous receiver transmitter. It is a hardware peripheral used to provide a means for serial communication.

### 2.5.5 EEPROM

EEPROM stands for electrically erasable programmable read-only memory and is a type of non-volatile memory used in computers and other electronic devices to store relatively small amounts of data but allowing individual bytes to be erased and reprogrammed. Its size is 256 bytes on the PIC16F887.

### 2.5.6 Device Protection features - Watchdog Timer, BOR, Power-up Timer

A **watchdog timer** is an electronic timer that is used to detect and recover from computer malfunctions. During normal operation, the computer regularly resets the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to reset the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective action or actions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation.

A **brownout** is an intentional or unintentional drop in voltage in an electrical power supply system. A brownout reset resets the controller when a brownout occurs.

The **Power-up** timer provides a delay of 64ms on power up, from a Power-on reset or Brownout reset.

# 3. DESCRIPTION TO DEVELOPMENT TOOLS

* MPLAB IDE
* Proteus – ISIS
* Programming tools

Contents

## 3.1 MPLAB IDE

MPLAB is a proprietary freeware integrated development environment for the development of embedded applications on PIC and dsPIC microcontrollers, and is developed by Microchip Technology.

MPLAB X is the latest edition of MPLAB, and is developed on the NetBeans platform. MPLAB and MPLAB X support project management, code editing, debugging and programming of Microchip 8-bit, 16-bit, and 32-bit PIC microcontrollers.

MPLAB is designed to work with MPLAB-certified devices such as the MPLAB ICD 3 and MPLAB REAL ICE, for programming and debugging PIC microcontrollers using a personal computer. PICKit programmers are also supported by MPLAB.

MPLAB 8.X is the last version of the legacy MPLAB IDE technology, custom built by Microchip Technology in Microsoft Visual C++.

MPLAB supports project management, editing, debugging and programming of Microchip 8-bit, 16-bit and 32-bit PIC microcontrollers. MPLAB is still available from Microchip's archives, but is not recommended for new projects.

MPLAB supports the following compilers:

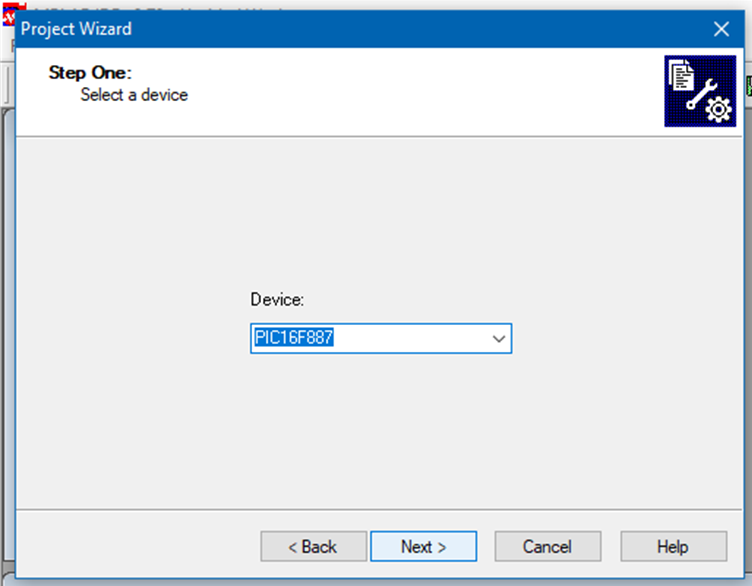
* MPLAB MPASM Assembler
* MPLAB ASM30 Assembler
* MPLAB C Compiler for PIC18
* MPLAB C Compiler for PIC24 and dsPIC DSCs
* MPLAB C Compiler for PIC32
* HI-TECH C

### 3.1.1 Demo of Project Making & sample programs

**Step 1:** Open MPLAB IDE and click on Project>Project Wizard.



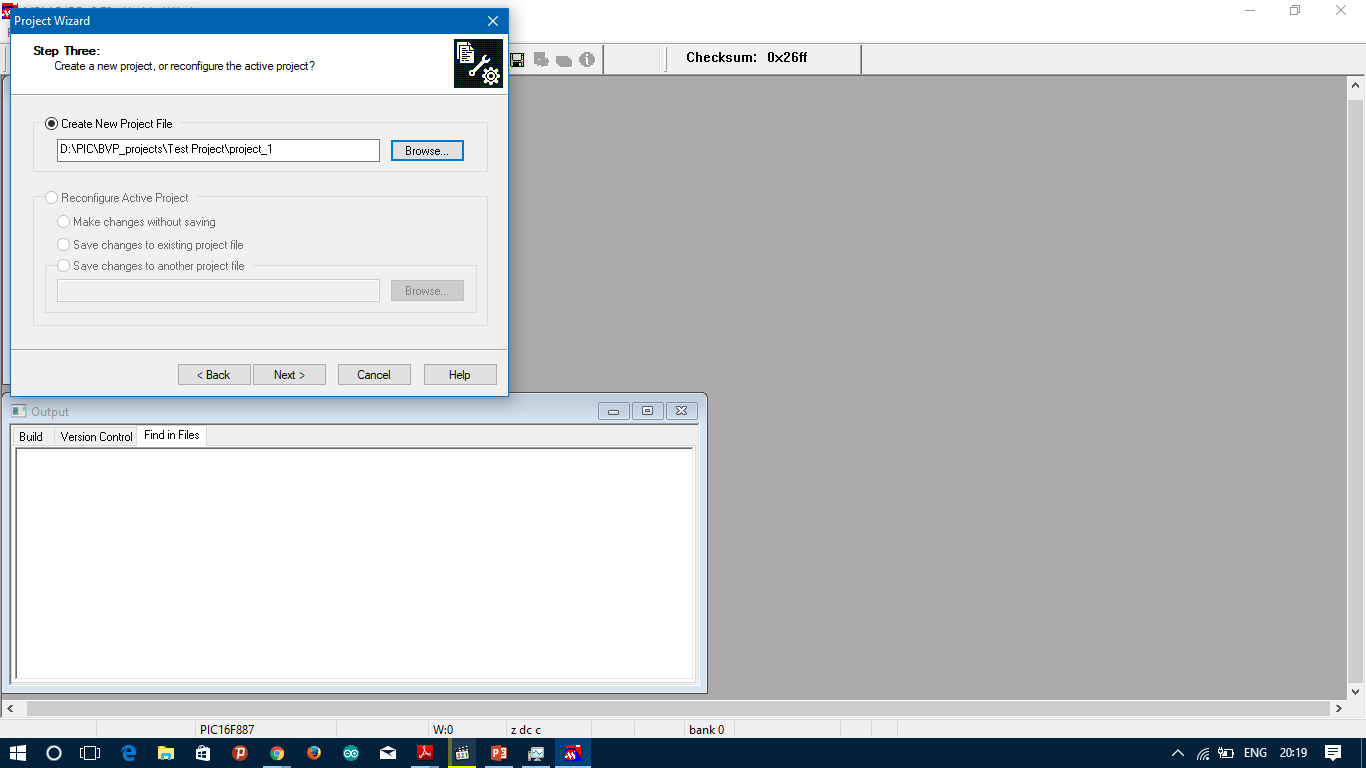
**Step 2:** Click on Next.



**Step 3:** Choose PIC16F887 from the dropdown list and click on Next.

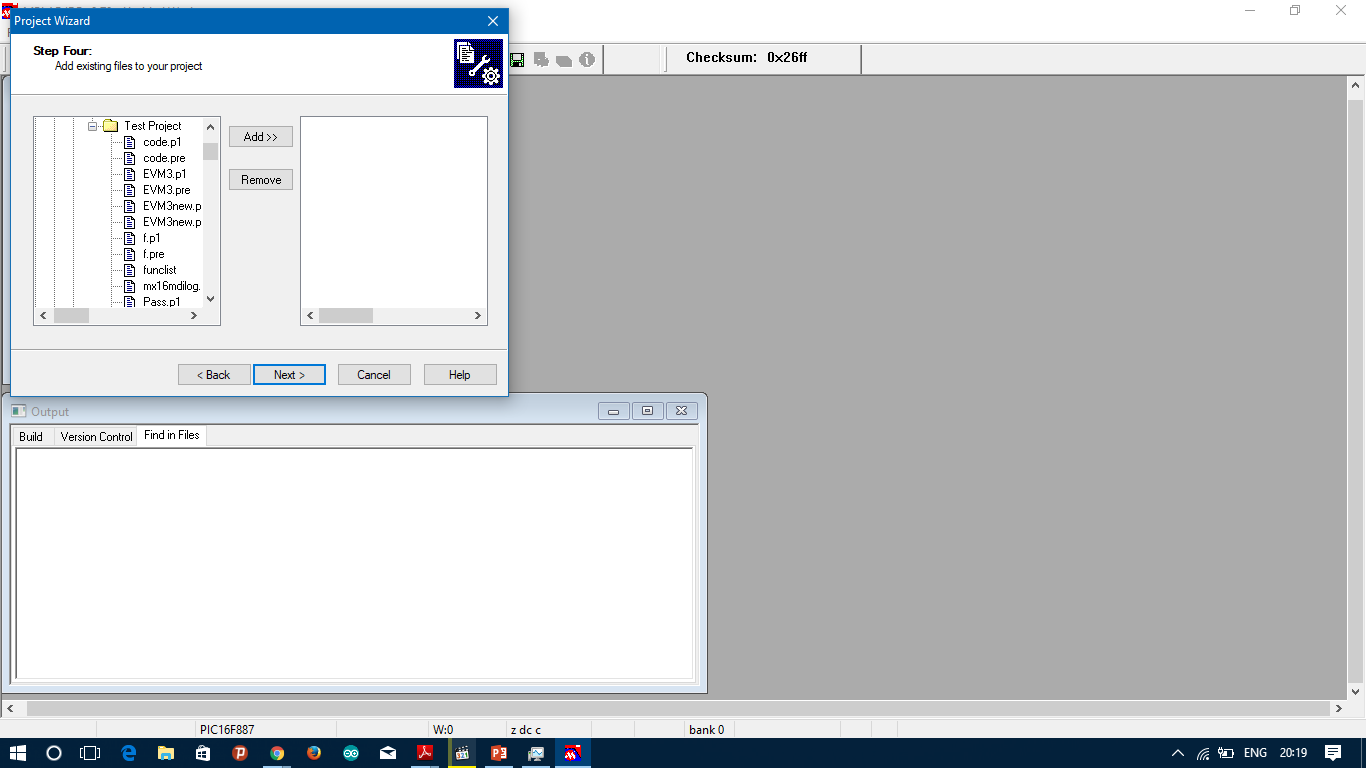


**Step 4:** Choose HiTech C Compiler and click on next.

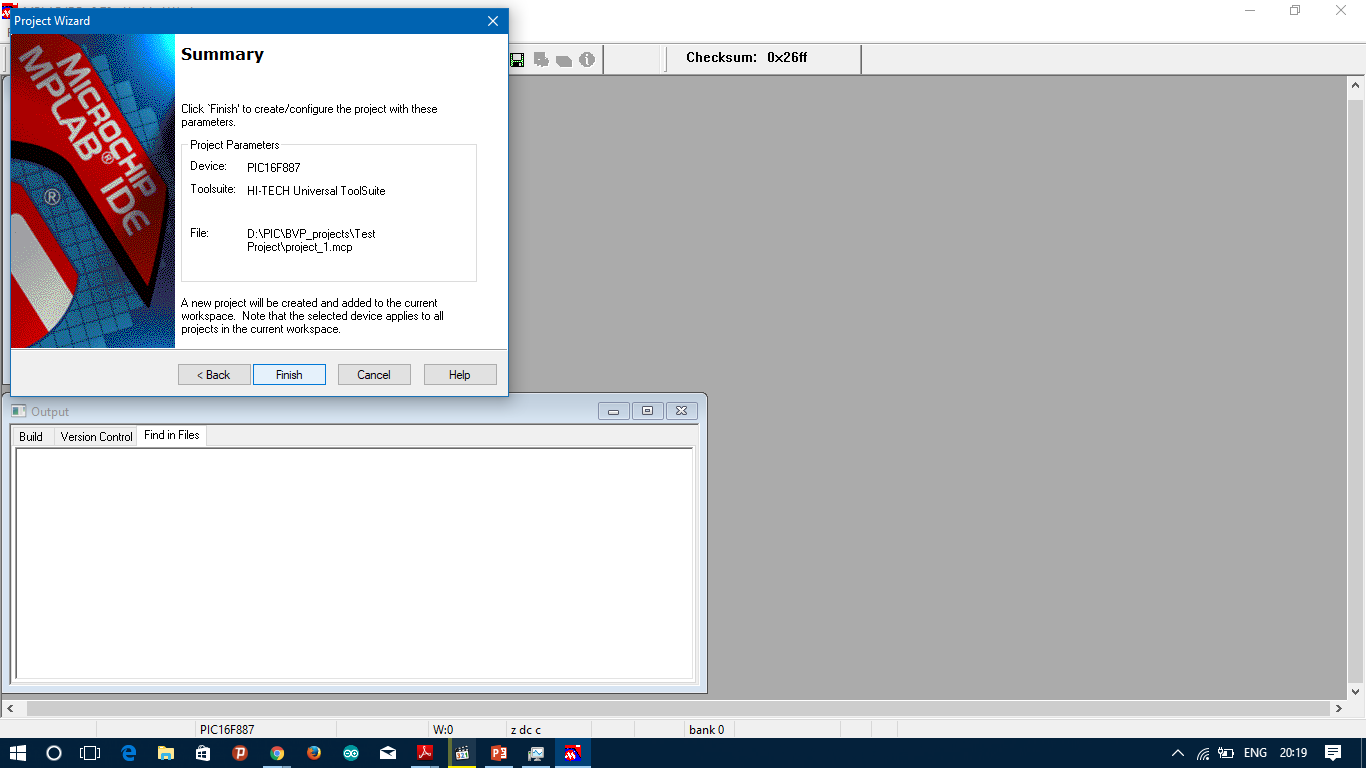


**Step 5:** Browse to the location where you want to save the project and give it a name.

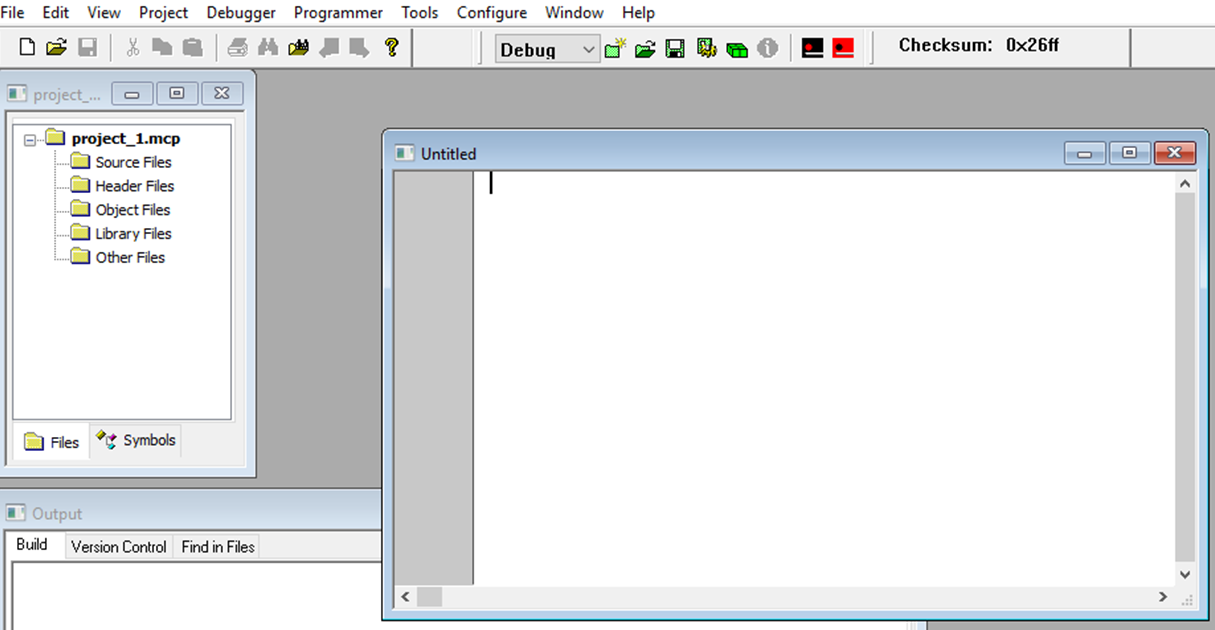
**Step 6:** Click on Next.



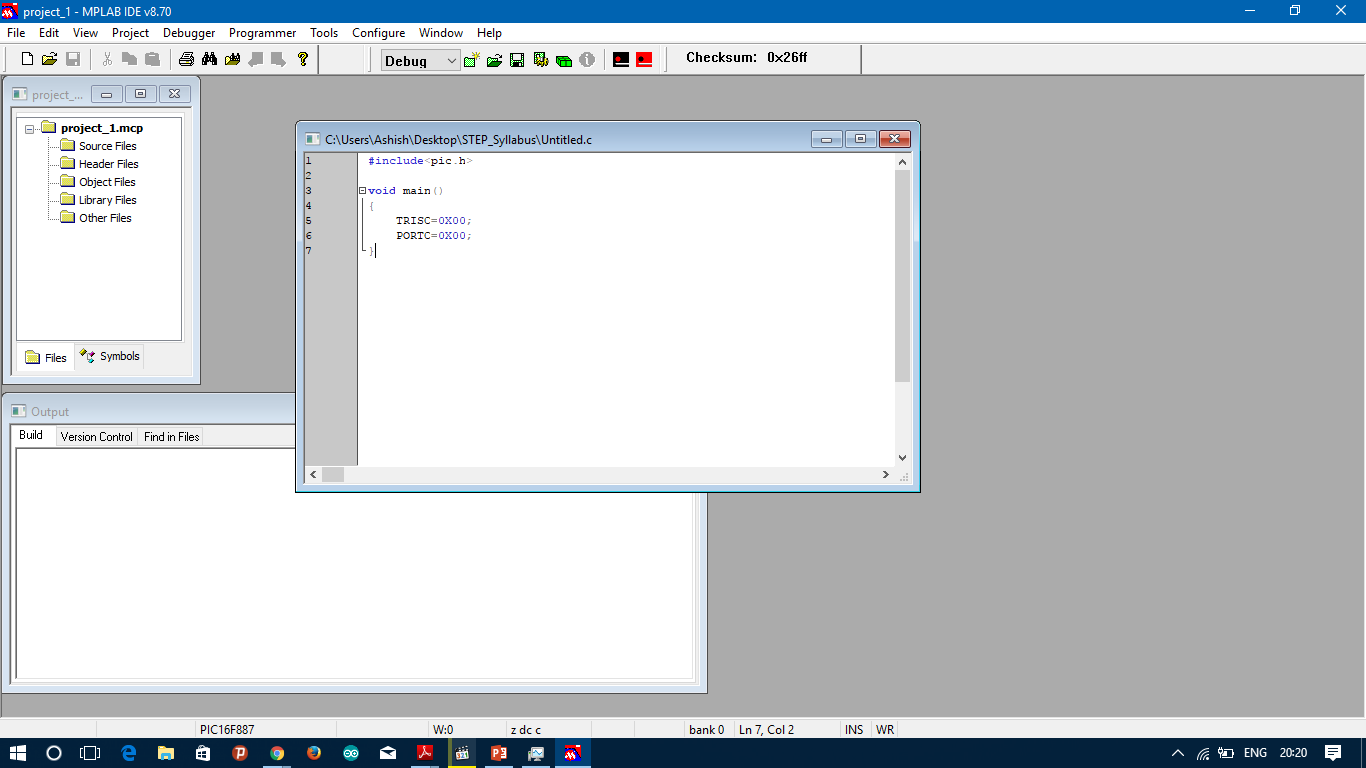
**Step 7:** Choose if you want any prewritten file to be included or else click on next.



**Step 8:** Click on Finish.

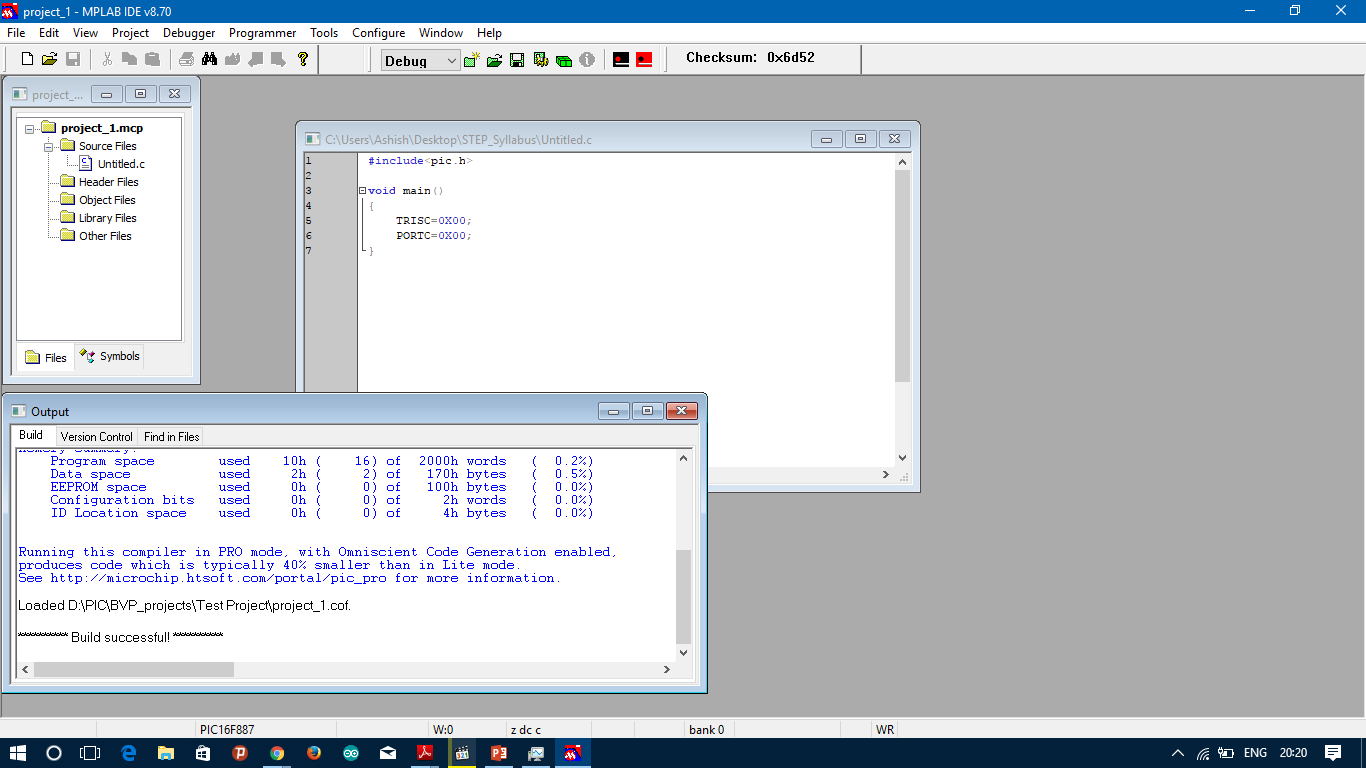
****

**Step 9:** Click on File>New and a blank document will open.



**Step 10:** Type the code in the document and save it with “.c” extension.

**Step 11:** Right click on the File and select **Add to project**. If the project was created correctly, the file should be added in Source Files.

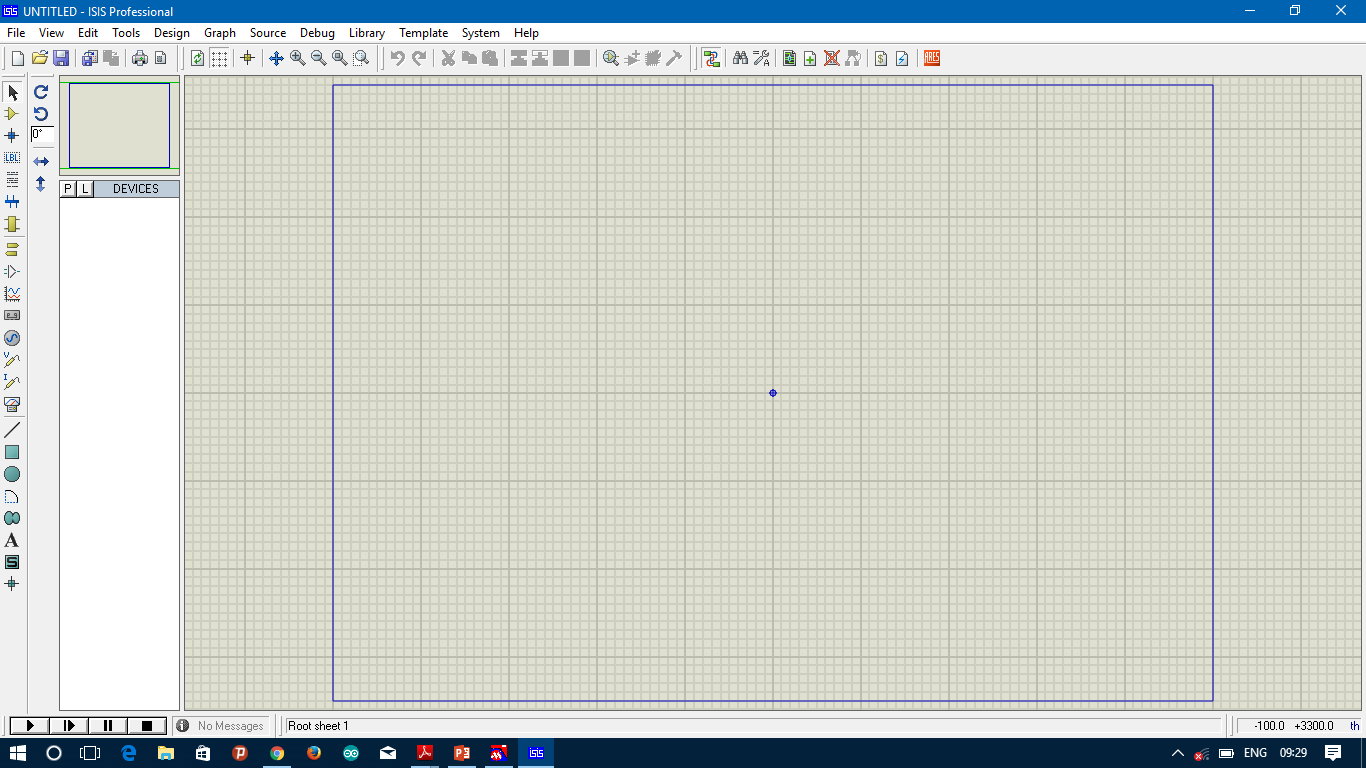


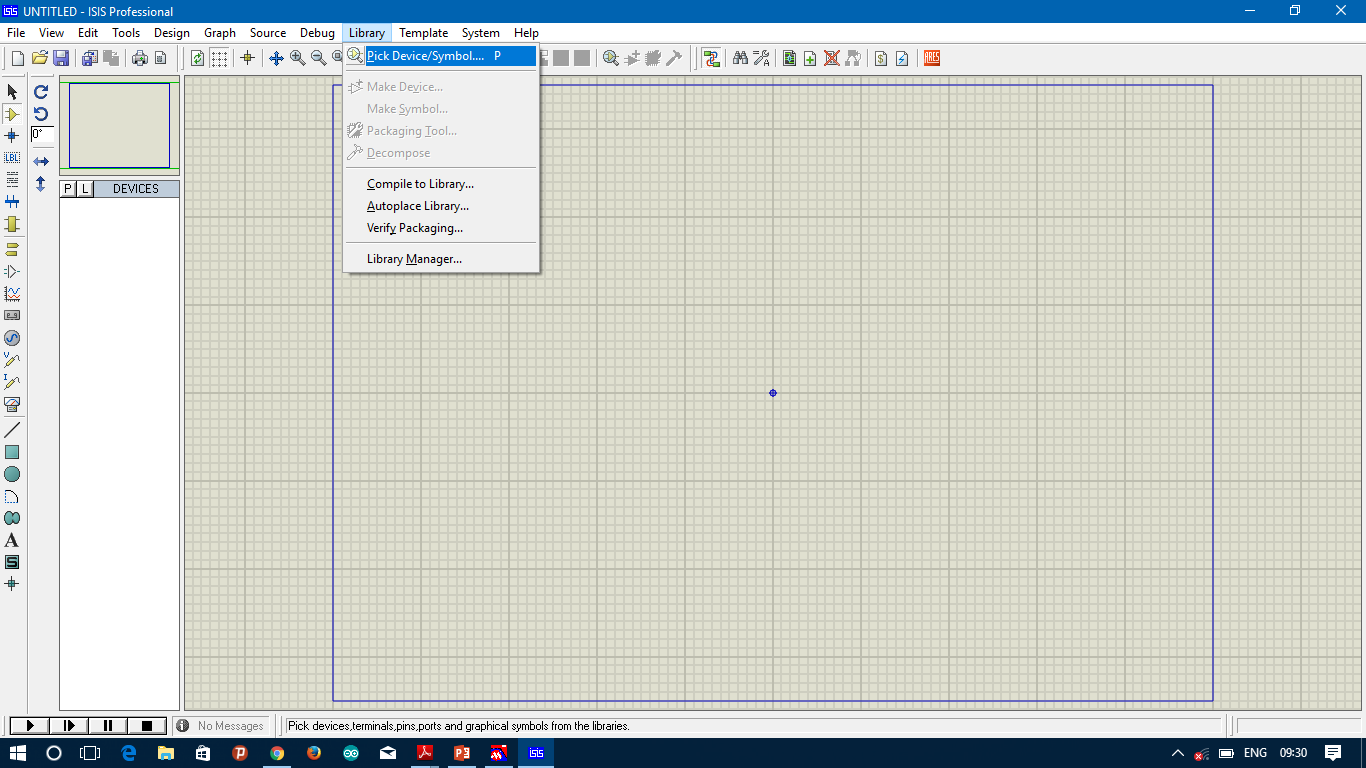
**Step 12:** Build the project. If output shows Build successful, you are done.

## 3.2 PROTEUS – ISIS

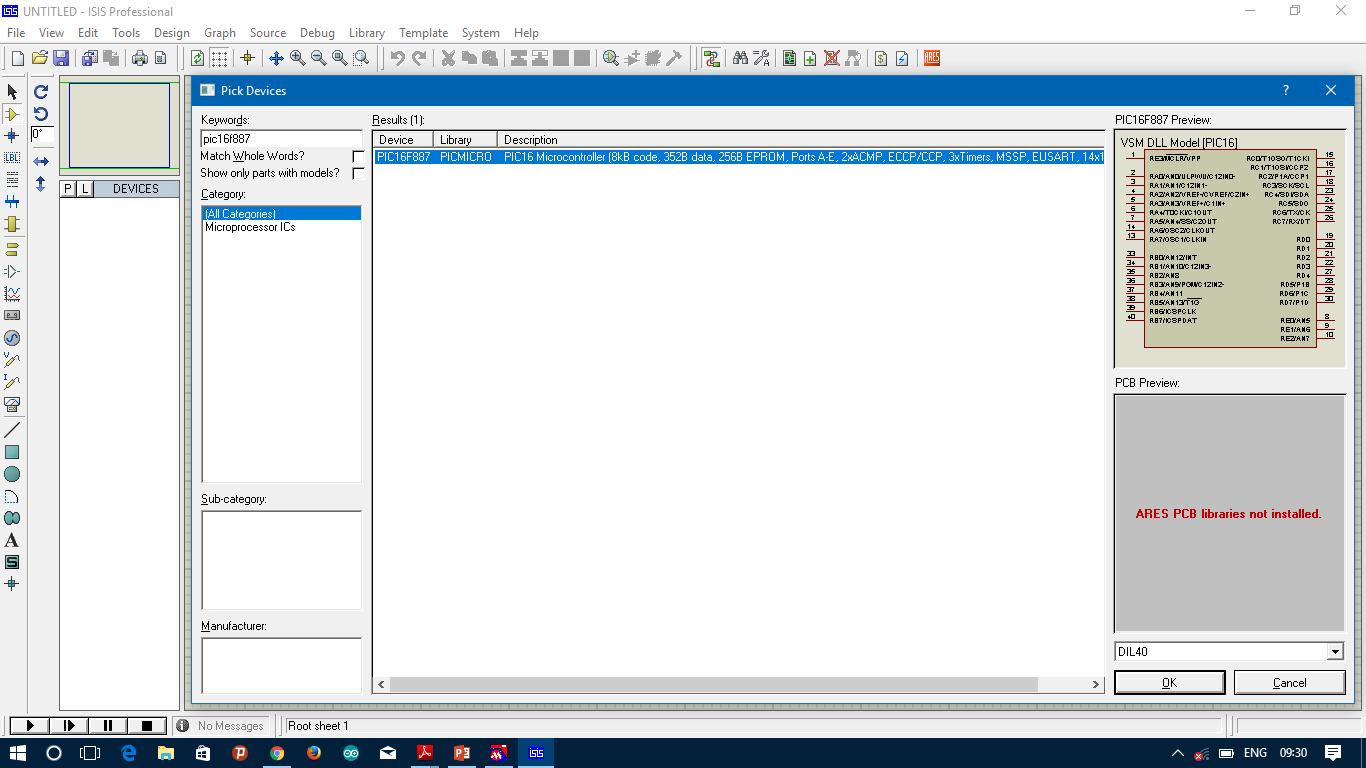
The Proteus Design Suite is a proprietary software tool suite used primarily for electronic design automation. The software is used mainly by electronic design engineers and electronic technicians to create electronic schematics and electronic prints for manufacturing printed circuit boards.

The micro-controller simulation in Proteus works by applying either a hex file or a debug file to the microcontroller part on the schematic. It is then co-simulated along with any analog and digital electronics connected to it. This enables its use in a broad spectrum of project prototyping in areas such as motor control, temperature control and user interface design. Since no hardware is required, is convenient to use as a training or teaching tool.

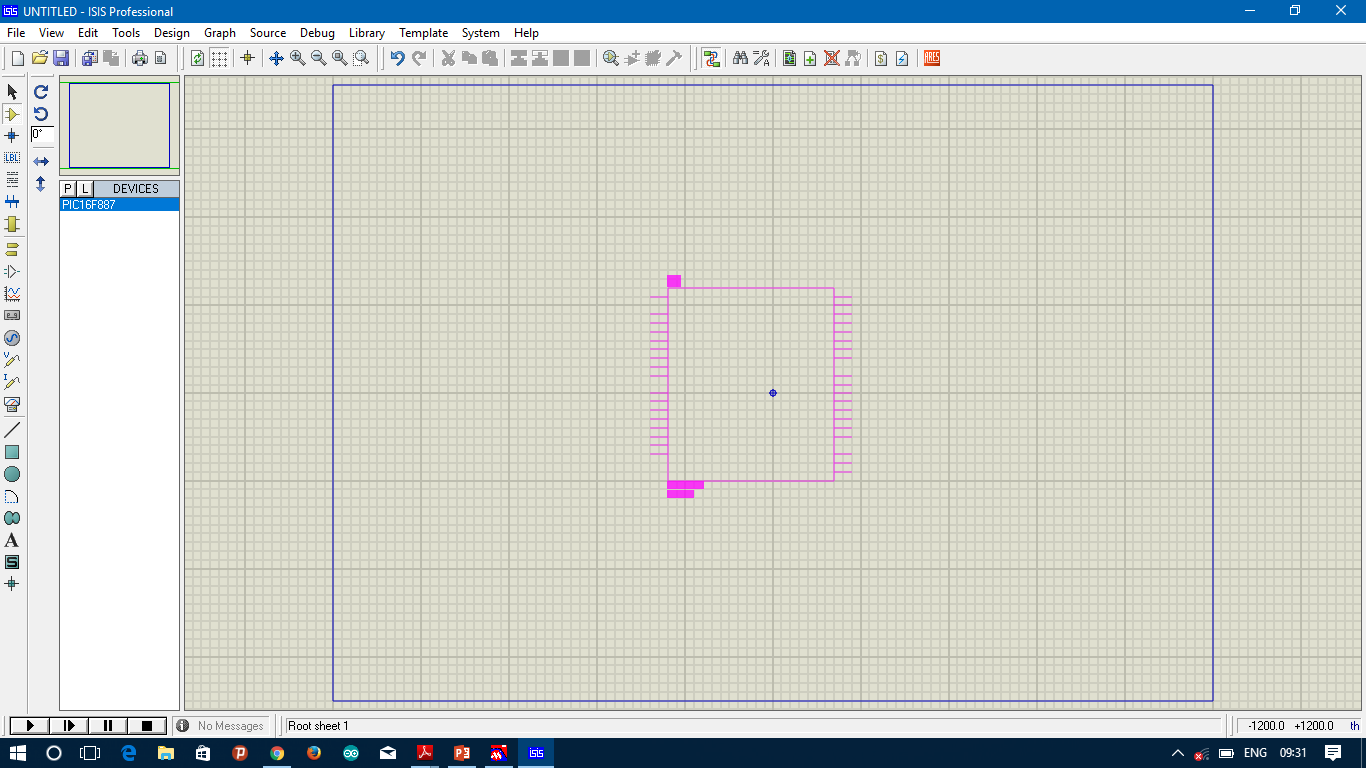
**Step 1:** Open ISIS.exe in the bin directory



**Step 2:** Click on Library>Pick devices



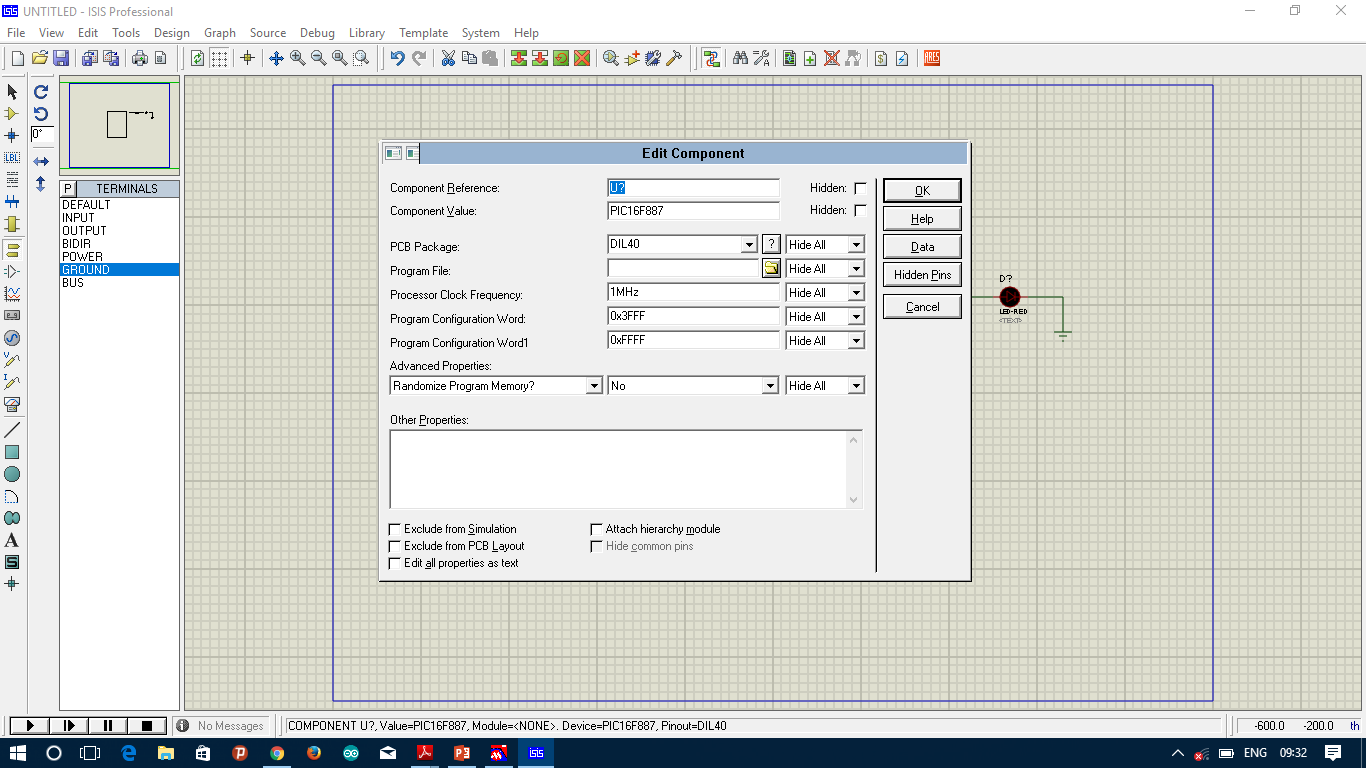
**Step 3:** Search for the component required



**Step 4:** Place the component on the sheet by selecting the component and left clicking on the sheet.



**Step 5:** Similarly pick other components and place them all and make connections.



**Step 6:** Right click on PIC controller and select edit component. Choose program file to load.



**Step 7:** Click on Play button to start simulation and stop button to stop the simulation.

## Programming tools

### 3.3.1 PICkit2

In the field of computer hardware, the term programmer, chip programmer or device programmer refers to an electronic equipment that configures programmable non-volatile integrated circuits (called programmable devices) such as EPROMs, EEPROMs, Flashes, eMMC, MRAM, FRAM, NV RAM, PALs, FPGAs or programmable logic circuits.

PICkit is a family of programmers for PIC microcontrollers made by Microchip Technology. They are used to program and debug microcontrollers, as well as program EEPROM. Some models also feature logic analyser and serial communications (UART) tool.

PICkit2 was introduced in May,2005 replacing the PICkit1. The PICkit 2 uses an internal PIC18F2550 with FullSpeed USB. The latest PICkit 2 firmware allows the user to program and debug most of the 8 and 16 bit PICmicro and dsPIC members of the Microchip product line. The PICkit 2 is open to the public, including its hardware schematic, firmware source code (in C language) and application programs (in C# language). The PICkit 2 has a programmer-to-go (PTG) feature, which can download the hex file and programming instructions into on-board memory (128 KB I²C EEPROM or 256 KB I²C EEPROM), so that no PC is required at the end application.

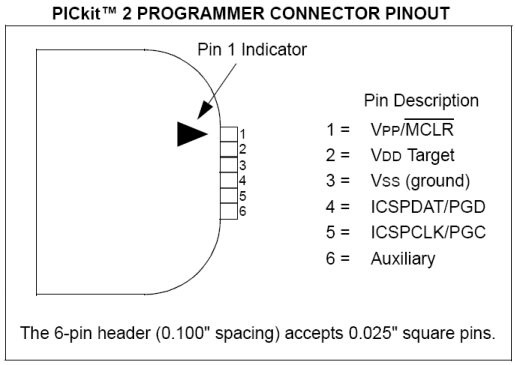


Fig 3.1 PICkit2 Pinout

# 4. EMBEDDED C PROGRAMMING

* LED
* Linear Switches
* Seven Segment Display
* Keypad Matrix
* Multi Seven Segment Display
* Liquid Crystal Display
* Timers / Counters

Contents

## 4.1 LED

A **light-emitting diode** (LED) is a two-lead semiconductor light source. It is a p–n junction diode that emits light when activated. When a suitable voltage is applied to the leads, electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence, and the color of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor. LEDs are typically small and integrated optical components may be used to shape the radiation pattern.

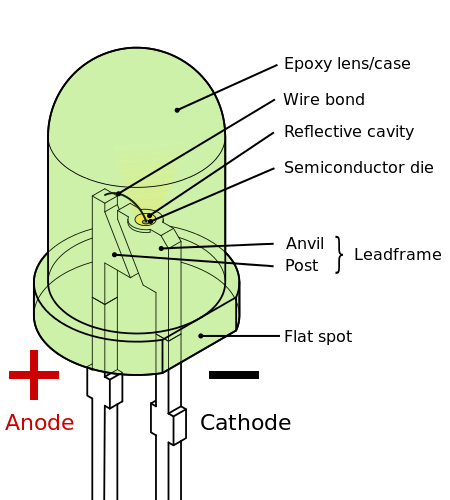


Fig 4.1 LED structure

LED symbol.svg

Fig 4.2 LED symbol

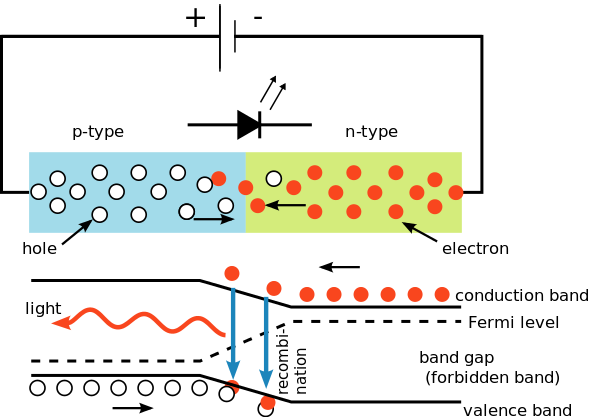
[](https://upload.wikimedia.org/wikipedia/commons/d/d7/PnJunction-LED-E.svg)

Fig 4.3 LED working

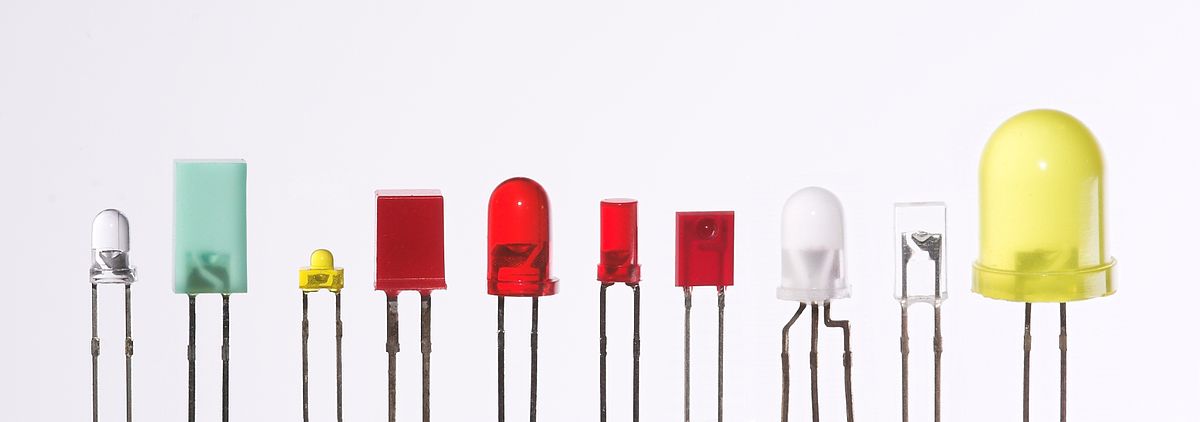
[](https://upload.wikimedia.org/wikipedia/commons/9/9e/Verschiedene_LEDs.jpg)

Fig 4.3 various LEDs

### 4.1.2 LED Interfacing with Microcontroller

Fig 4.4 shows a simple LED circuit having a voltage V volts, current I amperes and a resistor having resistance R Ω (ohm) in series with LED. To ensure proper operation of LED the value of Resistor used should be accurate. Suppose the ratings of LED are 20 mA at 2V, then using **KVL** we can find out the value of resistance which comes out to about 150 Ω

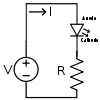


Fig 4.4 Simple LED circuit

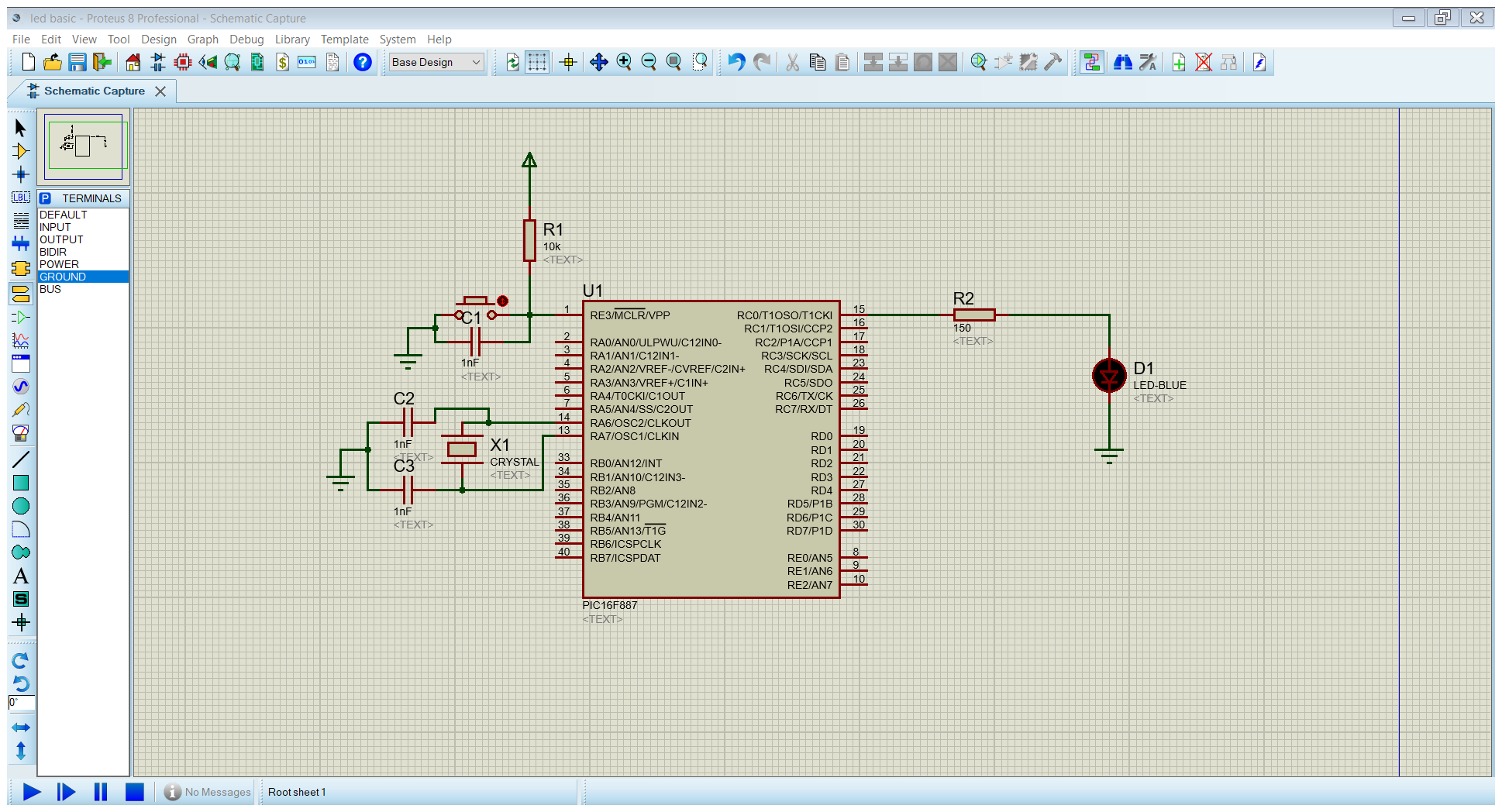


Fig 4.5 Schematic for connecting an LED to the microcontroller.

### 4.1.3 LED programming

## 4.2 Linear Switches

### 4.2.1 Introduction to Switches

A **switch** is a circuit element which is used to make or break a connection in an electrical circuit. In the simplest case, a switch has two conductive pieces, often metal, called contacts, connected to an external circuit, that touch to complete (make) the circuit, and separate to open (break) the circuit.

[SPST-Switch.svg](https://en.wikipedia.org/wiki/File:SPST-Switch.svg)

Fig 4.6 Switch symbol

### 4.2.2 Interfacing of Switches with Microcontroller

Fig 4.7 shows a simple switch circuit, in this circuit when the button is pressed output comes to Vcc but on releasing the button the output is no longer Vcc neither it is ground. The actual voltage on the controller is unknown and this type of input is called floating. To deal with the problem of floating input we use Pull-up or Pull-down resistors.

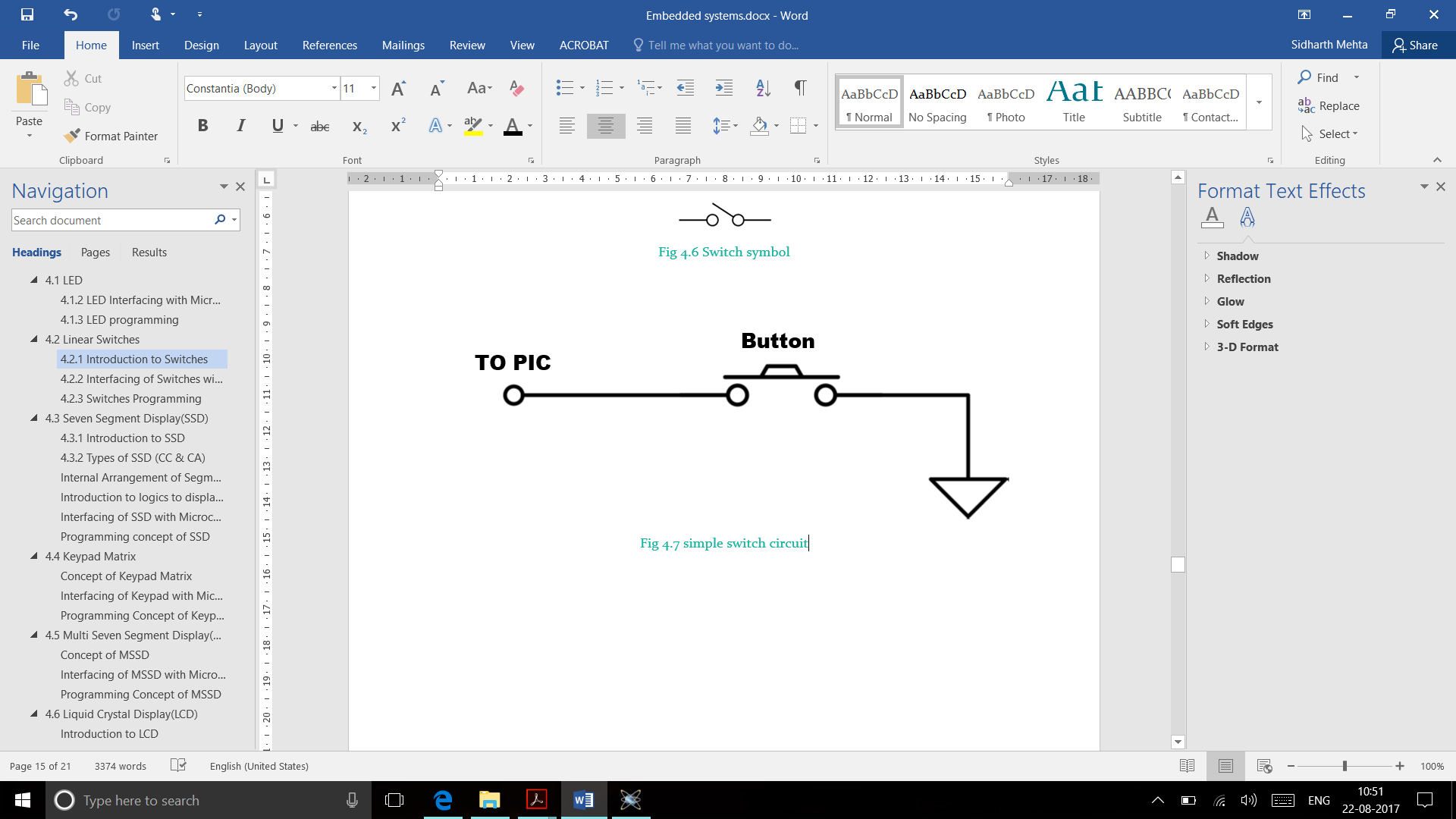


Fig 4.7 simple switch circuit

In electronic logic circuits, a **pull-up resistor** is a resistor connected between a signal conductor and a positive power supply voltage to ensure that the signal will be a valid logic level if external devices are disconnected or high-impedance is introduced. They may also be used at the interface between two different types of logic devices, possibly operating at different logic levels and power supply voltages.

A pull-up resistor pulls the voltage of the signal it is connected to towards its voltage source level. When the other components associated with the signal are inactive, the voltage supplied by the pull up prevails and brings the signal up to a logical high level. When another component on the line goes active, it overrides the pull-up resistor. The pull-up resistor ensures that the wire is at a defined logic level even if no active devices are connected to it.

A **pull-down resistor** works in the same way but is connected to ground. It holds the logic signal at a low logic level when no other active device is connected.

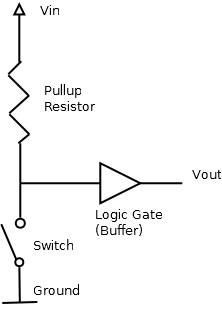
[](https://en.wikipedia.org/wiki/File:Pullup_Resistor.png)

Fig 4.8 Switch circuit with pull-up resistor

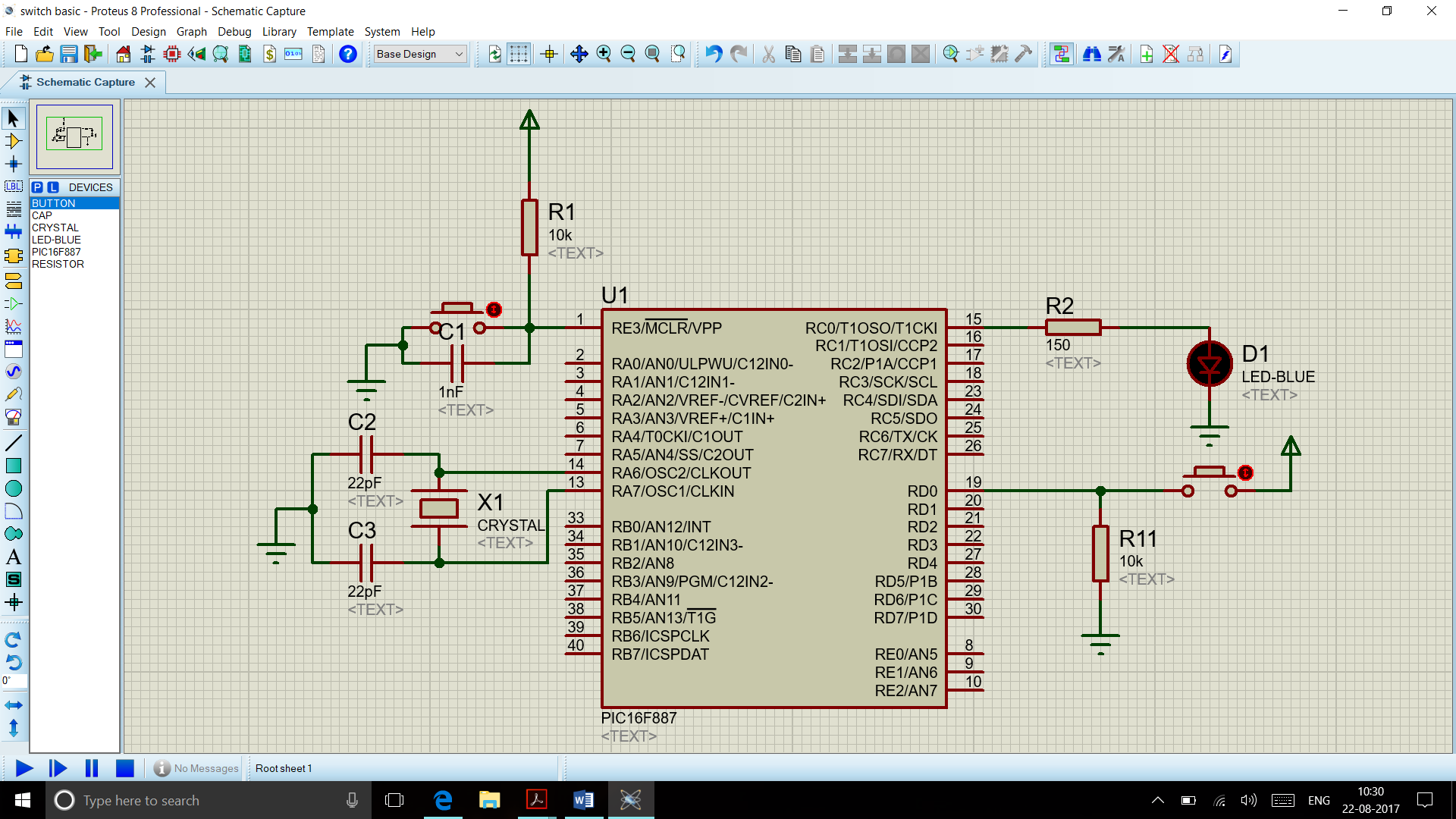


Fig 4.9 Schematic for connecting a switch to the microcontroller

### 4.2.3 Switches Programming

## 4.3 Seven Segment Display(SSD)

### 4.3.1 Introduction to SSD

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

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

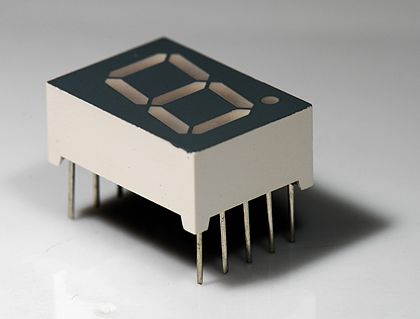
[](https://en.wikipedia.org/wiki/File:Seven_segment_02_Pengo.jpg)

Fig 4.10 Seven segment display

The seven elements of the display can be lit in different combinations to represent the Arabic numerals. Often the seven segments are arranged in an oblique (slanted) arrangement, which aids readability. The seven segments are arranged as a rectangle of two vertical segments on each side with one horizontal segment on the top, middle, and bottom. Additionally, the seventh segment bisects the rectangle horizontally. The segments of a 7-segment display are referred to by the letters A to G, where the optional decimal point (an "eighth segment", referred to as DP) is used for the display of non-integer numbers. The numerals 6 and 9 may be represented by two different glyphs on seven-segment displays, with or without a 'tail'.

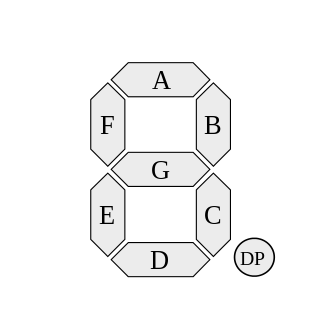
[](https://en.wikipedia.org/wiki/File:7_segment_display_labeled.svg)

Fig 4.11 Internal Arrangement of Segment LEDs in SSD

### 4.3.2 Types of SSD

In a simple LED package, typically all of the cathodes (negative terminals) or all of the anodes (positive terminals) of the segment LEDs are connected and brought out to a common pin; this is referred to as a "common cathode" or "common anode" device.

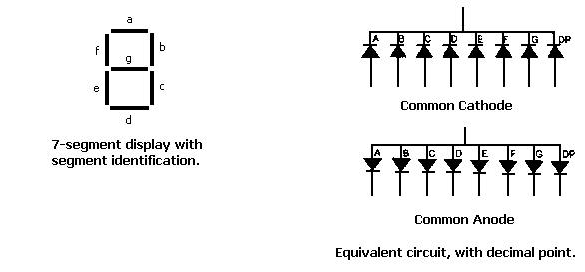


Fig 4.12 SSD displays

### 4.3.3 Logics of displaying Numeric values on SSD

Hexadecimal digits can be displayed on seven-segment displays. Today, a combination of uppercase and lowercase letters is commonly used for A–F this is done to obtain a unique, unambiguous shape for each hexadecimal digit (otherwise, a capital 'D' would look identical to a '0' and a capital 'B' would look identical to an '8'). Also the digit '6' must be displayed with the top bar lit to avoid ambiguity with the letter 'b'. To display a particular character, the LEDs corresponding to its pattern need to be turned **on**, on the SSD. Table 4.1 demonstrates this.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Digit** | **Display** | **gfedcba** | **abcdefg** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| 0 | [0](https://en.wikipedia.org/wiki/File:7-segment_abcdef.svg) | 0×3F | 0×7E | **on** | **on** | **on** | **on** | **on** | **on** | off |
| 1 | [1](https://en.wikipedia.org/wiki/File:7-segment_bc.svg) | 0×06 | 0×30 | off | **on** | **on** | off | off | off | off |
| 2 | [2](https://en.wikipedia.org/wiki/File:7-segment_abdeg.svg) | 0×5B | 0×6D | **on** | **on** | off | **on** | **on** | off | **on** |
| 3 | [3](https://en.wikipedia.org/wiki/File:7-segment_abcdg.svg) | 0×4F | 0×79 | **on** | **on** | **on** | **on** | off | off | **on** |
| 4 | [4](https://en.wikipedia.org/wiki/File:7-segment_bcfg.svg) | 0×66 | 0×33 | off | **on** | **on** | off | off | **on** | **on** |
| 5 | [5](https://en.wikipedia.org/wiki/File:7-segment_acdfg.svg) | 0×6D | 0×5B | **on** | off | **on** | **on** | off | **on** | **on** |
| 6 | [6](https://en.wikipedia.org/wiki/File:7-segment_acdefg.svg) | 0×7D | 0×5F | **on** | off | **on** | **on** | **on** | **on** | **on** |
| 7 | [7](https://en.wikipedia.org/wiki/File:7-segment_abc.svg) | 0×07 | 0×70 | **on** | **on** | **on** | off | off | off | off |
| 8 | [8](https://en.wikipedia.org/wiki/File:7-segment_abcdefg.svg) | 0×7F | 0×7F | **on** | **on** | **on** | **on** | **on** | **on** | **on** |
| 9 | [9](https://en.wikipedia.org/wiki/File:7-segment_abcdfg.svg) | 0×6F | 0×7B | **on** | **on** | **on** | **on** | off | **on** | **on** |
| A | [A](https://en.wikipedia.org/wiki/File:7-segment_abcefg.svg) | 0×77 | 0×77 | **on** | **on** | **on** | off | **on** | **on** | **on** |
| b | [b](https://en.wikipedia.org/wiki/File:7-segment_cdefg.svg) | 0×7C | 0×1F | off | off | **on** | **on** | **on** | **on** | **on** |
| C | [C](https://en.wikipedia.org/wiki/File:7-segment_adef.svg) | 0×39 | 0×4E | **on** | off | off | **on** | **on** | **on** | off |
| d | [d](https://en.wikipedia.org/wiki/File:7-segment_bcdeg.svg) | 0×5E | 0×3D | off | **on** | **on** | **on** | **on** | off | **on** |
| E | [E](https://en.wikipedia.org/wiki/File:7-segment_adefg.svg) | 0×79 | 0×4F | **on** | off | off | **on** | **on** | **on** | **on** |
| F | [F](https://en.wikipedia.org/wiki/File:7-segment_aefg.svg) | 0×71 | 0×47 | **on** | off | off | off | **on** | **on** | **on** |

Table 4.1 Logic for displaying on Common Cathode display

### 4.3.4 Interfacing of SSD with Microcontroller

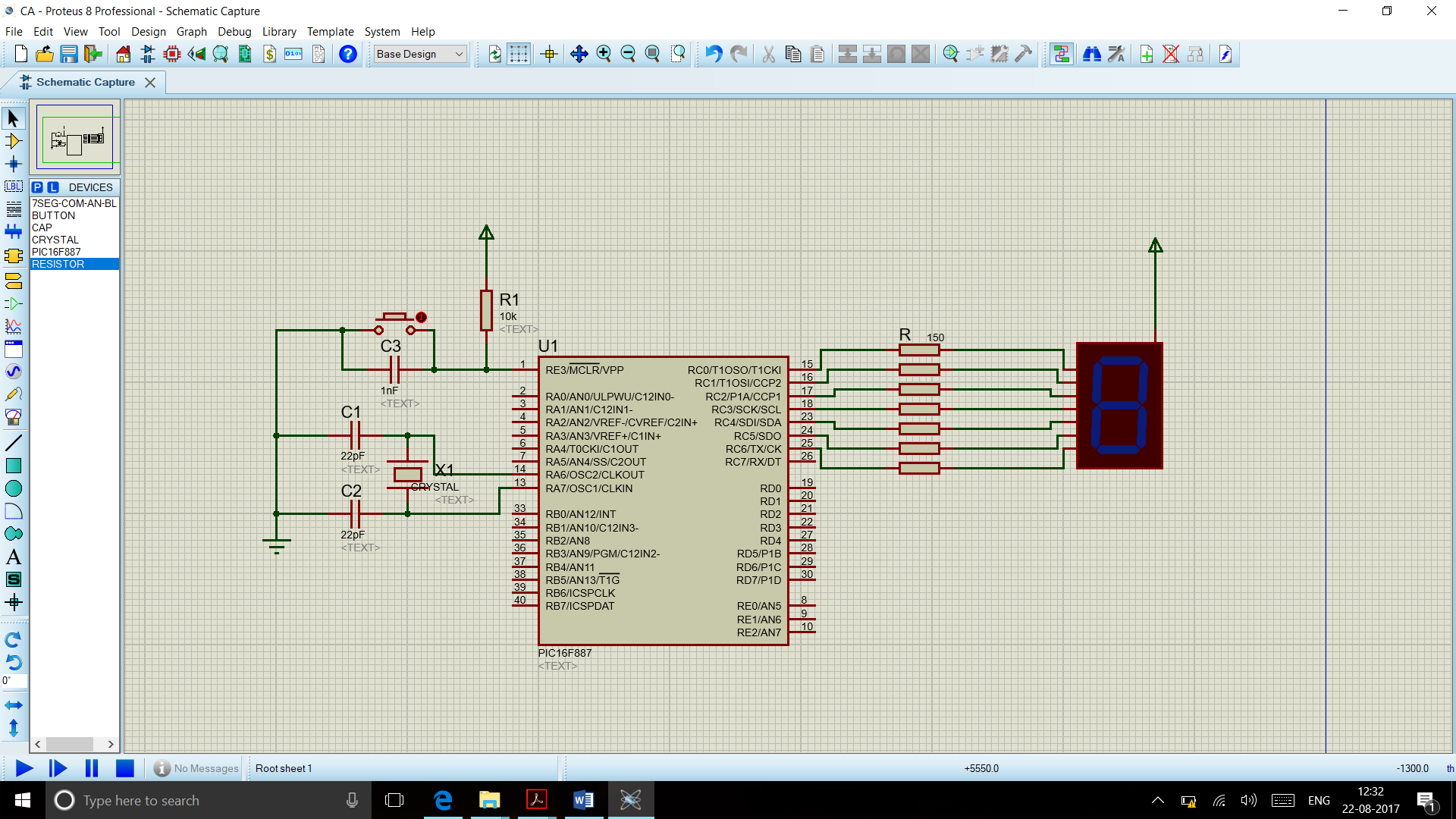


Fig 4.13 Common Anode SSD interfaced by microcontroller

### 4.3.5 SSD Programming

## 4.4 Keypad Matrix

### 4.4.1 Concept of Keypad Matrix

While using switches to take user input a problem arises, the problem is that the no of pins on the controller are limited as a result only a limited number of inputs can be taken from the user using the simple switch arrangement. To solve this problem, the Keypad matrix is used which allows to interface multiple switches with the controller using a limited number of pins.

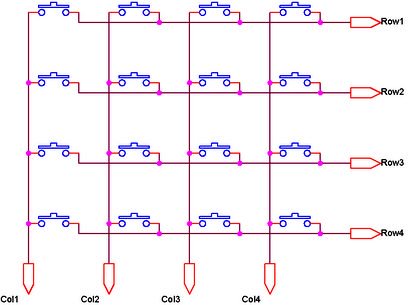


Fig 4.14 Keypad Matrix

In the keypad matrix switches are arranged in form of Rows and Columns, the rows can serve as connection to power source and columns are connected to the controller. The logic behind keypad matrix operation is activating a particular row at a time while keeping the others deactivated. This is done by providing a **HIGH** to one Row and providing other rows with a **LOW**. E.g. If only row 1 is active on pressing button of row 1 and column 4 we would get high on pin connected to column 1, as we know which row is active we can tell the exact button that was pressed.

### 4.4.2 Interfacing of Keypad with Microcontroller

The circuit of Fig 4.15 shows the connection to be made with controller, the pins RD4 to Rd7 serve as inputs. Notice the pull down resistors are provided to remove floating inputs.

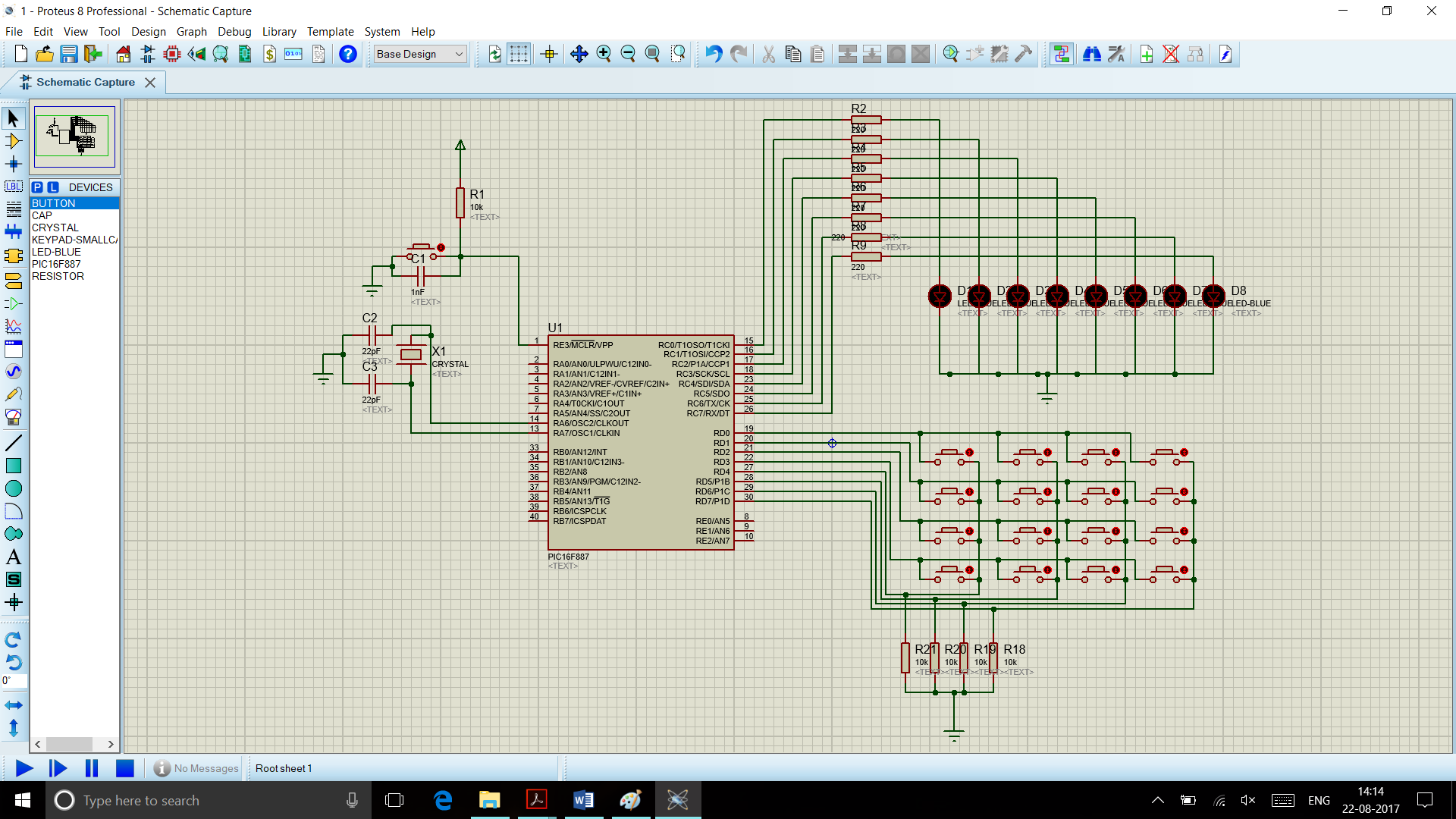


Fig 4.15 Keypad Matrix interfaced by microcontroller

### Programming of Keypad Matrix

## 4.5 Multi Seven Segment Display(MSSD)

### 4.5.1 Concept of MSSD

The MSSD works on the principle of persistence of vision i.e. the eye has can only perceive two frames as distinct if the frame rate is slower than 16 fps. In the MSSD containing n segments, segment 1 - n are not switched on at the same time rather in a 1 by 1 fashion. This method allows to control a large number of segments using only a small number of pins.

In this method the first segment is turned on by giving a **LOW** (common cathode) to its corresponding common pin and then sending data to its segment pins, after this it is turned off by giving **HIGH** to its common pin and switching on the next segment.

### 4.5.2 Interfacing of MSSD with Microcontroller

The MSSD’s common pins are connected to the microcontroller using transistors. Here the transistor is working as a switch and provides a reversed logic. The transistors are used to meet the source and sink current requirements of PIC as the current that will be sourced through or sink in the controller if transistors are not used is larger than that the PORT of controller can handle.

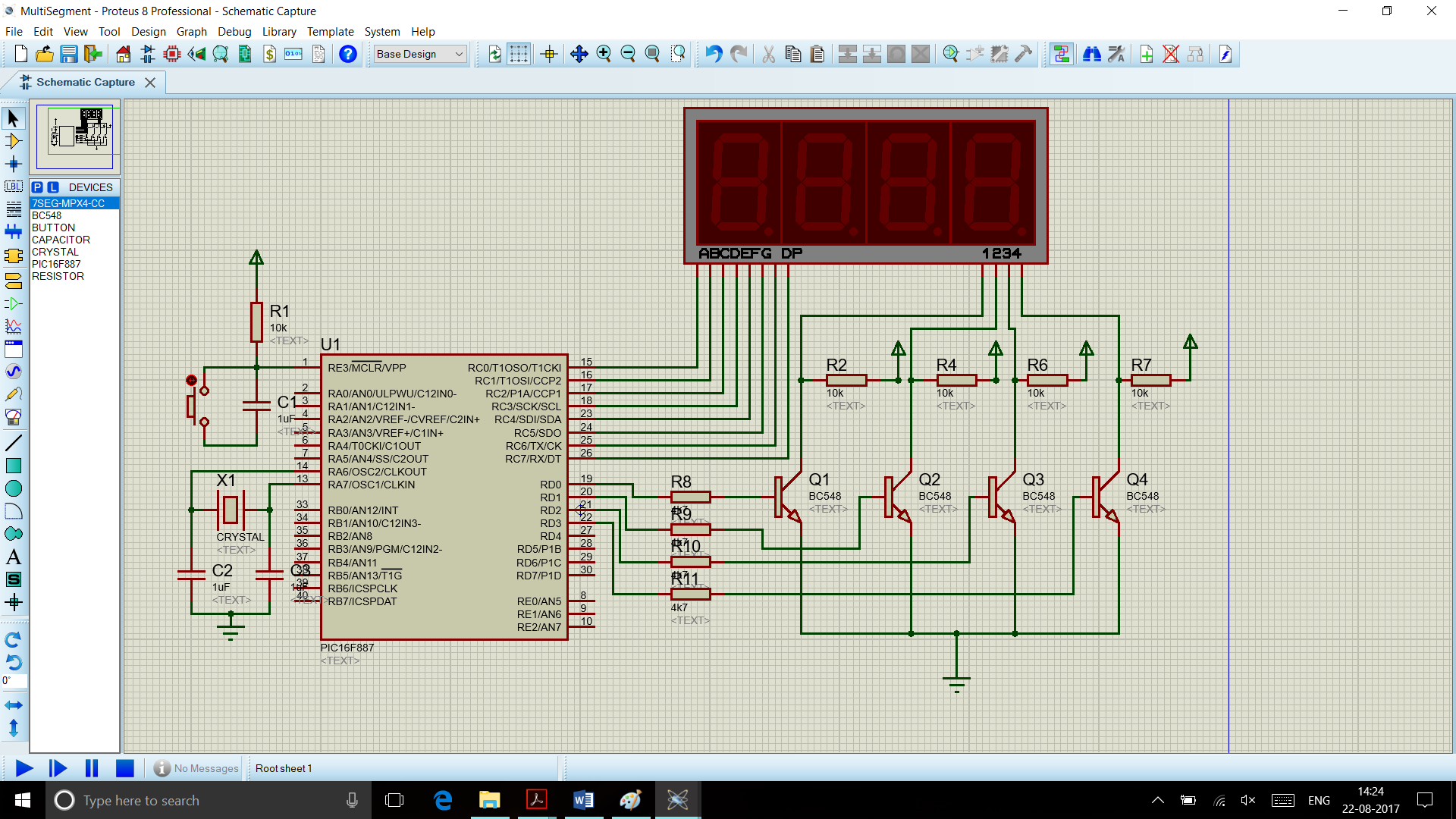


Fig 4.16 MSSD interfaced by microcontroller

### 4.5.3 Programming of MSSD

## 4.6 Liquid Crystal Display(LCD)

A **liquid crystal display**(**LCD**) is a thin, flat display device made up of any number of colour or monochrome pixels arrayed in front of a light source or reflector. It is often utilized in battery powered electronic devices because it uses very small amounts of electric power.

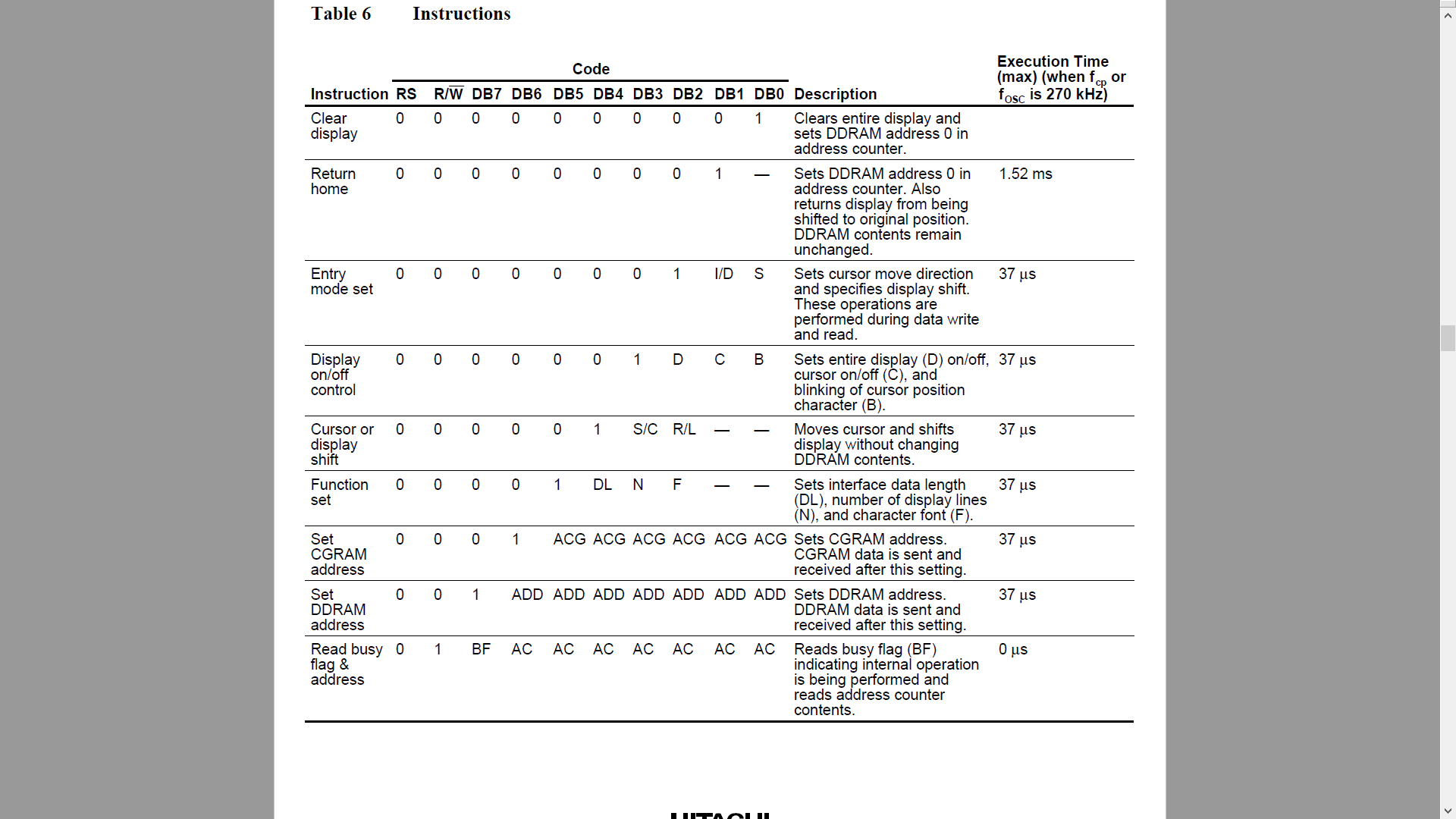
### 4.6.1 Types of LCD

LCDs can be classified into 2 groups: Graphical and Character LCDs

The graphical displays are used to display images or graphics, in a graphic display, the viewing area is laid out as a dot matrix and data can be sent to control each individual dot, whereas character LCD display characters.

Character LCD is available with and without a controller. The controller IC serves as an intelligent interface between the LCD and the microcontroller. The character LCD is generally available in forms like 8 x 2, 16 x 1, 16 x 2, 20 x 2, 20 x 4, 40 x 2, 40 x 4 etc.

### 4.6.2 Introduction to LCD Driver IC(HD44780) & Character LCD Module



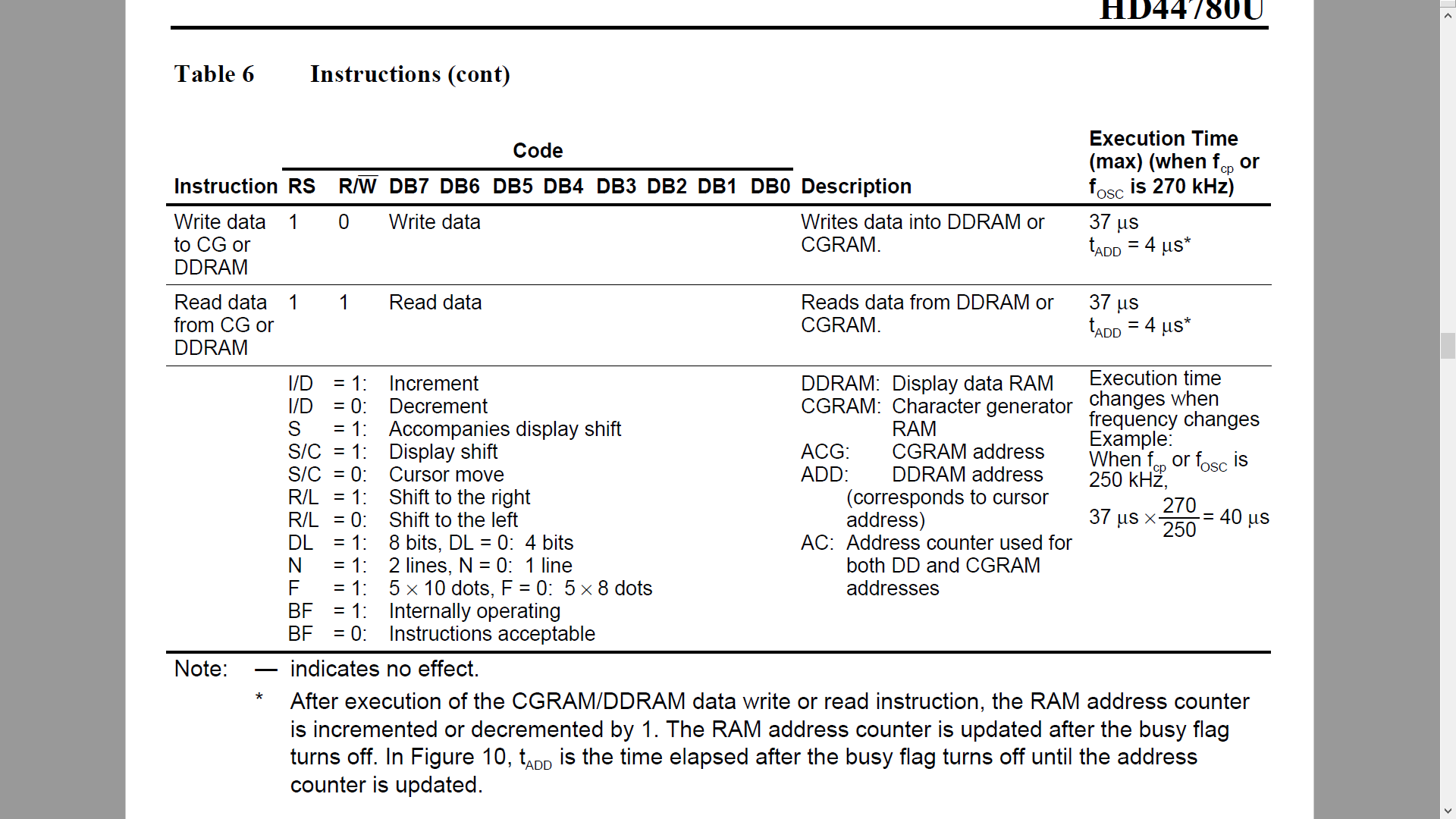


Fig 4.17 HD44780 commands

The 44780 standard requires 3 control lines as well as either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used, the LCD will require a total of 7 data lines. If an 8-bit data bus is used, the LCD will require a total of 11 data lines. The three control lines are **EN**, **RS**, and **RW**.

There are two modes of working of a character LCD.

The LCD memory has two 8-bit register IR (Instruction Register) and DR (Data Register). Instruction uses the Instruction Register. Data mode uses the data register. The IR stores instruction codes, such as display clear and cursor shift, and address information for display data RAM (DDRAM)and character generator RAM (CGRAM). The DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM. By the register selector (RS) signal, these two registers can be selected.

There are two different RAM available in the LCD:

***DDRAM***

The Display Data RAM (DD RAM) stores the display data represented in 8-bit character codes. Its capacity is 80 x 8 bits or 80 characters. The Display Data RAM that is not used for the display can be used as a general data RAM. Display data RAM (DDRAM) is where you send the characters (ASCII code) you want to see on the LCD screen.

***CGRAM***

In the character generator RAM, we can define our own character patterns by program. The CG RAM is a 64 x 8-bit RAM in which the user can program custom character patterns either 8 -5 x 7 character patterns or 4 -5 x 10 character patterns can be written. It has two command **CGRAM address command** and **write command to CGRAM**

Using the commands in Fig 4.17, we can find the LCD instructions in hex

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Decimal** | **HEX** |
| Function set (8-bit interface, 2 lines, 5\*7 Pixels) | 56 | 38 |
| Function set (8-bit interface, 1 line, 5\*7 Pixels) | 48 | 30 |
| Function set (4-bit interface, 2 lines, 5\*7 Pixels) | 40 | 28 |
| Function set (4-bit interface, 1 line, 5\*7 Pixels) | 32 | 20 |
| **Entry mode set** | **See Below** | **See Below** |
| Scroll display one character right (all lines) | 28 | 1E |
| Scroll display one character left (all lines) | 24 | 18 |
| Home (move cursor to top/left character position) | 2 | 2 |
| Move cursor one character left | 16 | 10 |
| Move cursor one character right | 20 | 14 |
| Turn on visible underline cursor | 14 | 0E |
| Turn on visible blinking-block cursor | 15 | 0F |
| Make cursor invisible | 12 | 0C |
| Blank the display (without clearing) | 8 | 08 |
| Restore the display (with cursor hidden) | 12 | 0C |
| Clear Screen | 1 | 01 |
| Set cursor position (DDRAM address) | 128 + address | 80+ address |
| Set pointer in character-generator RAM (CG RAM address) | 64 + address | 40+ address |



Fig 4. LCD JHD162A

### 4.6.3 Introduction to Pin Diagram of Character LCD

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Pin Name** | **Pin Purpose** |
| 1 | VSS | Ground connection |
| 2 | VDD | +5V connection |
| 3 | V0 | Contrast adjustment (to potentiometer) |
| 4 | RS | Register selection (Character vs. Command) |
| 5 | RW | Read/write |
| 6 | EN | Enable |
| 7 | D0 | Data line 0 |
| 8 | D1 | Data line 1 |
| 9 | D2 | Data line 2 |
| 10 | D3 | Data line 3 |
| 11 | D4 | Data line 4 |
| 12 | D5 | Data line 5 |
| 13 | D6 | Data line 6 |
| 14 | D7 | Data line 7 |
| 15 | A | Backlight anode |
| 16 | K | Backlight cathode |

Table 4.2 PIN description

Here’s a breakdown of the pin connections:

■ The contrast adjustment pin changes how dark the display is. It connects to the centre pin of a potentiometer.

■ The register selection pin sets the LCD to command or character mode, so it knows how to interpret the next set of data that is transmitted via the data lines. Based on the state of this pin, data sent to the LCD is either interpreted as a command (for example, move the cursor) or characters (for example, the letter a).

■ The R/W pin tells whether you are writing to the display or reading from it.

|  |  |  |
| --- | --- | --- |
| RS | R/W | Operation |
| 0 | 0 | IR write as an internal operation (display clear, etc.) |
| 0 | 1 | Read busy flag (DB7) and address counter (DB0 to DB6) |
| 1 | 0 | DR write as an internal operation (DR to DDRAM or CGRAM) |
| 1 | 1 | DR read as an internal operation (DDRAM or CGRAM to DR) |

■ The EN pin is used to tell the LCD when data is ready.

■ Data pins 4–7 are used for actually transmitting data, and data pins 0–3 are left unconnected.

■ You can illuminate the backlight by connecting the anode pin to 5V and the cathode pin to ground if you are using an LCD with a built-in resistor for the backlight. If you are not, you must put a current-limiting resistor in-line with the anode or cathode pin. The datasheet for your device will generally tell you if you need to do this.

You can connect the communication pins of the LCD to any I/O pins on the PIC

### 4.6.4 Interfacing of Character LCD with Microcontroller

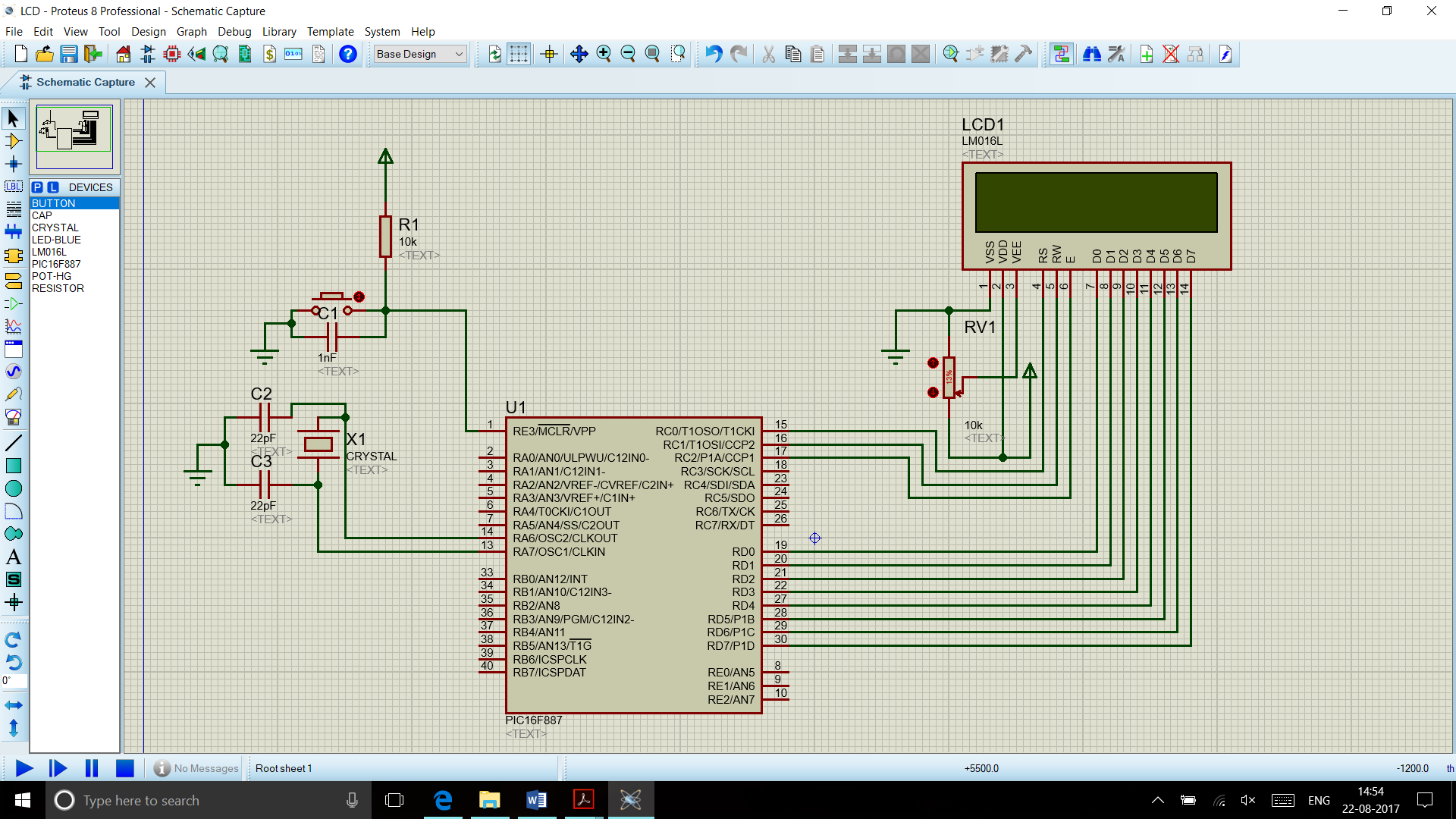


Fig 4. LCD interfaced by microcontroller

### 4.6.5 Programming Concept of Character LCD

## 4.7 Timers / Counters

Timers/Counters are a peripheral which are used to tell the time that has elapsed since a particular event has elapsed or count some external signal/event.

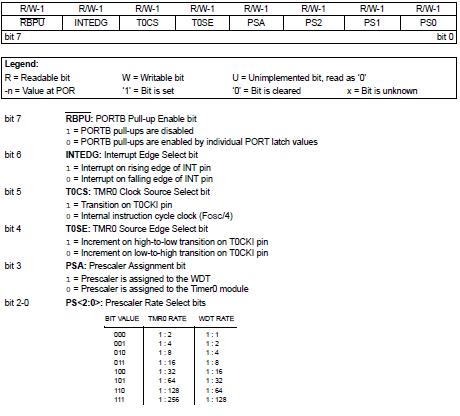
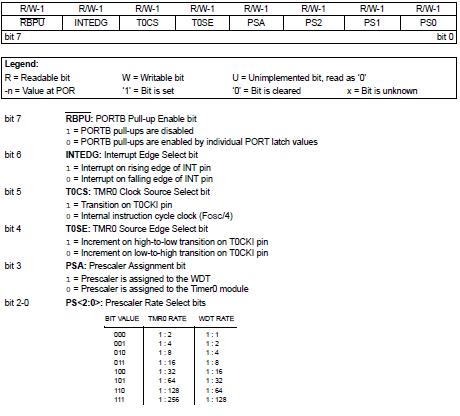
In PIC16F887 there are 3 peripherals available:

* Timer0
* Timer1
* Timer2

### 4.7.1 Timer0

The timer0 module is an 8-bit timer counter which means that the maximum value it can count is 255. The TMR0 register is used to store the count and this register is readable as well as writeable. When the Timer0 reaches 255 an overflow interrupt is generated. It can work on internal as well as external clock. For further division of clock, a pre-scaler is used. The Timer0 SFRs are:

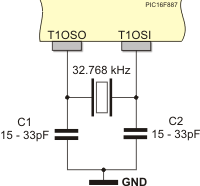
**OPTION Register**



### 4.7.2Timer1

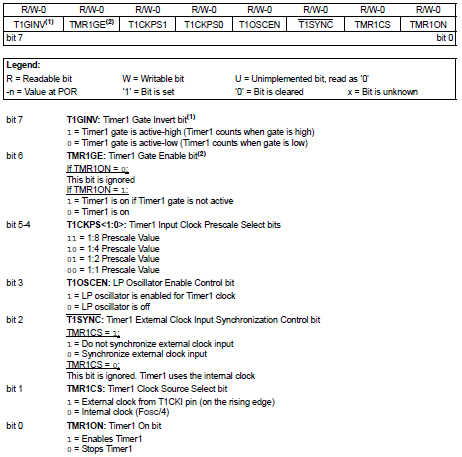
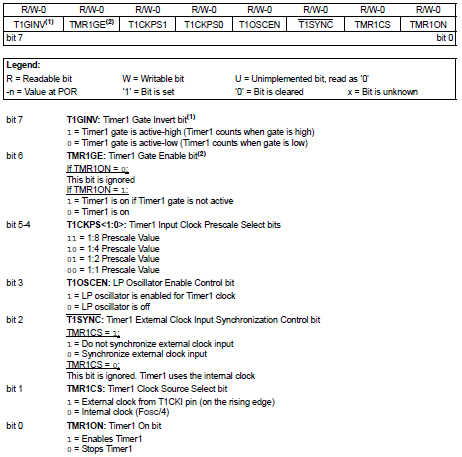
The Timer1 module is a 16-bit timer/counter which means that it consists of two registers (TMR1L and TMR1H). It can count up 65.535 pulses in a single cycle, i.e. before the counting starts from zero. It can count internal or external clock source.

It has a 3-bit pre-scaler T which allows 1, 2, 4 or 8 division of the clock input frequency. The pre-scaler counter is automatically cleared after writing to the TMR1H or TMR1L register.



RC0/T1OSO and RC1/T1OSI pins are used to register pulses coming from peripheral electronics, but they also have an additional function. They are simultaneously configured as both input (pin RC1) and output (pin RC0) of additional LP quartz oscillator (Low Power). This circuit is primarily designed for the operation at low frequencies (up to 200 KHz), more precisely, for the use of 32,768 KHz quartz crystal. Since this oscillator does not depend on internal clock, it can operate even in sleep mode. It is enabled by setting the T1OSCEN control bit of the T1CON register.

**T1CON**



### 4.7.3 Programming concept of Timers to Generate delays

# 5. ADVANCED INTERNAL PERIPHERALS

* Analog to Digital Converter
* UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER
* EEPROM
* INTERRUPTS

Contents

## 5.1 Analog to Digital Converter(ADC)

An analog-to-digital converter (ADC) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal.

The conversion involves quantization of the input, so it necessarily introduces a small amount of error. Furthermore, instead of continuously performing the conversion, an ADC does the conversion periodically, sampling the input. The result is a sequence of digital values that have been converted from a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal.

The resolution of the converter indicates the number of discrete values it can produce over the range of analog values.

Resolution can also be defined electrically and expressed in volts. The minimum change in voltage required to guarantee a change in the output code level is called the least significant bit (LSB) voltage. The resolution Q of the ADC is equal to the LSB voltage. The voltage resolution of an ADC is equal to its overall voltage measurement range divided by the number of intervals:



where M is the ADC's resolution in bits and EFSR is the full-scale voltage range (also called 'span'). EFSR is given by



where VRefHi and VRefLow are the upper and lower extremes, respectively, of the voltages that can be coded.

### 5.1.1 Types of ADC

**Direct-conversion:**

A direct-conversion ADC or flash ADC has a bank of comparators sampling the input signal in parallel, each firing for their decoded voltage range. The comparator bank feeds a logic circuit that generates a code for each voltage range. Direct conversion is very fast, capable of gigahertz sampling rates, but usually has only 8 bits of resolution or fewer, since the number of comparators needed, 2N – 1, doubles with each additional bit, requiring a large, expensive circuit. ADCs of this type have a large die size, a high input capacitance, high power dissipation, and are prone to produce glitches at the output (by outputting an out-of-sequence code). They are often used for video, wideband communications or other fast signals in optical storage.

**Successive approximation:**

A successive-approximation ADC uses a comparator to successively narrow a range that contains the input voltage. At each successive step, the converter compares the input voltage to the output of an internal digital to analog converter which might represent the midpoint of a selected voltage range. At each step in this process, the approximation is stored in a successive approximation register (SAR). For example, consider an input voltage of 6.3 V and the initial range is 0 to 16 V. For the first step, the input 6.3 V is compared to 8 V (the midpoint of the 0–16 V range). The comparator reports that the input voltage is less than 8 V, so the SAR is updated to narrow the range to 0–8 V. For the second step, the input voltage is compared to 4 V (midpoint of 0–8). The comparator reports the input voltage is above 4 V, so the SAR is updated to reflect the input voltage is in the range 4–8 V. For the third step, the input voltage is compared with 6 V (halfway between 4 V and 8 V); the comparator reports the input voltage is greater than 6 volts, and search range becomes 6–8 V. The steps are continued until the desired resolution is reached.

**Ramp-compare:**

A ramp-compare ADC produces a saw-tooth signal that ramps up or down then quickly returns to zero. When the ramp starts, a timer starts counting. When the ramp voltage matches the input, a comparator fires, and the timer's value is recorded. Timed ramp converters require the least number of transistors. The ramp time is sensitive to temperature because the circuit generating the ramp is often a simple oscillator. A very simple (non-linear) ramp-converter can be implemented with a microcontroller and one resistor and capacitor.

### 5.1.2 Introduction to PIC internal ADC and its SFRs

The ADC available in PIC16F887 is a 10-bit ADC, i.e. the conversion results in a 10-bit binary number. There are 14 channels which can be used one by one. The positive and negative reference voltage (Vref+ and Vref-) of ADC is software selectable, which can be Vdd, Vss or voltage at RA3 and RA2 respectively.

Resolution of ADC = **(Vref+ – Vref-)/(210 – 1)**, which is the minimum voltage required to change the ADC result by one bit. Here ADC Resolution = **(5 – 0)/(1024 – 1)** = 5/1023 = 0.004887V

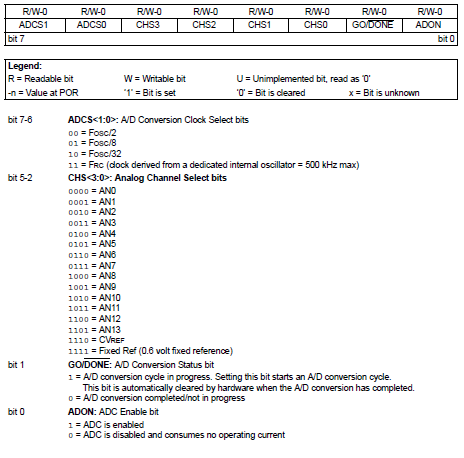
The ADC uses successive approximation method and stores the final result in ADC result registers (ADRESL and ADRESH).

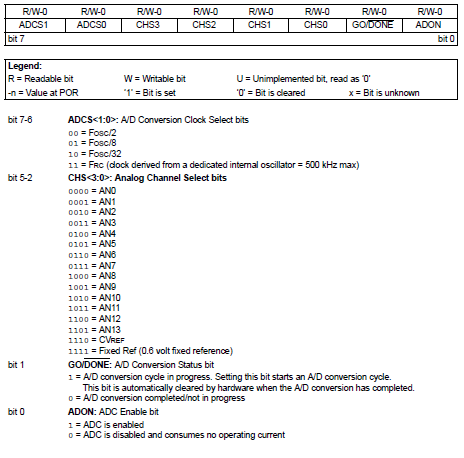
The ADC has 4 SFRs:

* ADCON0
* ADCON1
* ADRESL
* ADRESH

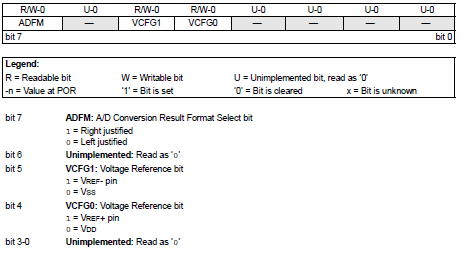
The ADCON0 and ADCON1 registers are used to configure the ADC. The ADRESH and ADRESL registers are used for storing the result.

**ADCON0**

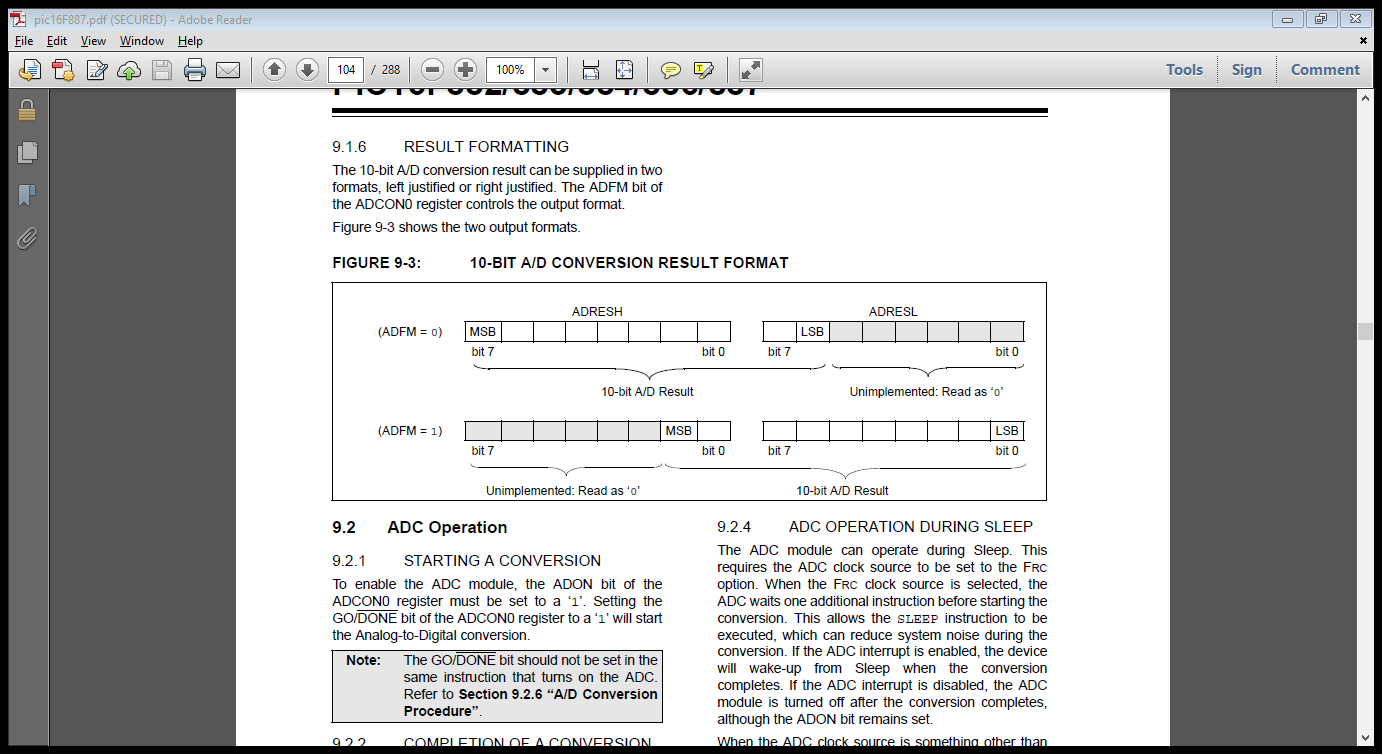
****

****

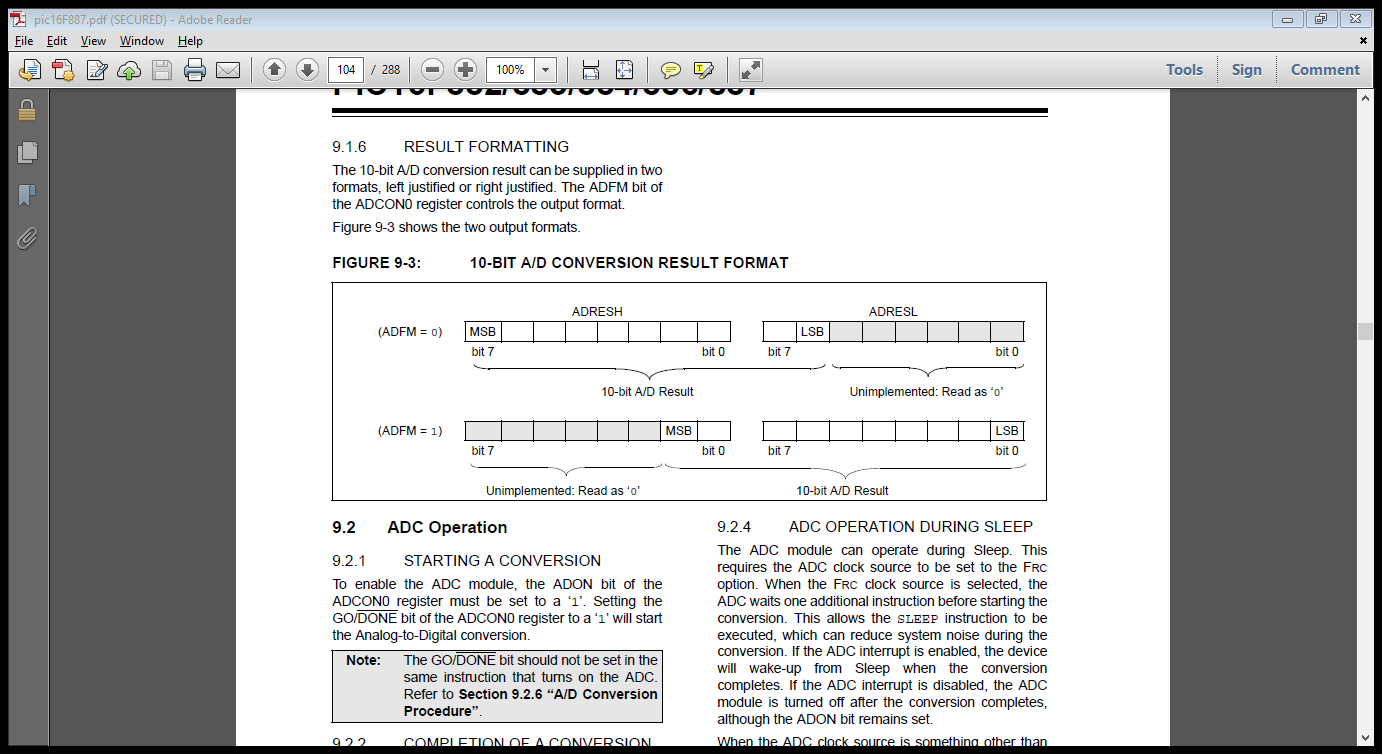
**ADCON1**

****

The 10-bit, ADC result can be stored in two formats i.e. left justified data and right justified data. The ADFM bit in ADCON1 register is used to select the result format

****

Left Justified

****

Right Justified

### 5.1.3 Programming Concept of ADC

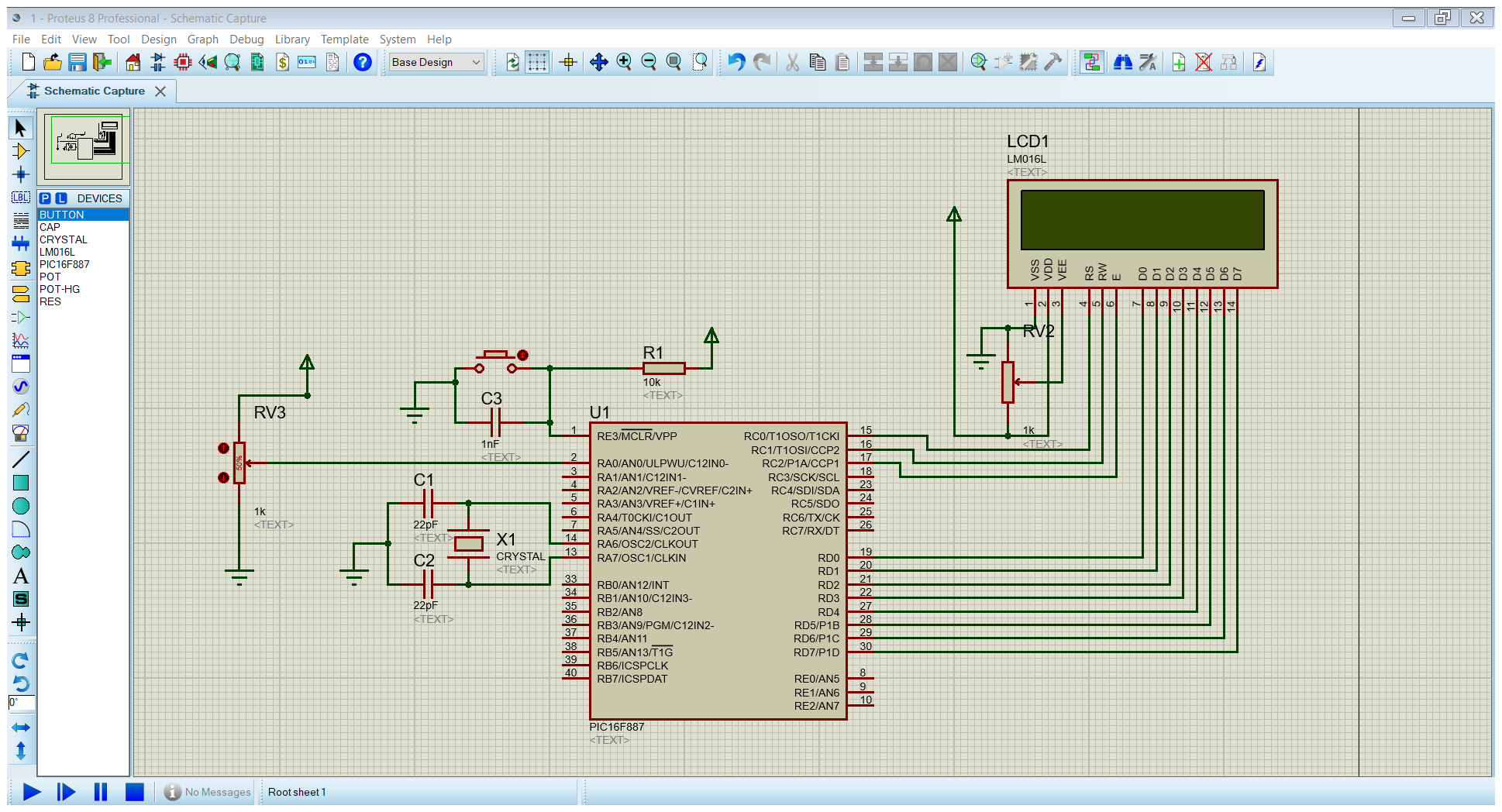


Fig 5. ADC interfacing with microcontroller

## 5.2 Universal Asynchronous Receiver Transmitter(UART)

A universal asynchronous receiver/transmitter is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. The electric signaling levels and methods are handled by a driver circuit external to the UART. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers.

The universal asynchronous receiver/transmitter (UART) takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms.

Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires.

Communication may be:

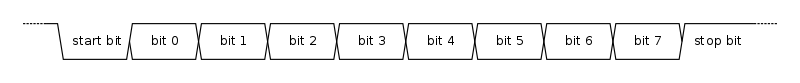
Simplex (in one direction only, with no provision for the receiving device to send information back to the transmitting device). For ex- FM transmission.

Full duplex (both devices send and receive at the same time). For ex-Mobile Phones

Half duplex (devices take turns transmitting and receiving). For ex- Walkie-talkie

**Baud Rate:** The baud rate is the rate at which information is transferred in a communication channel. In the serial port context, "9600 baud" means that the serial port is capable of transferring a maximum of 9600 bits per second.

**Data Framing:** The idle, no data state is high-voltage, or powered. This is a historic legacy from telegraphy, in which the line is held high to show that the line and transmitter are not damaged. Each character is framed as a logic low start bit, data bits, possibly a parity bit, and one or more stop bits. In most applications the least significant data bit (the one on the left in this diagram) is transmitted first.



The start bit signals the receiver that a new character is coming. The next five to nine bits, depending on the code set employed, represent the character. If a parity bit is used, it would be placed after all of the data bits.

**Receiver:** The receiver tests the state of the incoming signal on each clock pulse, looking for the beginning of the start bit.

If the apparent start bit lasts at least one-half of the bit time, it is valid and signals the start of a new character. If not, it is considered a spurious pulse and is ignored. After waiting a further bit time, the state of the line is again sampled and the resulting level clocked into a shift register.

After the required number of bit periods for the character length (5 to 8 bits, typically) have elapsed, the contents of the shift register are made available (in parallel fashion) to the receiving system. The UART will set a flag indicating new data is available and may also generate a processor interrupt to request that the host processor transfers the received data.

**Transmitter:** Transmission operation is simpler as the timing does not have to be determined from the line state, nor is it bound to any fixed timing intervals.

As soon as the sending system deposits a character in the shift register (after completion of the previous character), the UART generates a start bit, shifts the required number of data bits out to the line, generates and sends the parity bit (if used), and sends the stop bits.

Since full-duplex operation requires characters to be sent and received at the same time, UARTs use two different shift registers for transmitted and received characters.

Since transmission of single or multiple characters may take a long time relative to CPU speeds, a UART maintains a flag showing busy status so that the host system knows if there is at least one character in the transmit buffer or shift register.

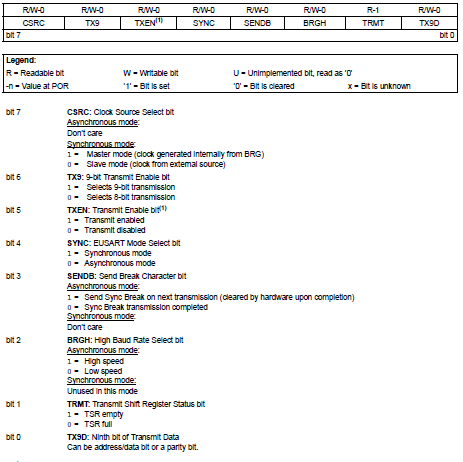
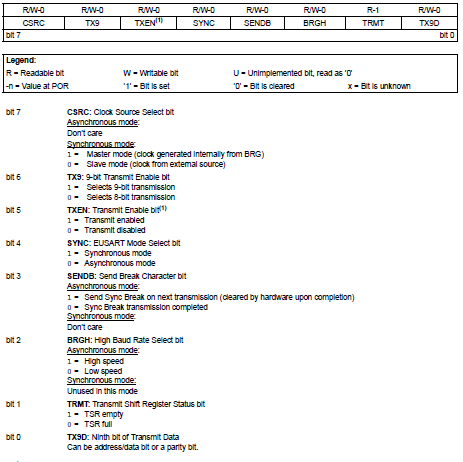
### 5.2.1 UART and its SFRs

The EUSART module includes the following capabilities:

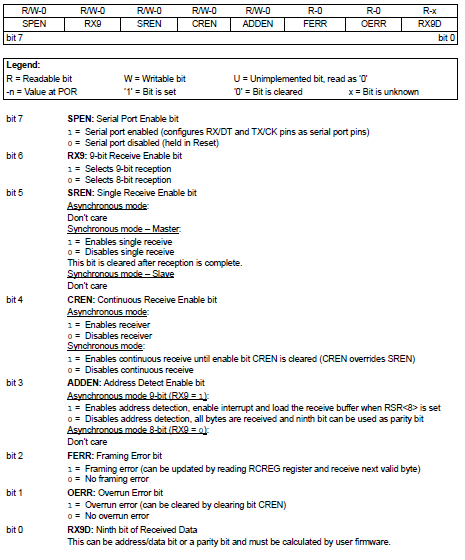
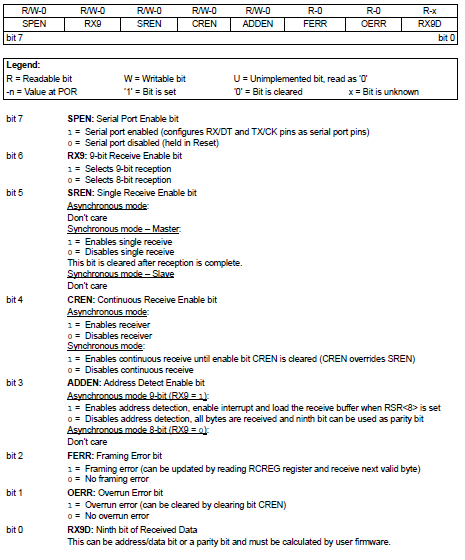
* Full-duplex asynchronous transmit and receive
* Two-character input buffer
* One-character output buffer
* Programmable 8-bit or 9-bit character length
* Address detection in 9-bit mode
* Input buffer overrun error detection
* Received character framing error detection

SFR for UART:

**TXSTA**

**** ****

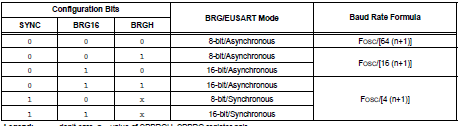
**RCSTA**

**** ****

**SPBRG and SPBRGH**

SPBRG and SPBRGH registers are used for Baud Rate generation.

Depending upon the values of BRGH and BRG16, the baud rate generation formula can be found out from below table, where n is the value of SPBRG and SPBRGH register and Fosc is the frequency of the oscillator used.

****

### 5.2.2 Programming concept of Serial Transmitter & Receiver using UART

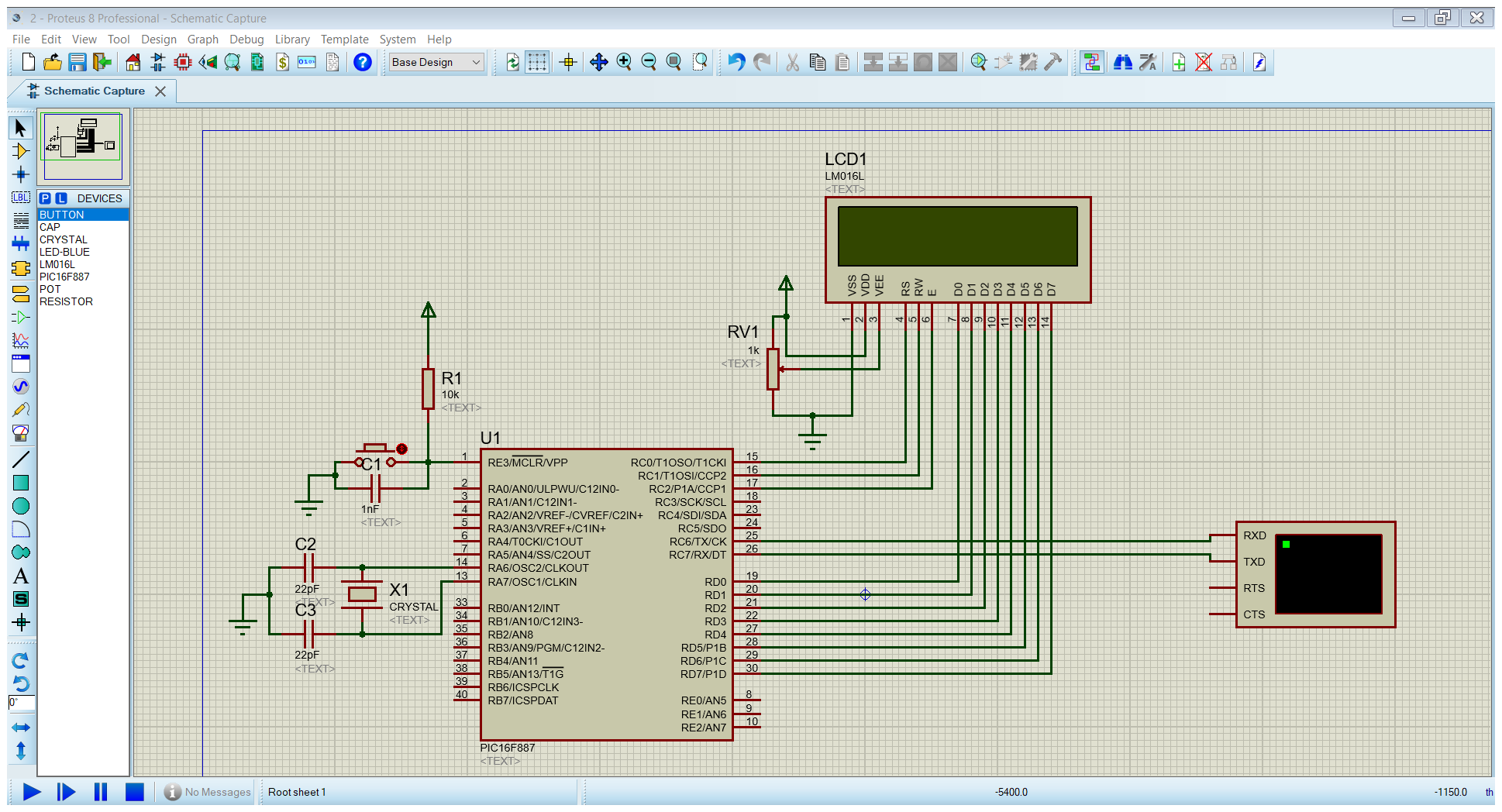


Fig 5. USART interfacing with microcontroller

## 5.3 EEPROM

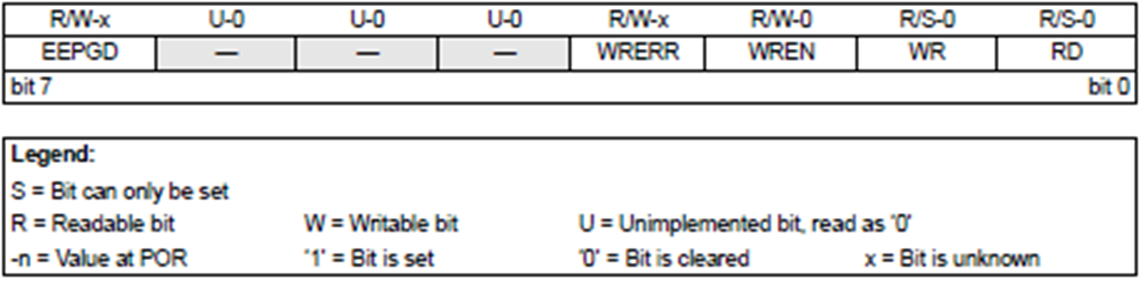
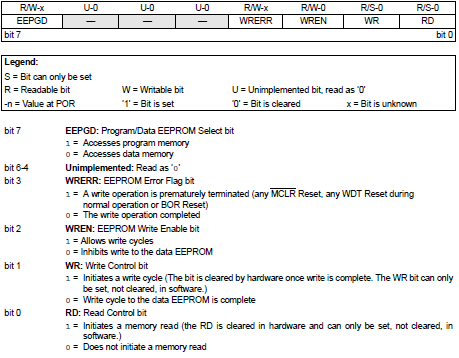
EEPROM stands for electrically erasable programmable read-only memory. It is a type of non-volatile memory used in computers and other electronic devices to store relatively small amounts of data but allowing individual bytes to be erased and reprogrammed. EEPROMs are organized as arrays of floating-gate transistors. EEPROMs can be programmed and erased in-circuit, by applying special programming signals. This device has 256 bytes of data EEPROM with an address range from 0h to 0FFh.

The Data EEPROM and Flash program memory are readable and writable during normal operation (full VDD range). These memories are not directly mapped in the register file space. Instead, they are indirectly addressed through the Special Function Registers (SFRs).

### EEPROM SFRs

There are six SFRs used to access these memories:

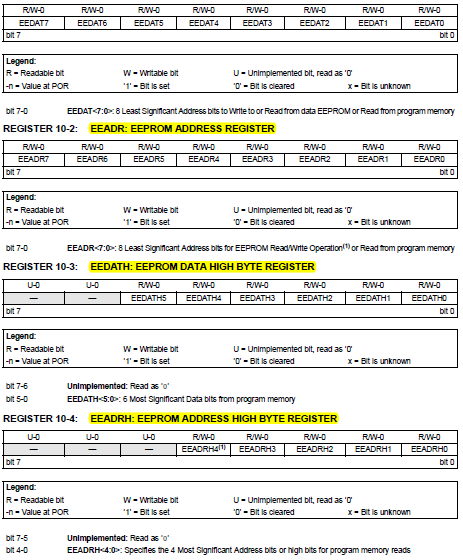
**EECON1**

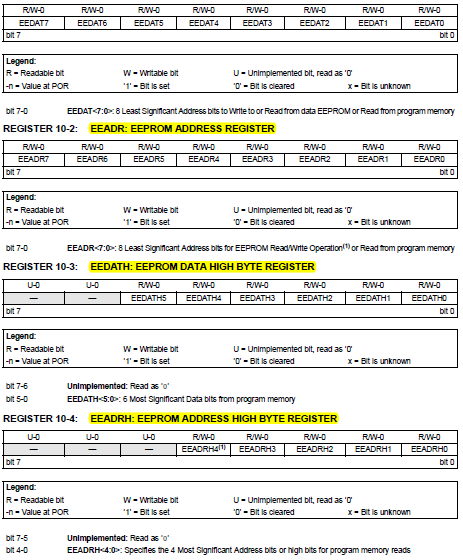
**EECON2**

EECON2 is not a physical register. Reading EECON2 will read all ‘0’s. The EECON2 register is used exclusively in the data EEPROM write sequence.

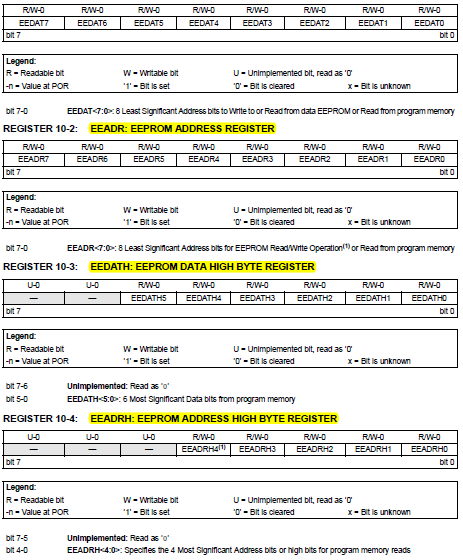
**EEDAT**



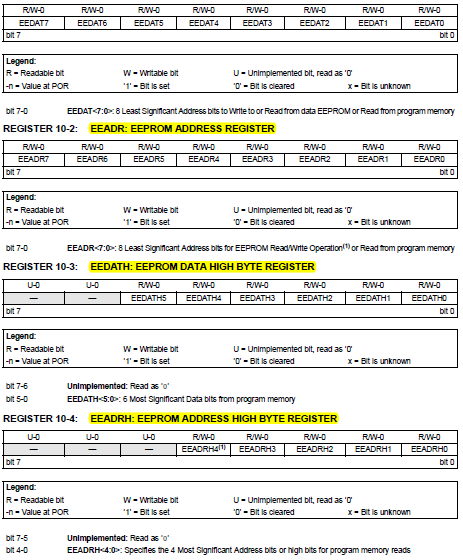
**EEDATH**

****

**EEADR**

****

**EEADRH**

****

When interfacing the data memory block, EEDAT holds the 8-bit data for read/write, and EEADR holds the address of the EEDAT location being accessed.

The EEPROM data memory allows byte read and write. A byte write automatically erases the location and writes the new data (erase before write).

### 5.3.4 Programming concept of EEPROM Writing & Reading

## 5.4 Interrupts

In system programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities. Each interrupt has its own interrupt handler. The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts. Interrupts are a commonly used technique for computer multitasking, especially in real-time computing.

### 5.4.2 Polling Vs. Interrupt

**Polling:** Using this method, the microcontroller accesses at the exact time interval the external device and gets the required information. The user is the one who determines the time intervals in which microcontroller contacts the device. The main drawback of this method when writing program is waste of time of microcontroller, which needs to wait and check whether the new information has arrived.

**Interrupt:** Interrupt is the signal sent to the microcontroller to notify of an event that requires immediate attention. An Interrupt signal requests the processor to suspend the execution of current program and jump to execute a special code called ISR. Once the system has finished the ISR, the processor will start the execution from where the program was suspended.

The request for the microcontroller to free itself to execute the interrupt could come from several sources.

### 5.4.3 Types of Interrupts

**Hardware interrupts:** Hardware interrupts are used by devices to communicate that they require attention from the operating system. Internally, hardware interrupts are implemented using electronic alerting signals that are sent to the processor from an external device, which is a part of the system.

**Software interrupts:** A software interrupt is caused either by an exceptional condition in the processor itself, or a special instruction in the instruction set which causes an interrupt when it is executed. For example, a divide-by-zero exception will be thrown if the processor's arithmetic logic unit is commanded to divide a number by zero as this instruction is an error and impossible.

Interrupts can also be categorized as:

**Maskable interrupt:** A hardware interrupt that may be ignored by setting a bit in an interrupt mask register's (IMR) bit-mask.

**Non-maskable interrupt (NMI):** A hardware interrupt that lacks an associated bit-mask, so that it can never be ignored. NMIs are used for the highest priority tasks such as timers, especially watchdog timers.

### 5.4.5 Interrupts in PIC16F887

In PIC16F887, there are a total of 15 Interrupts available. These are:

* External Interrupt RB0/INT
* Timer0, Timer1 Overflow Interrupt
* PORTB Change Interrupts
* 2 Comparator Interrupts
* A/D Interrupt
* Timer2 Match Interrupt
* EEPROM Data Write Interrupt
* Fail-Safe Clock Monitor Interrupt
* Enhanced CCP Interrupt
* EUSART Receive and Transmit Interrupts
* Ultra-Low-Power Wake-up Interrupt
* MSSP Interrupt

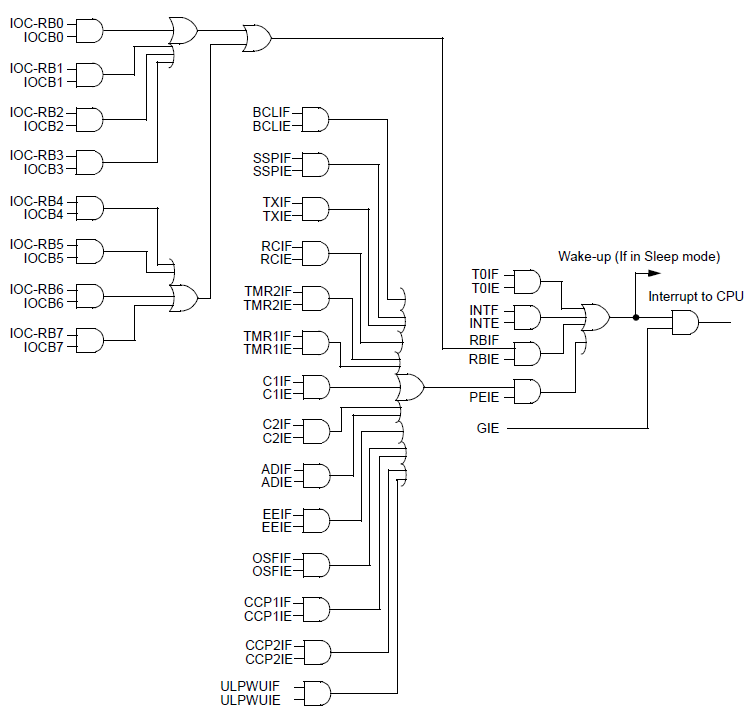
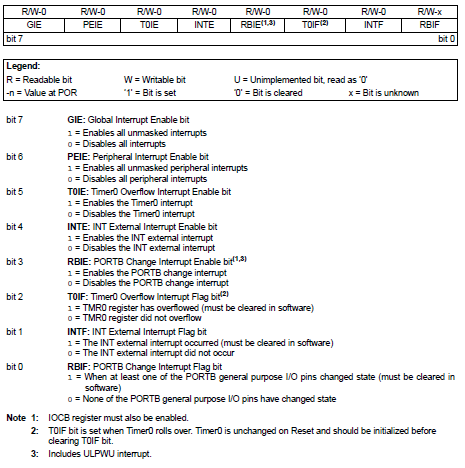
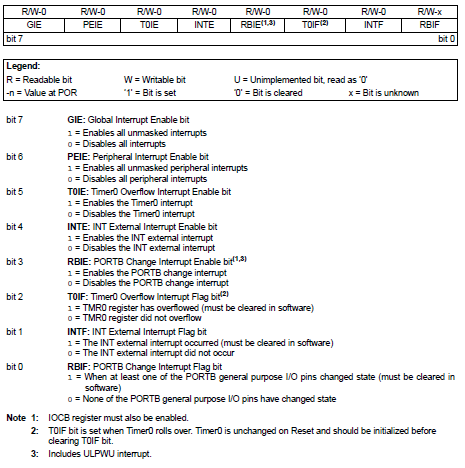


Fig 5. Interrupt Logic Diagram of PIC16F887

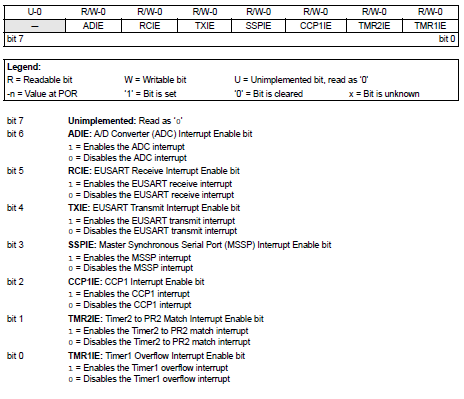
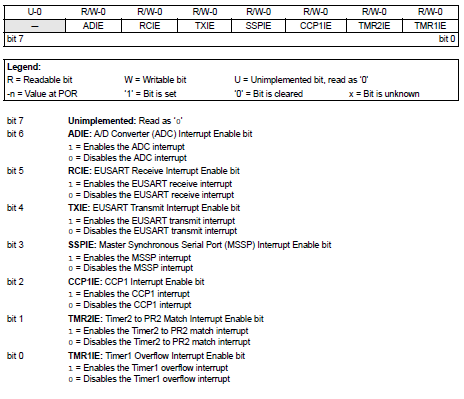
The SFR used for interrupts in PIC16F887 are:

**INTCON: Interrupt Control Register**

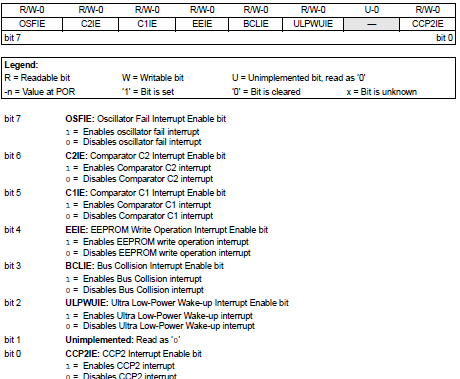
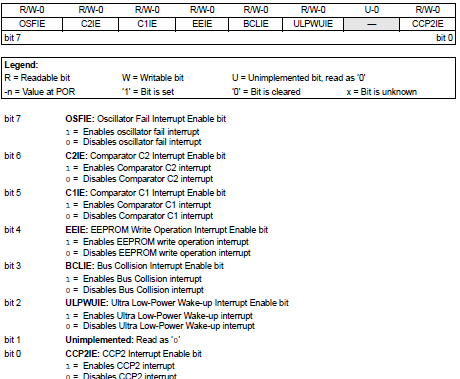




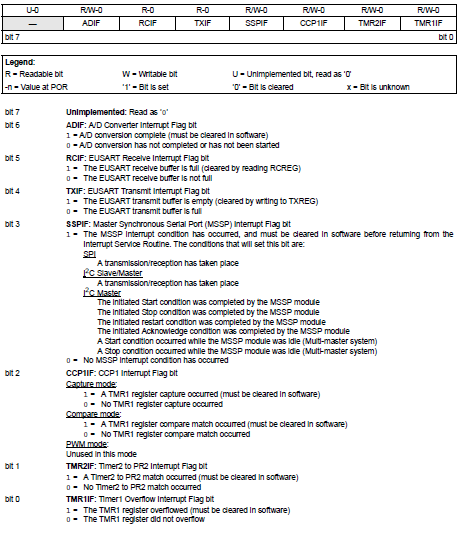
**PIE1: Peripheral Interrupt Enable 1 Register**

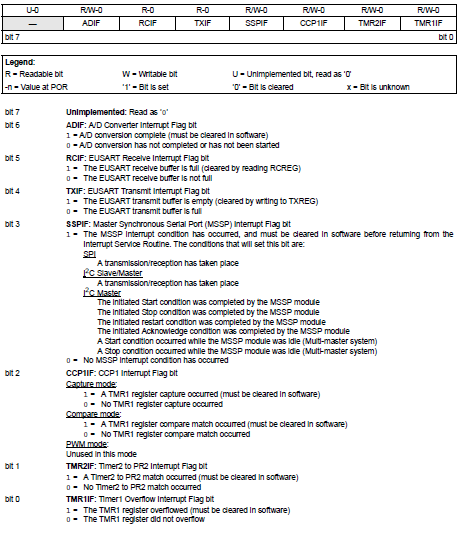


**PIE2: Peripheral Interrupt Enable 2 Register**

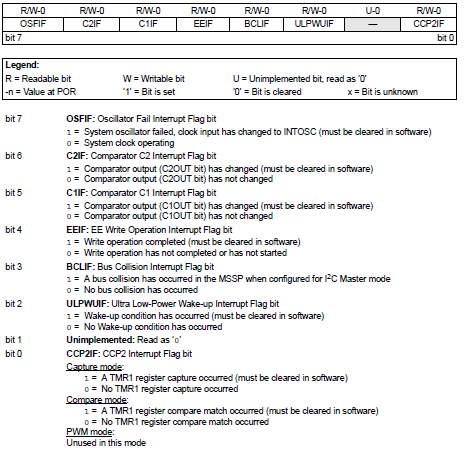
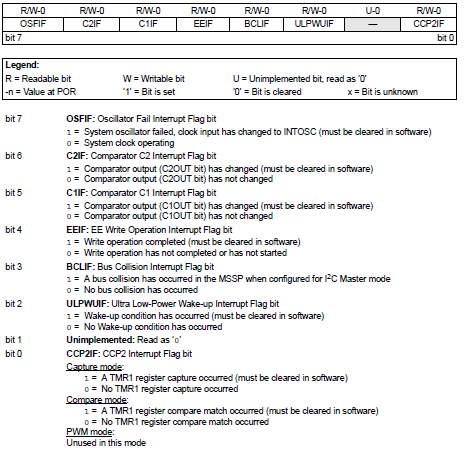
****

**PIR1: Peripheral Interrupt Register 1**

****

****

**PIR2: Peripheral Interrupt Register 2**

****

When an interrupt is serviced:

* The GIE is cleared to disable any further interrupt.
* The return address is pushed onto the stack.
* The PC is loaded with 0004h.
* For external interrupt events, such as the INT pin, PORTB change interrupts, the interrupt latency is three or four instruction cycles. The exact latency depends upon when the interrupt event occurs

### 5.4.6 Programming Interrupts

# Projects Undertaken

|  |  |  |
| --- | --- | --- |
| S.NO | Name | Modules |
| 1 | Automatic Irrigation System | LCD, Moisture sensor, Buzzer, Keypad matrix |
| 2 | 8 player quiz | Push button, buzzer, LED |
| 3 | Patient system | RF pair, LCD, LM35 |
| 4 | Light intensity controller | LED strip, LDR |
| 5 | Obstacle robot | Ultrasonic sensor,L293D, motors |
| 6 | Bidirectional counter | IR sensor, SSD |
| 7 | Water Level indicator | RF pair |
| 8 | Car parking system | IR pair, LCD |
| 9 | Ludo | USART, LED matrix, LCD |
| 10 | Bluetooth controlled car | Bluetooth module, L293D, motor |
| 11 | Rock paper scissor | Push button, LED |
| 12 | RFID EVM | RFID module, EEPROM |
| 13 | RFID attendance system | RFID module, EEPROM |
| 14 | Water level controller | Ultrasonic module, Keypad matrix, LCD |
| 15 | Light intensity controller | LED, LDR |
| 16 | Password based door locking | LCD, Keypad matrix, L293D, motor, SSD, LM35 |
| 17 | Fire extinguishing robot | LDR, L293D, motor |