DIGITAL NOTES

ON

EMBEDDED SYSTEMS & DESIGN

(R20A0903)

For

CSE(IOT)

Prepared by

Mr.M.Krishna Chaithanya
Assistant Professor, Dept. of ECE

B.TECH III YEAR – I SEM (R20)

(2022-2023)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

MALLAREDDYCOLLEGEOFENGINEERING&TECHNOLOGY

B. TECH - III- YEAR I-SEM-IOTLT/P/DC3-/-/-3

EMBEDDEDSYSTEMS AND DESIGN (R20A0903)

COURSEOBJECTIVES:

Fo rembeddedsystems ,the course will enablethestudentsto:

1. To understand micro controllers architecture and it functionalities
2. Understand the core of an embedded system
3. To learn the embedded firmware design and development
4. To understands the embedded programming concepts

UNIT-I:

INTRODUCTION TO MICROCONTROLLERS:

8051 Microcontroller:

Overview of 8051 Microcontroller, 8051 Architecture, Pin diagram, Memory Organization, Addressing Modes, and Instruction setof8051.

Arduino: Overview of Arduino, Introduction to ATMEGEA 328P, Arduino board. Introduction Arduino Programming: Setup (), loop (), Digital Read (), Digital Write () AnalogRead(), AnalogWrite().

UNIT-II: INTRODUCTION TO EMBEDDED SYSTEMS:

Definition of embedded systems, Embedded systems Vs General computing systems, History of embedded systems, Classification, Major application Areas, Purpose of embedded systems, characteristic sand quality attributes of embedded systems.

UNIT-III: TYPICAL EMBEDDED SYSTEM: Core of the embedded system, Sensors and actuators, Onboard communication interfaces-I2C,SPI,parallel interface;External communication interfaces-RS232,USB,infrared,Bluetooth,Wi-Fi,ZigBee,GPRS.

UNIT-IV: EMBEDDED FIRMWAREDESIGN AND DEVELOPMENT:

Embedded firmware design approaches -superloop-basedapproach,operating system-based approach; embedded firmware development languages -assembly language-based development, high level language-based development.

UNIT-V EMBEDDED PROGRAMMING:

Assembly language: Interfacing LED, LCD and keypad to 8051 Microcontroller.

Embedded C: Interfacing LED, RGB LED, LCD, Switch, Sensors, Buzzer to Arduino UNO, Serial communication programming with Arduino.

TEXTBOOKS:

1. Introduction to Embedded Systems-shibukv, McGraw Hill Education.
2. Kenneth. J.Ayala, The 8051 Microcontroller, 3rd Edition Cengage Learning

REFERENCEBOOKS:

1. The 8051 Microcontroller and Embedded Systems Second Edition Muhammad Ali Mazidi Janice GillispieMazidiRolin D. McKinlay
2. Embedded Systems- An integrated approach - Lyla B Das, Pearsoneducation2012.

COURSEOUTCOMES:

After going through this course, the student will be able to

1. The student will learn the internal organization of popular 8051 microcontrollers.
2. Understand the core of the embedded systems
3. Understand the internal and external communication interface
4. Understand Embedded Firmware design approaches
5. Understand embedded programming concepts

Unit-1

CONTENTS

INTRODUCTION TO MICROCONTROLLERS:

8051 Microcontroller:

- Overview of 8051 Microcontroller
- 8051 Architecture
- Pin diagram
- Memory Organization
- Addressing Modes
- Instruction setof8051.

Arduino:

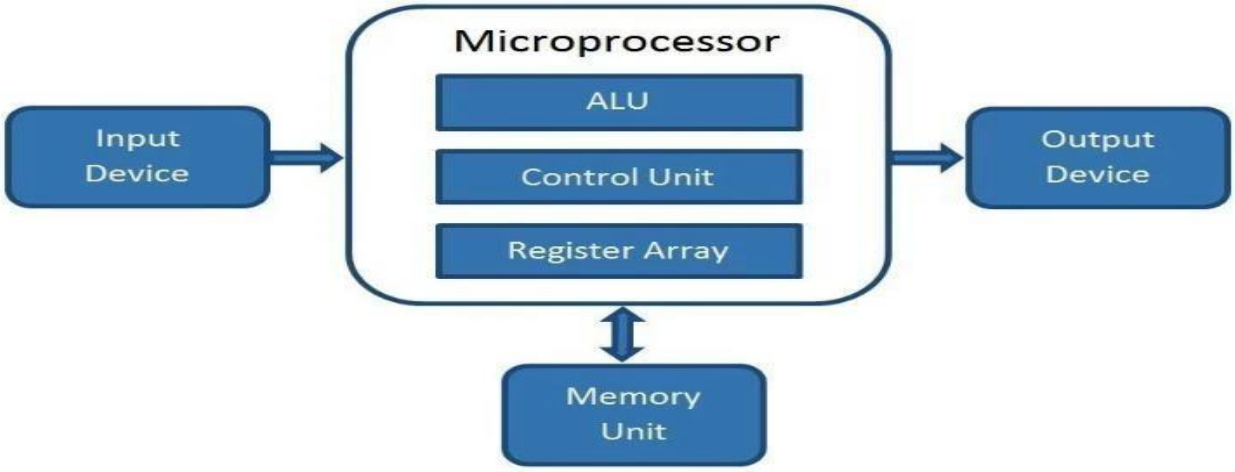
- Overview of Arduino.
- Introduction to ATMEGEA 328P.
- Arduinoboard:Introduction
- Arduino Programming:

Setup (), loop (), Digital Read (), Digital Write () AnalogRead (), Analog Write ().

Introduction

THE 8086 MICROPROCESSOR:

A microprocessor is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together. An Integrated circuit that contains all the functions of a central processing unit (CPU) of a computer is called Microprocessor.



FEATURES OF 8086 Microprocessor:

- The 8086 microprocessor is a 16-bit microprocessor. What this means is that the ALU and the internal registers work with 16 bit of binary data at a time.
- It has 16 bits of the data bus. Because of this, the 8086 can read or write either 16 bit or 8 bit of data at a time.
- The 8086 microprocessor has 20 bits of address lines that can access 2^{20} address locations.
- Each memory location is a byte-addressable memory location.
  Therefore, the total amount of memory that the 8086 microprocessor contains = 2^{20} bytes = 1 MB
  Therefore, the 8086 can access up to 1 MB of memory.
- It works in the frequency range of about 5-10 MHz
- It can prefetch up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.
- A 40 pin dual in line package.
- It has 256 vectored interrupts
- It consists of 29,000 transistors.
- 8086 is designed to operate in two modes, Minimum mode and Maximum mode.
  - The minimum mode is selected by applying logic 1 to the MN /MX input pin.
  - This is a single microprocessor configuration.
  - The maximum mode is selected by applying logic 0 to the MN / MX input pin.
  - This is a multi micro processors configuration.

Components of 8086 microprocessor

The 8086 microprocessor consists of two main blocks:

- Bus Interface Unit (BIU)
- Execution Unit (EU)

The Bus Interface Unit (BIU) consists of the following components:

- Instruction Queue
- Segment Registers
- Instruction Pointer (IP)

The Execution Unit (EU) consists the following:

- Arithmetic Logic Unit (ALU)
- Control Unit
- General Purpose registers
- Index registers and pointers (except IP)
- Flags and Operands

8051 Microcontroller: Introduction

- A microcontroller is a programmable integrated circuit (IC) that consists of a small CPU, RAM and I/O pins. Microcontroller units (MCUs) are widely used in many devices.
- A microcontroller is a highly integrated single chip, which consists of on chip CPU (Central Processing Unit), RAM (Random Access Memory), EPROM/PROM/ROM (Erasable Programmable Read Only Memory), I/O (input/output) Ports – serial and parallel, timers, interrupt controller etc..
- The 8051 microcontroller is a very popular 8-bit microcontroller introduced by Intel in the year 1981
- 8051 is one of the first most popular microcontroller also known as MCS-51
- 8051 works at +5 Volts DC.
- It is available as a 40-pin DIP chip
- It is an 8-bit microcontroller which means data bus is of 8-bits. Therefore, it can process 8-

bits at a time. It is used in wide variety of embedded systems like robotics, remote controls, automotive industry, telecom applications, power tools etc.

- Initially it came out as N-type metal-oxide-semiconductor (NMOS) based microcontroller, but later versions were based on complementary metal-oxide-semiconductor(CMOS) technology.
- These microcontrollers were named as 80C51, where C in the name tells that it is based on CMOS technology.

Development/Classification of microcontrollers

Development of some popular microcontrollers is given as follows.

Intel 4004	4 bit (2300 PMOS trans, 108 kHz)	1971
Intel 8048	8 bit	1976
Intel 8031	8 bit (ROM-less)	.
Intel 8051	8 bit (Mask ROM)	1980
Microchip PIC16C64	8 bit	1985
Motorola 68HC11	8 bit (on chip ADC)	.
Intel 80C196	16 bit	1982
Atmel AT89C51	8 bit (Flash memory)	.
Microchip PIC 16F877	8 bit (Flash memory + ADC)	.

Difference betweenMicroprocessorandMicrocontroller:

S.No	Microprocessor	Microcontroller
1	Microprocessor is the heart of Computer system.	Micro Controller is theheart of an embedded system.
2	It is only a processor, so memory and I/O components need to be connected externally	Micro Controller has a processor along with internal memory and I/O components.
3	Memory and I/O has to be connected externally, so the ciruit becomes large.	Memory and I/O are already present, and the internal circuit is small
4	You can't use it in compact systems	You can use it in compact systems
5	It is mainly used in personal computers	It is used mainly in a washing machine, MP3 players, and embedded systems
6	Cost of the entire system is high	Cost of the entire system is low
7	Due to external components, the total power consumption is high. Therefore, it is not ideal for the devices running on stored power like batteries.	As external components are low, total power consumption is less. So it can be used with devices running on stored power like batteries.
8	Most of the microprocessors do not have power saving features.	Most of the microcontrollers offer power-saving mode.
9	Microprocessor has a smaller number of registers, so more operations are memory- based	Microcontroller has more register. Hence the programs are easier to write
10	It's complex and expensive, with a large number of instructions to process.	It's simple and inexpensive with less number of instructions to process.

Features of 8051Microcontroller:

- Eight-bit CPU with registers A (the accumulator) and B
- 8-bit data bus and 16-bit address bus
- Eight-bit program status word (PSW)
- Eight-bit stack pointer (SP)
- Four register banks, each containing eight registers
- 4 KB on chip program memory (ROM or EPROM)).
- 128 bytes on chip data memory(RAM).
- Two -16 bit timers T_0 and T_1
- Five Interrupts (3 internal and 2 external).
- Four Parallel ports each of 8-bits (PORT0, PORT1,PORT2,PORT3) with a total of 32 I/O lines.
- One 16-bit program counter and One 16-bit DPTR (data pointer).

Overview of the 8051 Family

8051 microcontroller was initially designed by Intel Corporation in 1981. Features of 8051 made it extremely popular in market. Because of it's popularity and high demand Intel allowed other manufacturers to fabricate and market different variants of 8051 with a condition that all these variants should be code compatible with 8051.

This resulted in a lot of variants of 8051 in market, among which 8052 and 8031 are the most popular ones. Therefore, 8052 and 8031 are considered as the family members of 8051.

8052

8052 is the super set of 8051 as it has all the features of 8051 with an extra timer and an extra RAM of 128 bytes. Therefore, 8052 has a total of 256 bytes of RAM and 3 timers in all. Also all the programs written for 8051 will run on 8052 as 8052 is super set of 8051, but it's reverse is not true.

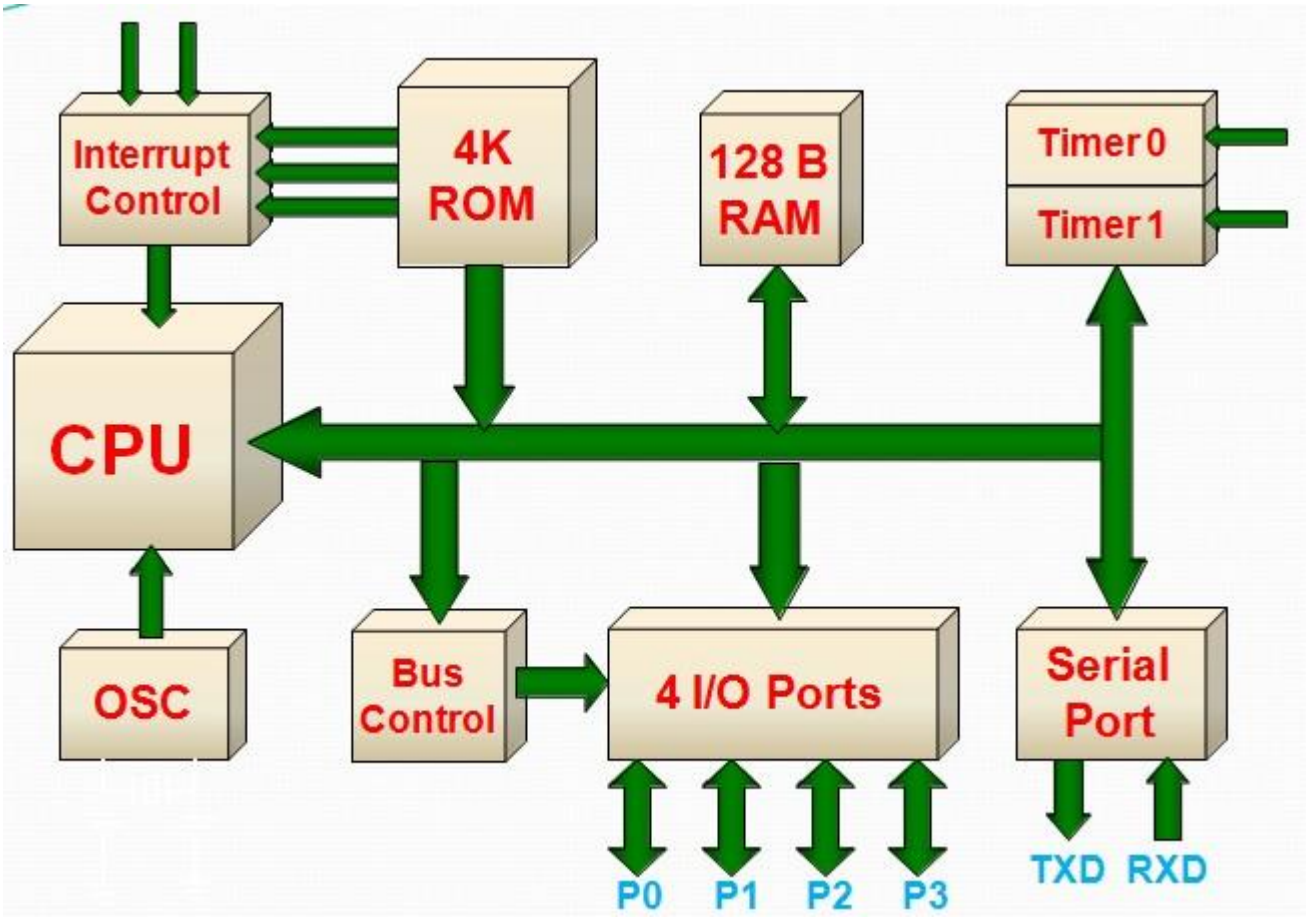
Characteristic	8051	8052	8031
RAM	128 bytes	256 bytes	128 bytes
ROM (on-chip)	4 KB	4 KB	0 KB
Number of Timer	2	3	2
Interrupt Sources	6	8	6
Serial Port	1	1	1

Number of I/O Ports	**32**	**32**	**32**

8031

8031 is referred to as ROM-less microcontroller chip because it has 0 K byes of on-chip ROM. For it's operation, 8031 requires external ROM which aids it in fetch and execute operations. Apart from this, it shares almost all the features of 8051

ARCHITECTURE& BLOCKDIAGRAM OF 8051 MICROCONTROLLER:



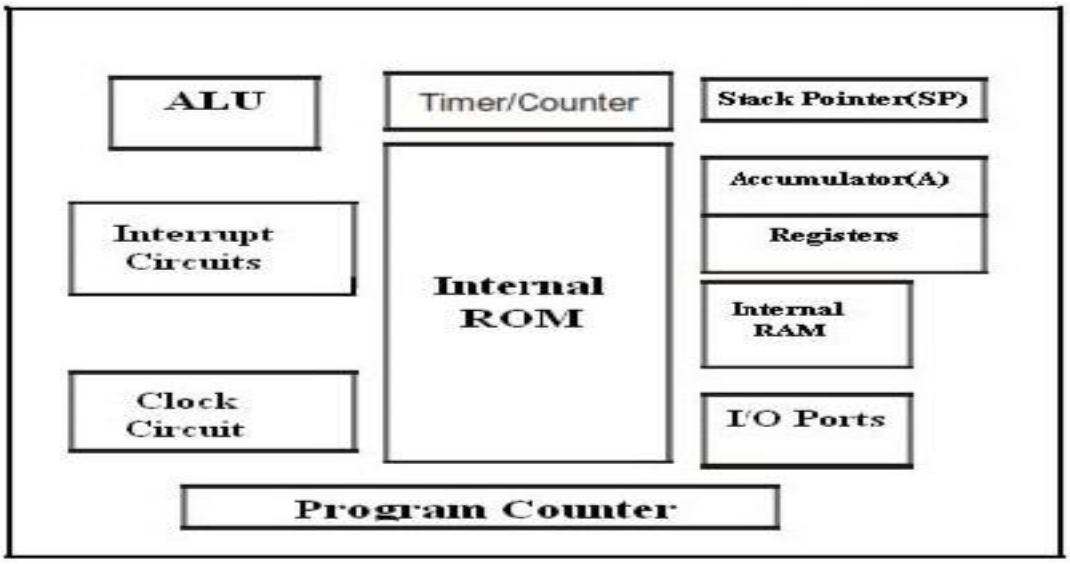
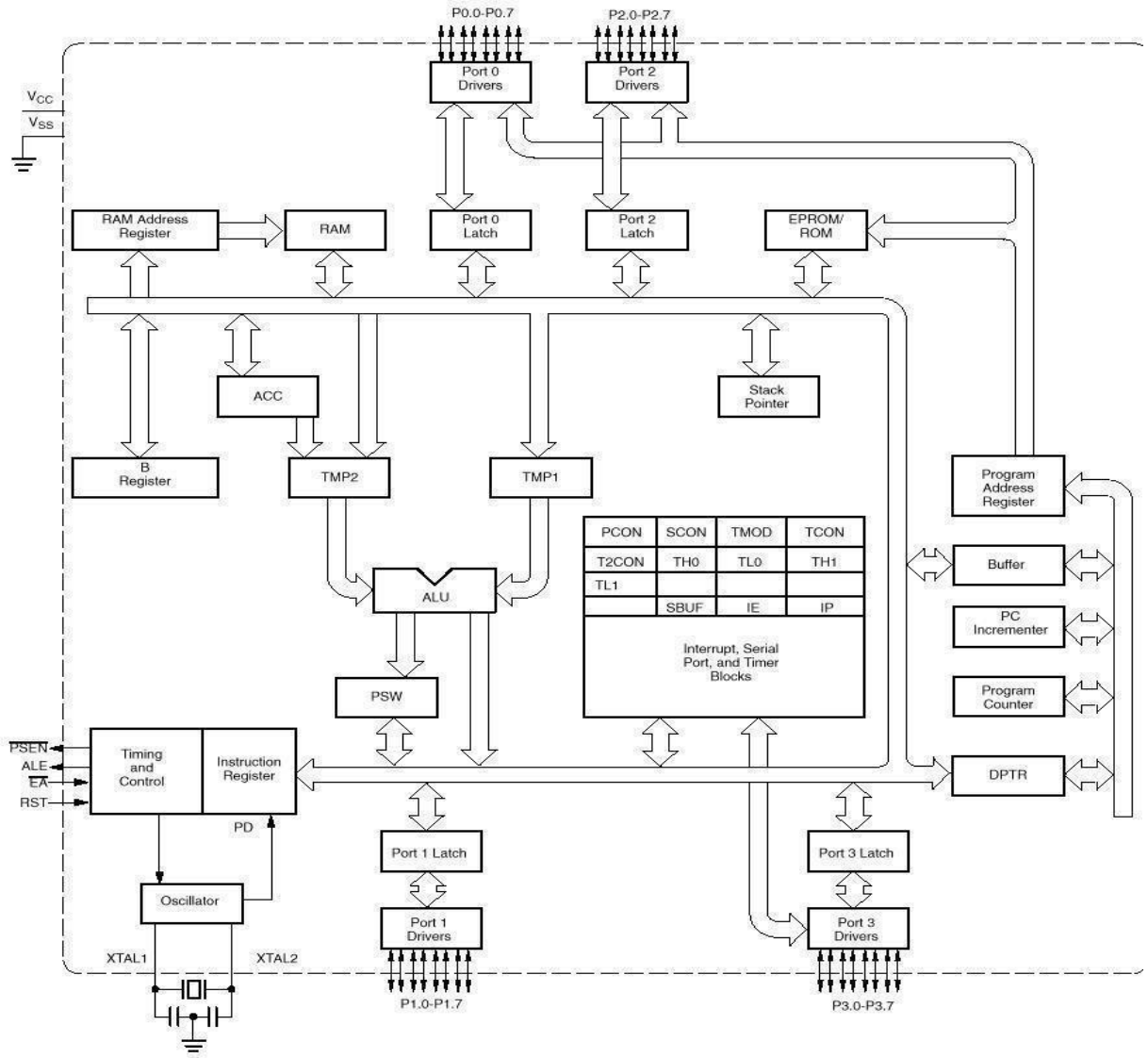


Fig.3.BlockDiagramof8051 Microcontroller.



8051 Microcontroller consists of

- an 8-bit ALU&CPU(Central Processing Unit)
- Oscillator
- Interrupt Control
- Bus Control
- one 8-bit PSW(Program Status Word Register),
- A and B registers ,
- one 16-bit Program counter ,
- one 16-bit Data pointer register(DPTR),
- 128 bytes of RAM and 4kB of ROM and

- four parallel I/O ports each of 8-bit width.

1. CPU(Central Processing Unit):

The CPU is the brain of any processing machine. It is the part that is responsible for managing all the tasks of the microcontroller. The CPU is an independent unit.

Users can not interfere with the CPU controlling how it should function. It identifies the tasks present in the ROM and then processes them.

In the 8051 microcontroller architecture, the CPU is responsible for managing registers. Registers are a type of memory in a computer. They can store and manipulate data.

2. Oscillator

- Generally, we know that the microcontroller is a device, therefore it requires clock pulses for its operation of microcontroller applications.
- For this purpose, microcontroller 8051 has an on-chip oscillator which works as a clock source for Central Processing Unit of the microcontroller. The output pulses of oscillator are stable.
- Therefore, it enables synchronized work of all parts of the 8051 Microcontroller.

3. Interrupts

- In the 8051 microcontroller architecture, interrupts stop the microcontroller's current task. Interrupts are caused when some other program has a higher priority request for execution.
- When an interrupt occurs, the ongoing task stops, the sub-routine for the interrupt is executed, and then the previous job resumes
- 8051 has 5 interrupt sources, out of which two are peripheral Interrupts, two are timer interrupts and one is a serial port interrupt.

INT0	It is an external interrupt having code 0. External hardware causes this interrupt.
INT1	It is an external interrupt having code 1. External hardware causes this interrupt.
TF0	This is for the timer 0 overflow interrupt.
TF1	This is for timer 1, overflow interrupt.
RI/TI	This is a serial communication interrupt.

4. Memory

The next part of the 8051 microcontroller architecture is the memory. For any data manipulation to occur, we require a set of instructions.

- A program is a set of commands, which is needed by the Microcontroller to perform a particular task. These programs need a storage space where they can be stored and later the microcontroller interpret them and perform the specific task.
- 8051 microcontroller architecture include two types of memory, such as : program memory and data memory.

- The instructions of the CPU are stored in the Program Memory. It is usually implemented as Read-Only Memory or ROM, where the Program written into it will be retained even when the power is down or the system is reset.
- Data Memory in a Microcontroller is responsible for storing values of variables, temporary data, intermediate results, and other data for the proper operation of the program.
- In the 8051 microcontroller architecture, the microcontroller has 4KB ROM and 128 bytes of RAM.

5. Bus

A bus is a group of wires. Communication within the microcontroller happens through this bus. There are either 8 or 16 or more wires in the bus. If the 8051 microcontroller architecture has 8 wires, it can carry 8 bits of data. If the 8051 microcontroller architecture has 16 wires, it can carry16 bits of data. The bus further falls into two categories:

Address Bus: The address bus in the 8051 microcontroller architecture is of 16 bits. This bus transfers data from the CPU to the memory. There are different addressing modes in this bus:

- Immediate addressing modes.
- Register addressing mode.
- Direct Addressing mode.
- Register indirect addressing mode.

Data Bus: In 8051 microcontroller architecture, the data bus is 8 bits. It helps in carrying the data from one place to another.

6. Input/output Ports

- A microcontroller controls small operations for a system. It is embedded in the systems. We might sometimes need to connect the microcontroller to other devices. In the 8051 microcontroller architecture, we have 4 input/output ports. We connect other input/output peripherals using these ports.
- All the 4 ports function bidirectionally i.e., either input or output according to the software control.

7. Timers/Counters

In the 8051 microcontroller architecture, we have two timers. They are each 16 bits. We have the timers to generate gaps between two events. The two timers generate two delays(gaps), and the suitable one is chosen.

- The timer produces the delay according to the demand of the processor and sends the signal to the processor once the respective delay gets produced.
- Microcontroller 8051 is incorporated with two 16 bit counters & timers. The counters are separated into 8-bit registers. The timers are utilized for measuring the intervals, to find out pulse width**, etc.**

The microcontroller also includes a program counter, data pointer, stack & stack pointer, instruction registers including latches, temporary registers & buffers for the I/O ports.

8. A and B Registers: The A and B registers are special function registers which hold the results of many arithmetic and logical operations of 8051

A register is also called the Accumulator and as its name suggests, is used as a general register to accumulate the results of a large number of instructions

9. The R registers: The "R" registers are a set of eight registers that are named R0, R1, etc. up to and including R7.

These registers are used as auxiliary registers in many operations.

The "R" registers are also used to temporarily store values.

10. STACK in 8051 Microcontroller

- The stack is a part of RAM used by the CPU to store information temporarily. This information may be either data or an address
- In 8051 the RAM locations 08 to 1F (24 bytes) can be used for the Stack
- The register used to access the stack is called the Stack pointer which is an 8-bit register.
- There are two important instructions to handle this stack. One is the PUSH and the other is the POP.
- The loading of data from CPU registers to the stack is done by PUSH and the loading of the contents of the stack back into a CPU register is done by POP.

11. Stack Pointer (SP)

Stack Pointer (SP) – it contains the address of the data item on the top of the stack. Stack may reside anywhere on the internal RAM. On reset, SP is initialized to 07 so that the default stack will start from address 08 onwards.

12. Data Pointer Register (DPTR) :

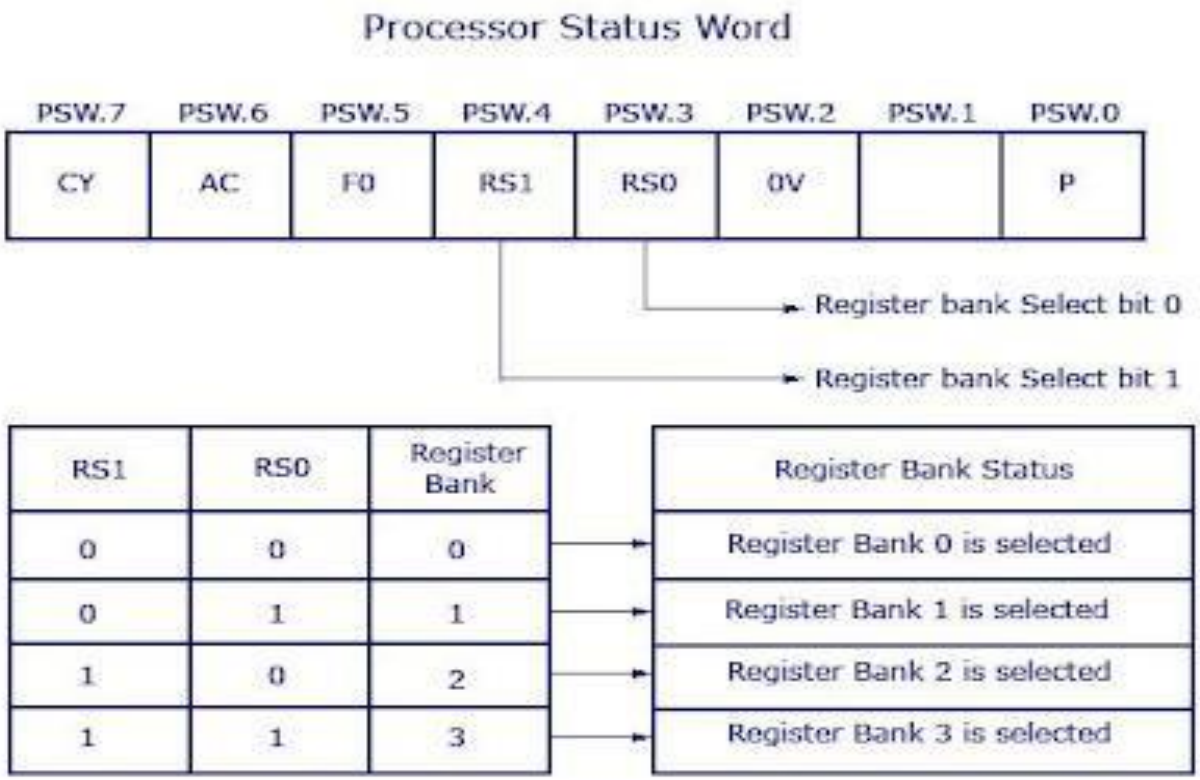
- It is a 16-bit register which is the only user-accessible.
- DPTR, as the name suggests, is used to point to data.
- It is used by a number of commands which allow the 8051 to access external memory.
- When the 8051 accesses external memory it will access external memory at the address indicated by DPTR.
- This DPTR can also be used as two 8-registers DPH and DPL
- Data Pointer (DPTR) – DPH (Data pointer higher byte), DPL (Data pointer lower byte).

13. PSW (Program Status Word)

- The program status word (PSW) register is an 8-bit register. It is also referred to as the *flag register*. Although the PSW register is 8 bits wide, only *6* bits of it are used by the 8051.The two unused bits are user-definable flags.
- Four of the flags are called *conditional flags*, meaning that they indicate some conditions that result after an instruction is executed. These four are CY (carry), AC (auxiliary carry), P (parity), and OV (overflow).
- As seen from below figure, the bits PSW.3 and PSW.4 are designated as RS0 and RS1 as register selection bit, respectively, and are used to change the bank registers.

The PSW.5 and PSW.l bits are general-purpose status flag bits and can be used by the programmer for any purpose. In other words, they are user definable. See below Figure for the bits of the PSW

register



CY, the carry flag

- This flag is set whenever there is a carry out from the D7 bit.
- This flag bit is affected after an 8-bit addition or subtraction.
- It can also be set to 1 or 0 directly by an instruction such as "SETB C" and "CLR C" where "SETB C" stands for "set bit carry" and "CLR C" for "clear carry".

AC, the auxiliary carry flag

- This flag is used by instructions that perform BCD (binary coded decimal) arithmetic.

P, the parity flag

The parity flag reflects the number of 1 s in the A (accumulator) register only.

- If the A register contains an odd number of Is, then P = 1. Therefore, P = 0 if A has an even number of 1s.

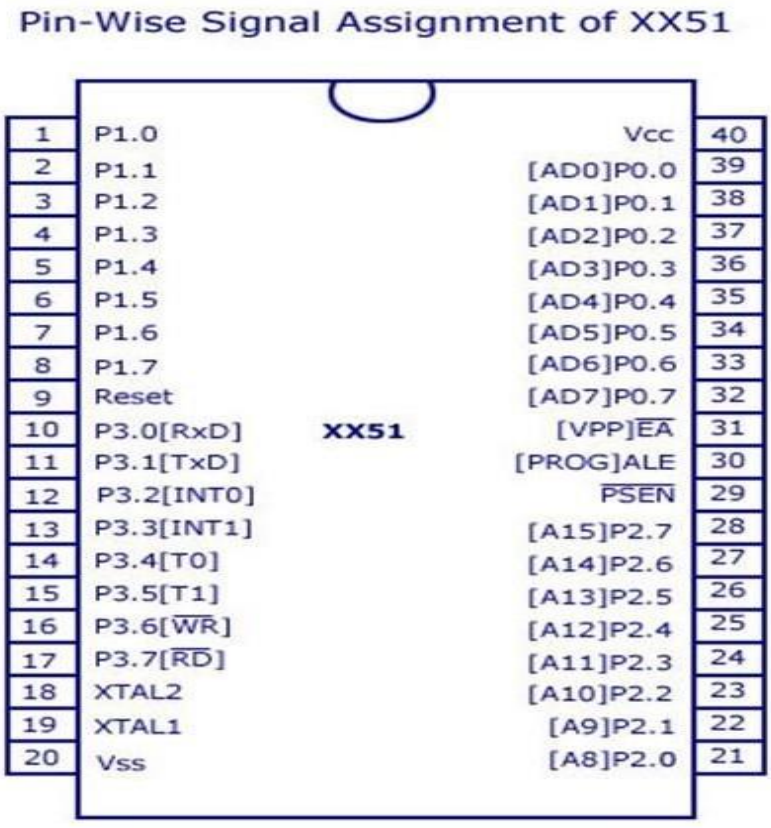
OV, the overflow flag

- This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
- In general, the carry flag is used to detect errors in unsigned arithmetic operations.
- The overflow flag is only used to detect errors in signed arithmetic operations.

The selection of the register Banks and their addresses are given below.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Pin Description of 8051 Microcontroller



Pins from 1-8

Port 1: The pins in this port are bi-directional and can be used for input and output. The pins are individually controlled; some are used for input while others are used for output purposes.

Pin 9

This pin is also called ‘Reset Pin’. This is used for resetting the microcontroller to its initial values. If the pin is set at logic 0, the chip runs normally. When the oscillator is running, setting the pin at logic 1 for more than two machine cycles will reset the microcontroller.

Pins from 10-17

PORT 3: These pins are same as pins in port 1 because of their bidirectional input port.

- Pin 10 and Pin 11 perform receiving and transmitting operation of serial data using ‘RS-232’ protocol.
- Pin 12 and Pin 13 are used to interrupt inputs.
- Pin 14 and Pin 15 perform alternative functions linked with Timer 0 and Timer 1.
- Pin 16 and Pin 17 are used when working with external memory.

Pins from 18-19

The pins are used for connecting an external crystal oscillator module with the microcontroller.

Pin 20

Also called Vss. This is the ground pin that represents 0V.

Pins from 21-28

PORT 2: These are another set of bidirectional input port, they are used when processing external memory. Higher order address bus signals are multiplexed with this quasi-bidirectional port.

Pin 29

Also called PSEN (Program Store Enable) it controls and manages the access to external CODE memory.

Pin 30

Named as Address Latch Enable (ALE). It is used when working with external memory. ALE activity is disabled in some devices where external memory is not used. Thus helping in reducing the electromagnetic interference generated by the product.

Pin 31

Named as External Access (EA). In order to execute code from internal memory this pin is connected to Vcc. To execute code from external memory the pin must be grounded.

Pins from 32-39

Port 0: These are set of another bidirectional input port. Unlike Port 1, Port2 and Port 3 this port lacks pull-up resistors.

Pin 40

Named as Vcc. This is usually a 5V pin on 5V devices and 3V pin on 3V devices.

I/O Pins

The 8051 microcontrollers are mostly 8-bit ports, thus giving a total of 32 pins which you can use to read input and control output. All of them are bidirectional in nature so they can perform as both input and output. Some of the ports perform alternate functions as discussed above such as to support access to external memory. This is done to reduce the size of the microcontroller device. When these ports are busy in performing their alternate functions, they can not be made toact as input-output ports.

Memory organization :

The 8051 microcontroller has 128 bytes of Internal RAMand 4kB of on chip ROM .The RAM is also known as Data memory and the ROM isknownasprogrammemory.

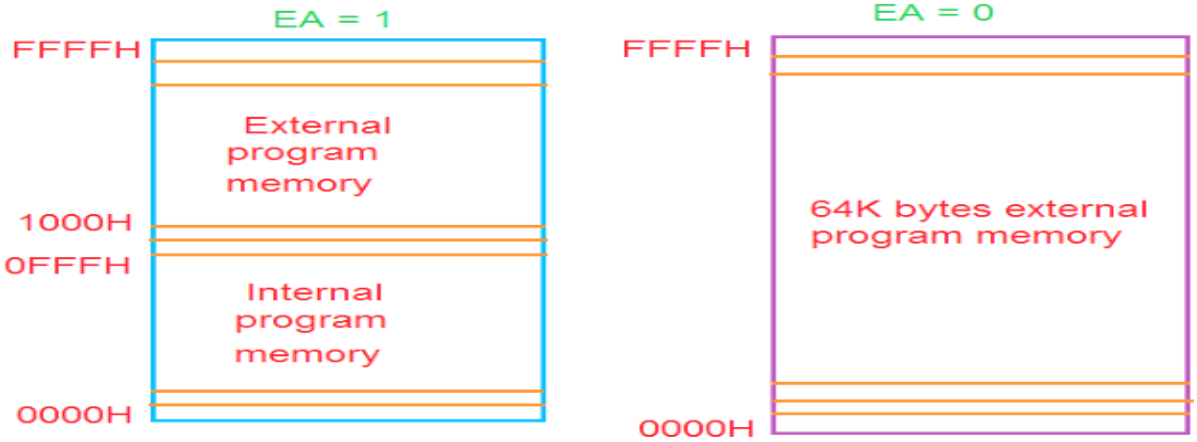
Program memory organization in 8051:

The Program Memory or ROM is a type of non-volatile memory used in microcontrollers where the code or the program to be executed is stored using the program counter (PC), like tables or initialization program.

Intel 8051 has an internal/ built-in ROM of 4KB and can be extended up to 64KB by using an external program memory. The program memory allocation can be done in two ways depending on the status of the EA pin (External Access Pin), which is an active low pin (i.e., activates when low-signal is provided).

Program memory:

Program memory accessed through EA pin. In program memory two categories takes place:



a)If EA is high, internal program memory is accessed to 0FFFH memory location and external program memory accessed from 1000H to FFFFH memory locations.

b)If EA is low, only external program memory accessed from 0000H to FFFFH memory locations.

Data memory organization in 8051:

The Data Memory or RAM is a volatile memory since cutting off power to the IC will result in loss of information or data. RAM is used for temporarily storing the data and the auxiliary results generated during the runtime.

Older version of 8051 used to consist built-in 256 bytes of RAM now, and it consists of an additional 128 bytes, which are accessed by indirect addressing.

The data memory in 8051 is divided into three parts:

1. Lower 128 bytes (00H – 7FH), which are addressed b either Direct or Indirect addressing. Further, the Lower 128 bytes are divided into three parts,

- Register Banks (Bank 0,1,2,3) from 00H to 1FH – 32 bytes
- Bit Addressable Area from 20H to 2FH – 16 bytes
- General Purpose Register (Scratch Pad Area) from 30H to 7FH – 80 bytes

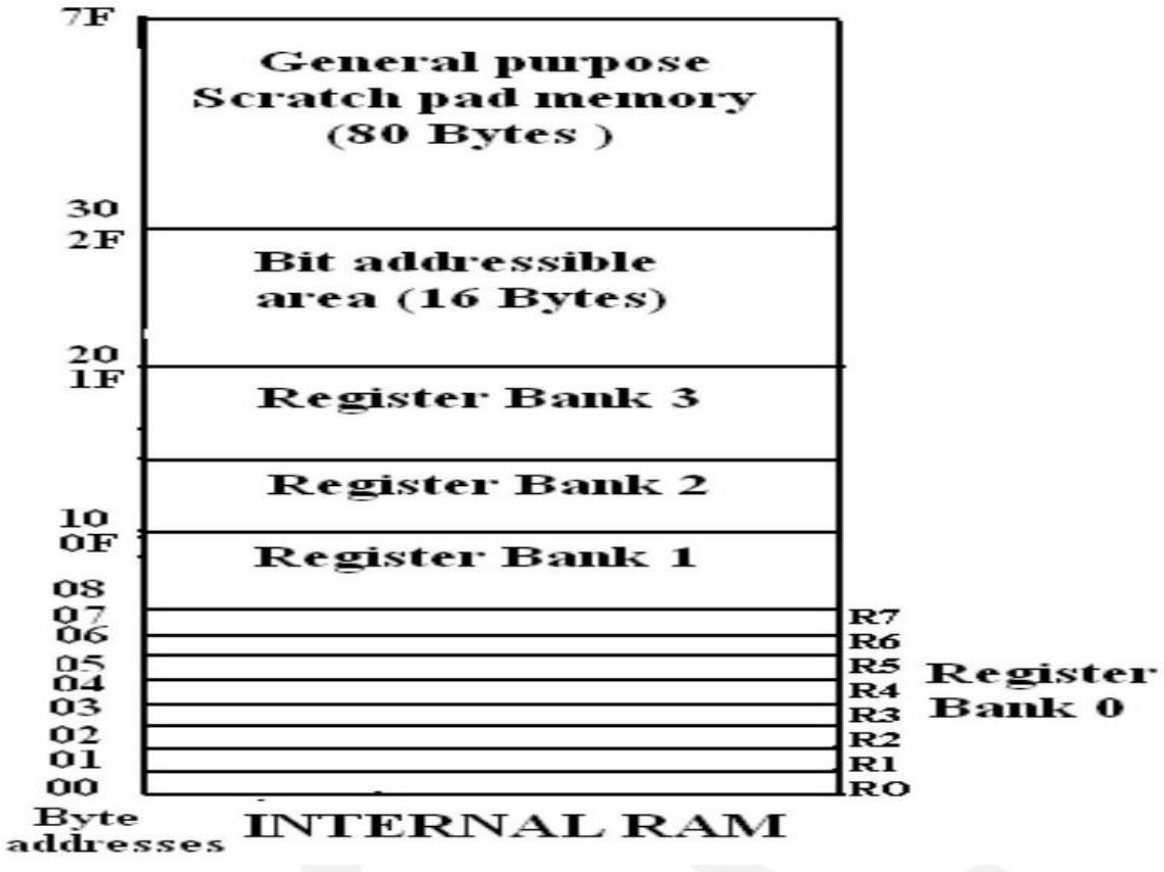
2. Upper 128 bytes (80H – 0FFH) for the Special Function Register (SFRs) which includes I/O ports (P0, P1, P2, P3), Accumulator (A), Timers(THx, TLx, TMOD, TCON, PCON), Interrupts(IE, IP), Serial Communication controls(SBUF, SCON), Program Status Word (PSW). These are addressed using Direct addressing.

3. 128 bytes of Additional Memory

Register Banks:

The 8051 has four Register Banks with each bank having 8 “R” registers, which are used in many of its instructions. These R registers are numbered from 0 through 7 (R0, R1, R2, R3, R4, R5, R6,

and R7). These registers are generally used to assist in manipulating values and moving data from one memory location to another. For example, to add three numbers say, 0AH, 0BH, 0CH, we would execute the following instruction:



These banks are accessed one at a time by bit addressing of D3 and D4 bits of PSW (Program Status Word) SFR by changing the values of RS0 and RS1 by SETB, and CLR commands. For example, “SETB PSW.3” will set PSW.3 = 1, and Bank 1 register is selected. By default, Bank 0 is selected.

RS1 (PSW.4)	RS2 (PSW.3)	Bank Selected	Memory Location
0	0	Bank 0	00 – 07
0	1	Bank 1	08 – 0F
1	0	Bank 2	10 – 17
1	1	Bank 3	18 – 1F

- 16 bytes of bit addressable areaand
- 80 bytes of general purpose area (Scratch pad memory) as shown in the diagram below. This area is also utilized by the microcontroller as a storage area for the operating stack.
- The registers are named as R0-R7 .Each register can be addressed by its name or by its RAM address.

ForEx :MOV A, R7orMOV R7,#05H

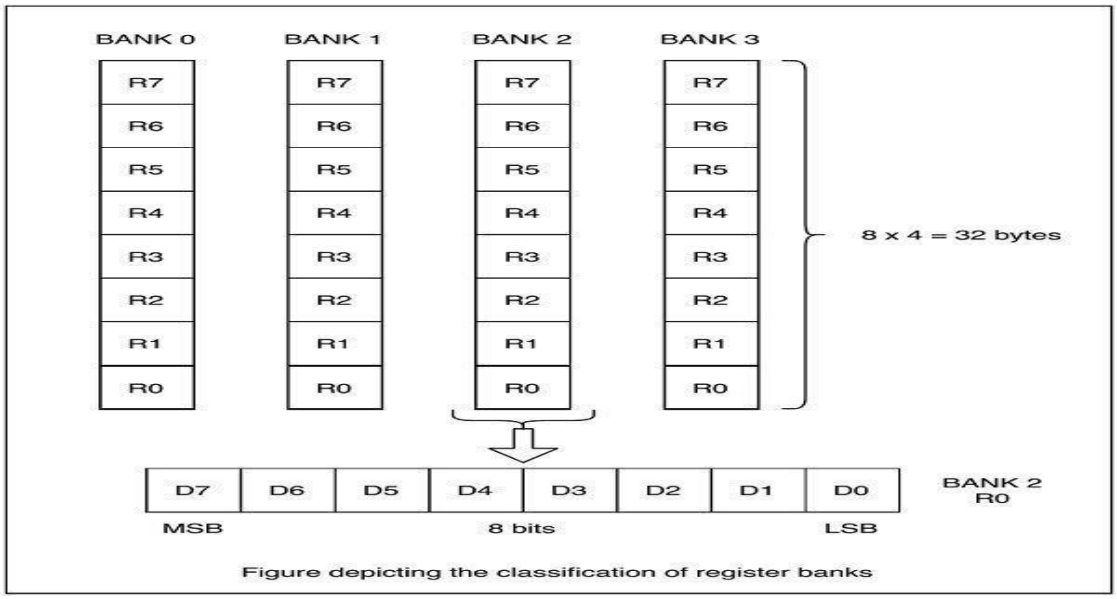


Figure depicting the classification of register banks

Bit addressable memory area

16 bytes of RAM is allotted as a bit addressable memory. It consists of 16 x 8 = 128 bits. These 128 bits can be individually bit-addressed, starting from 00H to 7FH or can be byte-addressable for 20H to 2FH.

Bit addressing is very useful when we want to monitor the status of a particular pin of any port.

General-purpose area/Scratchpad area

The rest of the RAM from 30H to 7FH, these 32 bytes can be addressed directly or indirectly using direct or indirect addressing modes, respectively. Therefore it's also called Scratch Pad Memory. You may read or write a full byte (8-bit) data at these locations.

8051 Microcontroller: Addressing Modes

Addressing:-The CPU can access data in a register or in memory or be provided as an immediate value. These various ways of accessing data are called addressing modes.

The 8051 provides a total of five distinct addressing modes.

They are as follows:-

1) Immediate addressing modes.
2) Register addressing modes.
3) Direct addressing modes.
4) Register indirect - addressing mode.
5) Indexed addressing mode.

1) Immediate Addressing mode:-

In Immediate Addressing mode, the operand, which follows the Opcode, is a constant data of either 8 or 16 bits. The name Immediate Addressing came from the fact that the constant data to be stored in the memory immediately follows the Opcode.

The constant value to be stored is specified in the instruction itself rather than taking from a register. The destination register to which the constant data must be copied should be the same size as the operand mentioned in the instruction.

Example:-

```
MOV A, # 82H              ; load 82H into A
MOV R4, # 100             ; load the decimal value 100 in        to R4
MOV B, # 40H              ; load 40H in to B
MOV P1, # 55H             ; load 55H directly to port 1
MOV DPTR , # 4521 H       ; DPTR =4521 H
```

2) Register - Addressing Mode:-

In the 8051 Microcontroller Memory Organization Tutorial, we have seen the organization of RAM and four banks of Working Registers with eight Registers in each bank.

In Register Addressing mode, one of the eight registers (R0 – R7) is specified as Operand in the Instruction.

It is important to select the appropriate Bank with the help of PSW Register.

Example:-

```
MOV  A,RO          ;copy the contents of R0 in to A
MOV R2,A           ;copy the contents of A in to R2
ADD A,R5           ;Add contents of R5 to A
MOV  R7,DPL        ;load R7 with contents of DPL
```

3) Direct Addressing mode:-

In direct addressing mode the data is in a RAM Memory location whose addressing is known and this address is given as a part of the instruction.

Although the entire 128 bytes of RAM can be addressed using direct addressing mode It is most often used to access RAM locations 30-7FH

Example:-

```
MOV R0, 40H      ; save content of RAM location 40H in Ro
MOV 7FH, A       ; save content of A in to RAM location 7FH
MOV A,07H        ;same as move A1 R7 is copies data in R7 to A
MOV 03H, 04H          ; it copies data in R4 to R3
```

4) Indirect addressing mode:-

In the Indirect Addressing Mode or Register Indirect Addressing Mode, the address of the Operand is specified as the content of a Register. This will be clearer with an example.

Example: MOV A, @R1

The @ symbol indicates that the addressing mode is indirect. If the contents of R1 is 56H, for example, then the operand is in the internal RAM location 56H. If the contents of the RAM location 56H is 24H, then 24H is moved into accumulator.

Only R0 and R1 are allowed in Indirect Addressing Mode. These register in the indirect addressing mode are called as Pointer registers.

5) Indexed addressing mode:-

With Indexed Addressing Mode, the effective address of the Operand is the sum of a base register and an offset register. The Base Register can be either Data Pointer (DPTR) or Program Counter (PC) while the Offset register is the Accumulator (A).

In Indexed Addressing Mode, only MOVC and JMP instructions can be used. Indexed Addressing Mode is useful when retrieving data from look-up tables.

Example: MOVC A, @A+DPTR

Here, the address for the operand is the sum of contents of DPTR and Accumulator.

8051 Instruction Set:

Before seeing the types of instructions, let us see the structure of the 8051 Microcontroller Instruction. An 8051 Instruction consists of an Opcode (short of Operation – Code) followed by Operand(s) of size Zero Byte, One Byte or Two Bytes.

The Op-Code part of the instruction contains the Mnemonic, which specifies the type of operation to be performed. All Mnemonics or the Opcode part of the instruction are of One Byte size.

- No Operand
- Data value
- I/O Port
- Memory Location
- CPU register

There can multiple operands and the format of instruction is as follows:

MNEMONIC DESTINATION OPERAND, SOURCE OPERAND

A simple instruction consists of just the opcode. Other instructions may include one or more operands. Instruction can be one-byte instruction, which contains only opcode, or two-byte instructions, where the second byte is the operand or three byte instructions, where the operand makes up the second and third byte.

Based on the operation they perform, all the instructions in the 8051 Microcontroller Instruction Set are divided into five groups. They are:

- Data Transfer Instructions
- Arithmetic Instructions
- Logical Instructions
- Boolean or Bit Manipulation Instructions
- Program Branching Instructions

We will now see about these instructions briefly.

1. Arithmetic Instructions:

Using Arithmetic Instructions, you can perform addition, subtraction, multiplication and division. The arithmetic instructions also include increment by one, decrement by one and a special instruction called Decimal Adjust Accumulator.

Fo rexample:

ADDA, R1-Theresult of addition(A+R1) will be stored in the accumulator.

- ADD-Addition Operation
- ADDC-Addition with Carry
- SUB-Subtract operation
- SUBB-Subtract with Barrow
- MUL-Multiplication Operation
- DIV-Divison Operation

- INC -Incrementing by one
- DEC-Decreamenting by one

2. **Logical Instructions:** The Logical Instructions are used toperform logical operationslikeAND, OR, XOR, NOT, Rotate, Clear and Swap. Logical Instruction are performed on Bytes of data ona bit-by-bit basis. Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.
   - ANL-Logical AND Operation
   - ORL-Logical OR operation
   - XRL-Logical Exclusive Operation
   - CPL-Complement Operation
   - RL-Rotate Left Operation
   - RLC-Rotate Left Through Carry
   - RC-Rotate Right Operation
   - RRC-Rotate Right Through Carry
3. **Data Transfer Instructions:** Data transfer instructions move the content of one register toanother. The register the content of which is moved remains unchanged. If they have the suffix "X"(MOVX),thedatais exchangedwithexternal memory.

Data Transfer Instructions

Mnemonic	Description
MOV A, Rn	Move register to Accumulator
MOV A, direct	Move direct byte to Accumulator
MOV A, @Ri	Move indirect RAM to Accumulator
MOV A, #data	Move immediate data to Accumulator
MOV Rn, A	Move Accumulator to register
MOV Rn, direct	Move direct byte to register
MOV Rn, #data	Move immediate data to register
MOV direct, A	Move Accumulator to direct byte
MOV direct, Rn	Move register to direct byte
MOV direct, direct	Move direct byte to direct
MOV direct, @Ri	Move indirect RAM to direct byte

4. **Boolean Variable Instructions:** Boolean or Bit Manipulation Instructions will deal with bit

variables. Similar to logic instructions, bit oriented instructions perform logic operations. The difference is that these are performed upon single bits.

DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

8051 MICROCONTROLLER INSTRUCTION SET

5. Program Branching Instructions:

The last group of instructions in the 8051 Microcontroller Instruction Set is the Program Branching Instructions. These instructions control the flow of program logic. The mnemonics of the Program Branching Instructions are as follows.

Consider the below table lists the conditional jumps instruction used in 8051

Instructions	Action
JC	Jump if CY = 1
JNC	Jump if CY ≠ 1
JNB	Jump if bit = 0
JB	Jump if bit = 1
JZ	Jump if A = 0
DJNZ	Decrement and Jump if register ≠ 0

JNZ	Jump if A ≠ 0
CJNE A, data	Jump if A ≠ data
CJNE reg, #data	Jump if byte ≠ data
JBC	Jump if bit = 1 and clear bit

Arduino

Overview of Arduino:

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Board Types

Various kinds of Arduino boards are available depending on different microcontrollers used. However, all Arduino boards have one thing in common: they are programed through the Arduino IDE.

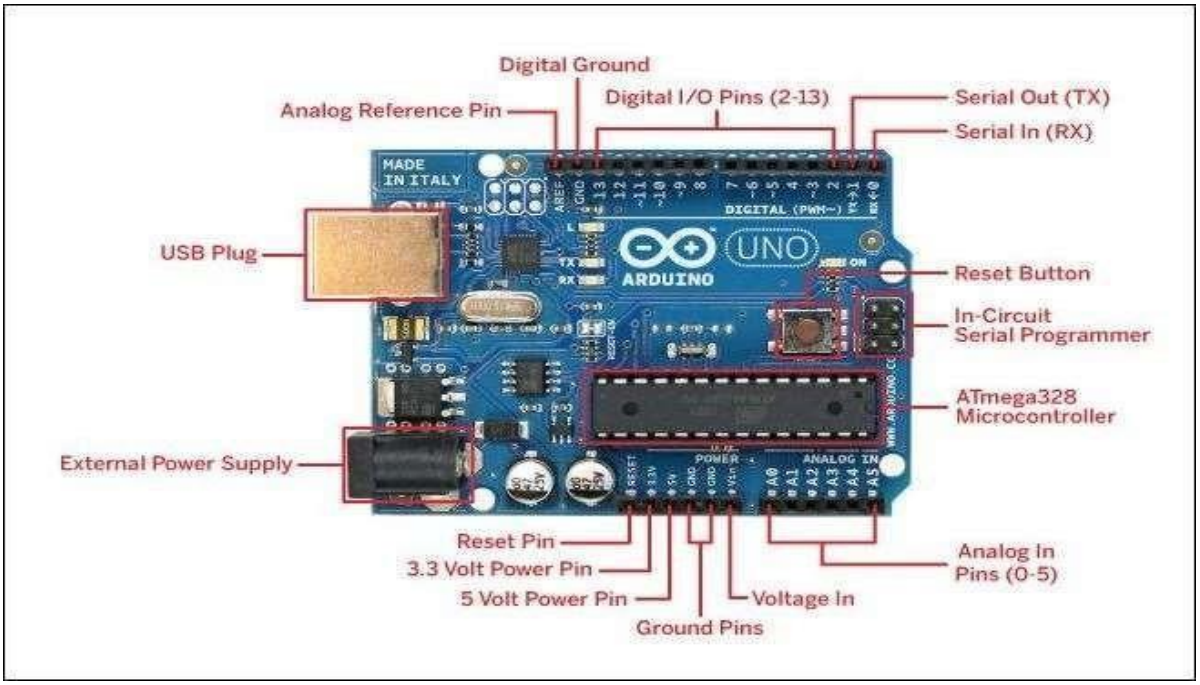
The differences are based on the number of inputs and outputs (the number of sensors, LEDs, and buttons you can use on a single board), speed, operating voltage, form factor etc. Some boards are designed to be embedded and have no programming interface (hardware), which you would need to buy separately. Some can run directly from a 3.7V battery, others need at least 5V.

Arduino UNO

Arduino UNO is a basic and inexpensive Arduino board and is the most popular of all the Arduino boards with a market share of over 50%. Arduino UNO is considered to be the best prototyping board for beginners in electronics and coding.

Arduino UNO comes with different features and capabilities. As mentioned earlier, the microcontroller used in UNO is ATmega328P, which is an 8-bit microcontroller based on the AVR architecture.

UNO has 14 digital input – output (I/O) pins which can be used as either input or output by connecting them with different external devices and components. Out of these 14 pins, 6 pins are capable of producing PWM signal. All the digital pins operate at 5V and can output a current of 20mA.

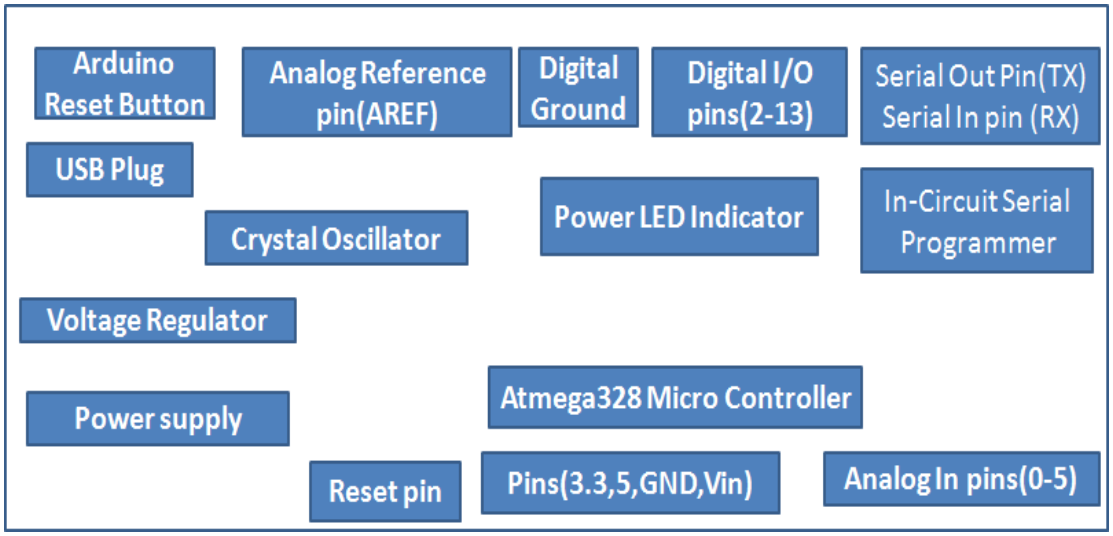


Arduino hardware is a PCB-mounted microcontroller that you can program and use for simple daily tasks, mathematical computations, and prototyping and testing. An **Arduino development board** consists of the core microcontroller with its supplementary components and the necessary circuitry to communicate with the PC which we will be using for both communications as well as programming the microcontroller.

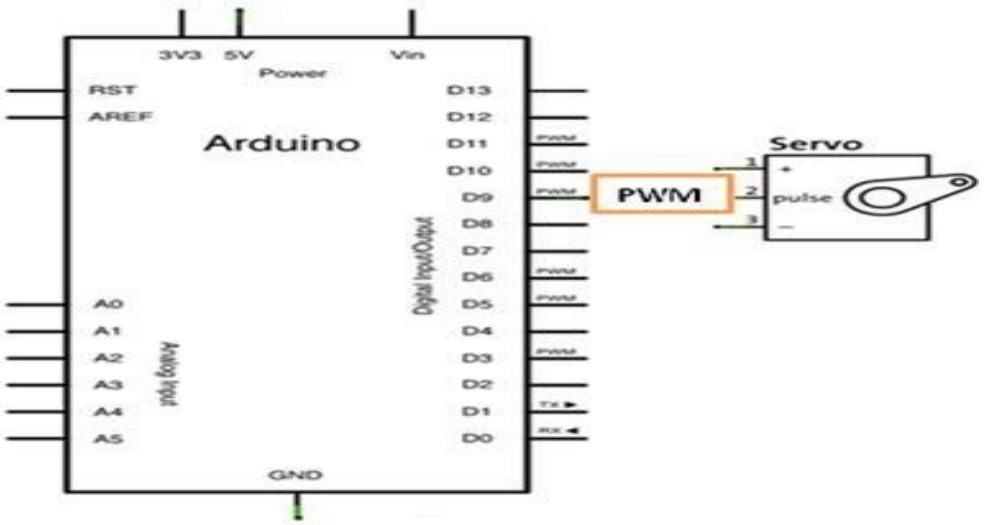
For communication or programming purposes, we will be using a **USB to TTL converter**, which will be embedded within the Arduino board.

Arduino block diagram shows the important modules on an Arduino UNO board.

In the UNO board, the main component is the **ATMega328P**. It is the heart of the Arduino UNO.



Arduino block diagram



Arduino UNO-Pin diagram

Arduino UNO BoardDescription:

Power USB

Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).

Power (Barrel Jack)

Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).

Voltage Regulator

The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.

Crystal Oscillator

The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.

Arduino Reset

You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labeled RESET (5).

Pins (3.3, 5, GND, Vin)

3.3V (6) − Supply 3.3 output volt

5V (7) − Supply 5 output volt

Most of the components used with Arduino board works fine with 3.3 volt and 5 volt.

GND (8)(Ground) − There are several GND pins on the Arduino, any of which can be used to ground your circuit.

Vin (9) − This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.

Analog pins

The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital valuethat can be read by the microprocessor.

Main microcontroller

Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC.

ICSP pin

Mostly, ICSP is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.

Power LED indicator

This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.

TX and RX LEDs

On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

Digital I/O

The Arduino UNO board has 14 digital I/O pins (of which 6 provide PWM (Pulse Width Modulation) output. The pins 0 to 13 are used as a digital input or output for the Arduino board. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc.

External Interrupt Pins: This pin of the Arduino board is used to produce the External interrupt and it is done by pin numbers 2 and 3.

PWM Pins: This pins of the board is used to convert the digital signal into an analog by varying the width of the Pulse. The pin numbers 3,5,6,9,10 and 11 are used as a PWM pin.

SPI Pins: This is the Serial Peripheral Interface pin, it is used to maintain SPI communication with the help of the SPI library. SPI pins include:

SS: Pin number 10 is used as a Slave Select

MOSI: Pin number 11 is used as a Master Out Slave In

MISO: Pin number 12 is used as a Master In Slave Out

SCK: Pin number 13 is used as a Serial Clock

AREF Pin: This is an analog reference pin of the Arduino board. It is used to provide a reference voltage from an external power supply.

Introduction to ATMEGEA 328P:

ATMEGA328P is high performance, low power controller from Microchip. ATMEGA328P is an 8-bit microcontroller based on AVR RISC architecture. It is the most popular of all AVR controllers as it is used in ARDUINO boards.

Applications

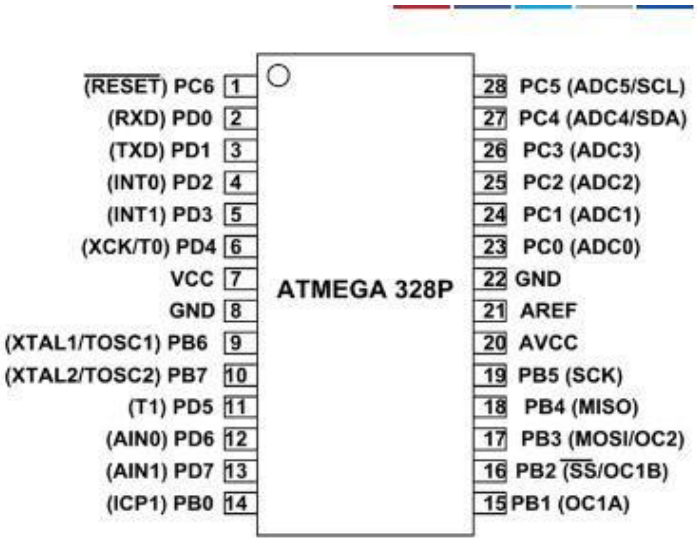
There are hundreds of applications for ATMEGA328P:

- Used in ARDUINO UNO, ARDUINO NANO and ARDUINO MICRO boards.
- Industrial control systems.
- SMPS and Power Regulation systems.
- Digital data processing.
- Analog signal measuring and manipulations.
- Embedded systems like coffee machine, vending machine.
- Motor control systems.
- Display units.
- Peripheral Interface system.

ATMega328 Pinout Configuration

ATMEGA328P is a 28 pin chip as shown in pin diagram above. Many pins of the chip here have more than one function. We will describe functions of each pin in below table.

ATMega328P Microcontroller



ATMega328P Pinout

Pin No.	Pin name	Description	Secondary Function
1	PC6 (RESET)	Pin6 of PORTC	Pin by default is used as RESET pin. PC6 can only be used as I/O pin when RSTDISBL Fuse is programmed.
2	PD0 (RXD)	Pin0 of PORTD	RXD (Data Input Pin for USART) USART Serial Communication Interface [Can be used for programming]
3	PD1 (TXD)	Pin1 of PORTD	TXD (Data Output Pin for USART) USART Serial Communication Interface [Can be used for programming] INT2(External Interrupt 2 Input)

4	PD2 (INT0)	Pin2 of PORTD	External Interrupt source 0
5	PD3 (INT1/OC2B)	Pin3 of PORTD	External Interrupt source1 OC2B(PWM - Timer/Counter2 Output Compare Match B Output)
6	PD4 (XCK/T0)	Pin4 of PORTD	T0(Timer0 External Counter Input) XCK (USART External Clock I/O)
7	VCC		Connected to positive voltage
8	GND		Connected to ground
9	PB6 (XTAL1/TOSC1)	Pin6 of PORTB	XTAL1 (Chip Clock Oscillator pin 1 or External clock input) TOSC1 (Timer Oscillator pin 1)
10	PB7 (XTAL2/TOSC2)	Pin7 of PORTB	XTAL2 (Chip Clock Oscillator pin 2) TOSC2 (Timer Oscillator pin 2)
11	PD5 (T1/OC0B)	Pin5 of PORTD	T1(Timer1 External Counter Input) OC0B(PWM - Timer/Counter0 Output Compare Match B Output)
12	PD6 (AIN0/OC0A)	Pin6 of PORTD	AIN0(Analog Comparator Positive I/P) OC0A(PWM - Timer/Counter0 Output Compare Match A Output)
13	PD7 (AIN1)	Pin7 of PORTD	AIN1(Analog Comparator Negative I/P)

14	PB0 (ICP1/CLKO)	Pin0 of PORTB	ICP1(Timer/Counter1 Input Capture Pin) CLKO (Divided System Clock. The divided system clock can be output on the PB0 pin)
15	PB1 (OC1A)	Pin1 of PORTB	OC1A (Timer/Counter1 Output Compare Match A Output)
16	PB2 (SS/OC1B)	Pin2 of PORTB	SS (SPI Slave Select Input). This pin is low when controller acts as slave. [Serial Peripheral Interface (SPI) for programming] OC1B (Timer/Counter1 Output Compare Match B Output)
17	PB3 (MOSI/OC2A)	Pin3 of PORTB	MOSI (Master Output Slave Input). When controller acts as slave, the data is received by this pin. [Serial Peripheral Interface (SPI) for programming] OC2 (Timer/Counter2 Output Compare Match Output)
18	PB4 (MISO)	Pin4 of PORTB	MISO (Master Input Slave Output). When controller acts as slave, the data is sent to master by this controller through this pin. [Serial Peripheral Interface (SPI) for programming]
19	PB5 (SCK)	Pin5 of PORTB	SCK (SPI Bus Serial Clock). This is the clock shared between this controller and other system for accurate data transfer. [Serial Peripheral Interface (SPI) for programming]
20	AVCC		Power for Internal ADC Converter

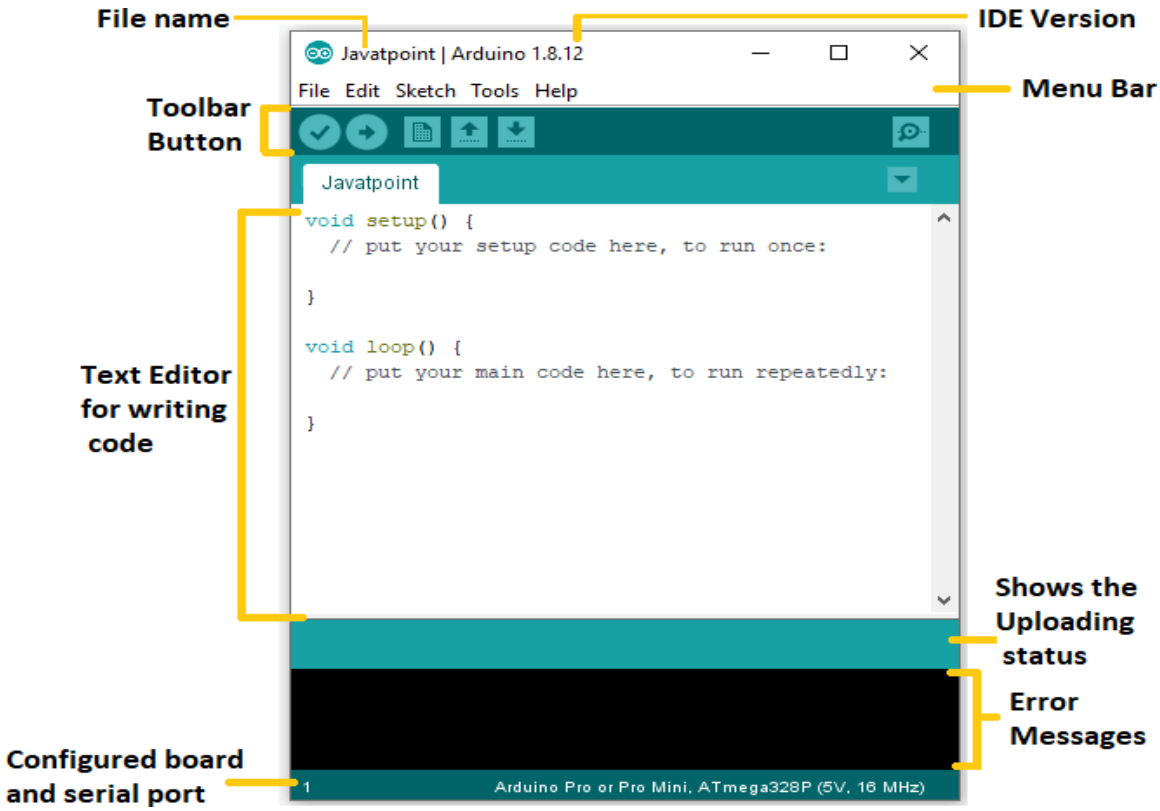
21	AREF		Analog Reference Pin for ADC
22	GND		GROUND
23	PC0 (ADC0)	Pin0 of PORTC	ADC0 (ADC Input Channel 0)
24	PC1 (ADC1)	Pin1 of PORTC	ADC1 (ADC Input Channel 1)
25	PC2 (ADC2)	Pin2 of PORTC	ADC2 (ADC Input Channel 2)
26	PC3 (ADC3)	Pin3 of PORTC	ADC3 (ADC Input Channel 3)
27	PC4 (ADC4/SDA)	Pin4 of PORTC	ADC4 (ADC Input Channel 4) SDA (Two-wire Serial Bus Data Input/output Line)
28	PC5 (ADC5/SCL)	Pin5 of PORTC	ADC5 (ADC Input Channel 5) SCL (Two-wire Serial Bus Clock Line)

Introduction to Arduino Programming:

Arduino IDE

The Arduino IDE is open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as **Windows, Mac OS X, and Linux**. It supports the programming languages C and C++. Here, IDE stands for **Integrated Development Environment**.

The Arduino IDE will appear as:

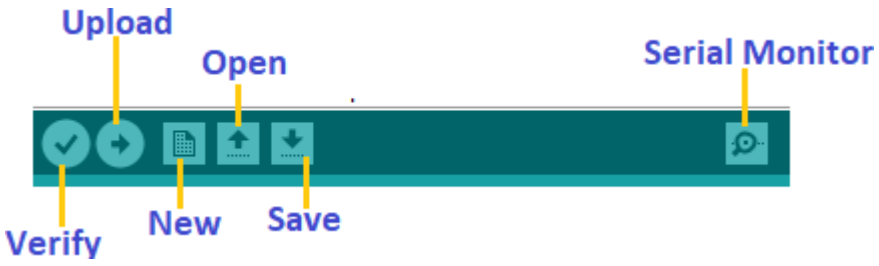


The program or code written in the Arduino IDE is often called as sketching. We need to connect the Genuino and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ino.

Toolbar Button

The icons displayed on the toolbar are **New, Open, Save, Upload,** and **Verify**

It is shown below:



Upload

The Upload button compiles and runs our code written on the screen. It further uploads the code to the connected board. Before uploading the sketch, we need to make sure that the correct board and ports are selected.

We also need a USB connection to connect the board and the computer. Once all the above measures are done, click on the Upload button present on the toolbar.

Open

The Open button is used to open the already created file. The selected file will be opened in the current window.

Save

The save button is used to save the current sketch or code.

New

It is used to create a new sketch or opens a new window.

Verify

The Verify button is used to check the compilation error of the sketch or the written code.

Serial Monitor

The serial monitor button is present on the right corner of the toolbar. It opens the serial monitor.

Arduino Functions

The functions allow a programmer to divide a specific code into various sections, and each section performs a particular task. The functions are created to perform a task multiple times in a program.

The function is a type of procedure that returns the area of code from which it is called.

For example, to repeat a task multiple times in code, we can use the same set of statements every time the task is performed.

Advantages of using Functions

Let's discuss some advantages of using functions in programming, which are listed below:

- It increases the readability of the code.
- It conceives and organizes the program.
- It reduces the chances of errors.
- It makes the program compact and small.
- It avoids the repetition of the set of statements or codes.
- It allows us to divide a complex code or program into a simpler one.
- The modification becomes easier with the help of functions in a program.

BUILT IN FUNCTIONS

The Arduino programming language comes with a huge array of built in functions that are useful for all of our projects. These built in functions allow us to save time on the programming side of our builds. We don't have to spend time developing our own functions, we can just use the ones that are provided.

The Arduino has two common functions **setup()** and **loop(),** which are called automatically in the background. The code to be executed is written inside the curly braces within these functions.The Arduino void setup and void loop functions are mandatory.

Void setup ()

void setup() - It includes the initial part of the code, which is executed only once. It is called as the **preparation block**.

As the void setup function is called only once at the very beginning of the program, this will be the place to:

- Initialize variables' values.
- Setup communications (ex: Serial).
- Setup modes for digital pins (input/output).
- Initialize any hardware component (sensor/actuator) plugged to the Arduino.
- Etc.

The void setup, as its name suggest, is made for you to do any setup required at the beginning of the program. Don't write the core functionalities here, just the initialization code.

Depending on the complexity of your program, you may have a lot of instructions to write in that void function. You can create new functions that you call from the void setup, no problem with that.

Void loop ():

void loop() - It includes the statements, which are executed repeatedly. It is called the **execution block**.

Now, in the void loop you'll write your main program, knowing that the initialization is already done. In this function, always keep in mind that the last line is followed by the first line!

Also, any variable you've declared inside the void loop will be lost when the program exits and enters the function again. So, if you want to be able to keep data between 2 void loop, make sure to declare variables in a more global scope.

As for void setup, there's no need to write all the code directly in the function. You can create as many other functions as you want (and classes too), and call those functions in the void loop. Ideally, the void loop should contain just a few lines calling other functions.

DIGITALWRITE ()

The digital Write () function is used to control the voltage that is output from a digital pin on the Arduino board. This is usually written in the loop() function of a sketch.

Syntax:
digitalWrite(pin, value)
Parameters:
pin: the pin number to be used
value: HIGH or LOW

When reading or writing to a digital pin there are only two possible values a pin can take/be-set-to: HIGH and LOW. If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to 5 volts for HIGH and 0 volts for LOW.

Example:

```
int ledPin = 13;                // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

DIGITALREAD ()

The digital Read () function reads the value from a specified digital pin, either HIGH or LOW. It will read voltage connected to the digital pin as either HIGH or LOW, on or off.

Syntax:
digitalRead(pin)

Parameters:
pin: the number of the digital pin you want to read (*int*)

Example:
Sets pin 13 to the same value as pin 7, declared as an input.

```
int ledPin = 13;      // LED connected to digital pin 13
int inPin = 7;        // pushbutton connected to digital pin 7
int val = 0;          // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin 13 as output
```

```
  pinMode(inPin, INPUT);           // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin); // read the input pin
  digitalWrite(ledPin, val);    // sets the LED to the button's value
}
```

DELAY()

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax:
delay(ms)

Parameters:
ms: the number of milliseconds to pause.

ANALOGWRITE ():

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightness's or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite(). On the Arduino UNO, analog pins 0 - 5, digital pins 11, 10, 9, 6, 5, 3 and use the analogWrite() function

Syntax:
analogWrite(pin, value)

Parameters:
pin: the pin to write to.
value: the duty cycle: between 0 (always off) and 255 (always on).

Example:
This example sets the output to the LED proportional to the value read from the potentiometer. Tjhis will allow us to control the brightness of an LED using a potentiometer.

```
int ledPin = 9;      // LED connected to digital pin 9
int analogPin = 3;  // potentiometer connected to analog pin 3
int val = 0;         // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);  // sets the pin as output
}

void loop()
{
  val = analogRead(analogPin);  // read the input pin
  analogWrite(ledPin, val / 4);  // analogRead values go from 0 to 1023, analogWrite values from 0 to
255
}
```

ANALOGREAD ():

Reads the value from the specified analog pin. It will map input voltages between 0 and 5 volts into integer values between 0 and 1023.

Syntax:
analogRead(pin)

Parameters:
pin: the number of the analog input pin to read from (0 to 5 on most boards)

Returns:
int (0 to 1023)

Example:
This example reads a value from a potentiometer.

```
int analogPin = 3;      // potentiometer middle pin connected to analog pin 3
                        // outside leads to ground and +5V
int val = 0;            // variable to store the value read

void setup()
{
  Serial.begin(9600);          //  setup serial
}

void loop()
{
  val = analogRead(analogPin);    // read the input pin
  Serial.println(val);            // debug value
}
```

Built In function	**Description**
setup()	This function is called once, when the program starts, and when the Arduino is shut down and restarted.
loop()	This function is repeatedly called while the Arduino program is running.
digitalRead()	Reads the value from a digital pin. Accepts a pin number as a parameter, and returns the HIGH or LOW constant.
digitalWrite()	Writes a HIGH or LOW value to a digital output pin. You pass the pin number and HIGH or LOW as parameters.
pinMode()	Sets a pin to be an input, or an output. You pass the pin number and the INPUT or OUTPUT value as parameters.

analogRead()	Reads the value from an analog pin.
analogWrite()	writes an analog value to a pin
delay()	pauses the program for a number of milliseconds specified as parameter

```
#define LED_PIN 13

void setup() {
    // Configure pin 13 to be a digital output
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    // Turn on the LED
    digitalWrite(LED_PIN, HIGH);
    // Wait 1 second (1000 milliseconds)
    delay(1000);
    // Turn off the LED
    digitalWrite(LED_PIN, LOW);
    // Wait 1 second
    delay(1000);
}
```

Your first Arduino program will surely involve making a led turn on the light, and then turn off.

UNIT-II

INTRODUCTION TO EMBEDDED SYSTEMS

EMBEDDED SYSTEM:

An Electronic/Electro mechanical system which is designed to perform a specific function and is a combination of both hardware and firmware (Software)

EXAMPLES:

Electronic Toys,

Mobile Handsets,

Washing Machines, Air Conditioners,

Automotive Control Units,

Set Top Box, DVD Player etc…

Embedded Systems are:

- Unique in character and behavior
- With specialized hardware and software

History of Embedded Systems:

First Recognized Modern Embedded System: Apollo Guidance Computer (AGC) developed by Charles Stark Draper at the MIT Instrumentation Laboratory. It has two modules

- 1.Command module (CM) 2. Lunar Excursion module (LEM)
- RAM size 256, 1K ,2K words
- ROM size 4K,10K,36K words
- Clock frequency is 1.024MHz
- 5000 ,3-input RTL NOR gates are used
- User interface is DSKY (display/Keyboard)

Embedded Systems Vs General Computing Systems:

General Purpose Computing System	Embedded System
A system which is a combination ofgeneric hardware and General Purpose Operating System for executing a variety of applications	A system which is a combination of special purpose hardware and embedded OSfor executing a specific set of applications
Contain a General Purpose Operating System (GPOS)	May or may not contain an operating system for functioning
Applications are alterable (programmable)by user (It is possible for the end user to re-install the Operating System, and add or removeuser applications)	The firmware of the embedded systemis pre-programmed and it is non-alterable by end-user
Performance is the key deciding factor onthe selection of the system. Always „Faster is Better"	Application specific requirements (like performance, power requirements, memory usage etc) are the key deciding factors
Less/not at all tailored towards reduced operating power requirements, options for different levels of power management.	Highly tailored to take advantage of the power saving modes supported by hardware and Operating System
Response requirements are not time critical	For certain category of embedded systems like mission critical systems, the response time requirement is highly critical
Need not be deterministic in execution behavior	Execution behavior is deterministic fo certain type of embedded systems like „Hard Real Time" systems

CLASSIFICATION OF EMBEDDED SYSTEMS:

- **Based on Generation**
- **Based on Complexity & Performance**
- **Based on deterministic behavior**
- **Based on Triggering**

Embedded Systems - Classification based on Generation:

First Generation: The early embedded systems built around 8-bit microprocessors like 8085 and Z80 and 4-bit microcontrollers with simple hardware and firmware.

EX. stepper motor control units, Digital Telephone Keypads etc.

Second Generation: Embedded Systems built around 16-bit microprocessors and 8 or 16-bit microcontrollers, following the first generation embedded systems

EX. SCADA, Data Acquisition Systems etc

Third Generation: Embedded Systems built around high performance 16/32 bit Microprocessors/controllers, Application Specific Instruction set processors like Digital Signal Processors (DSPs), and Application Specific Integrated Circuits (ASICs).The instruction set is complex and powerful.

Ex: Robotics, industrial process control, networking etc.

Fourth Generation: Embedded Systems built around System on Chips (SoC's), Re- configurable processors and multicore processors. It brings high performance, tight integration and miniaturization into the embedded device market

Ex: Smart phone devices, MIDs etc.

Embedded Systems - Classification based on Complexity & Performance

Small Scale:

The embedded systems built around low performance and low cost 8 or 16 bit microprocessors/ microcontrollers. It is suitable for simple applications and where performance is not time critical. It may or may not contain OS.

- washing machine.
- Oven.
- Automatic Door Lock.
- Motion Controlled Home Security System.
- Keyboard controller.
- CD Drive.
- fax machine.

Medium Scale:

Embedded Systems built around medium performance, low cost 16- or 32-bit microprocessors / microcontrollers or DSPs. These are slightly complex in hardware and firmware. It may contain GPOS/RTOS.Various examples of medium scale embedded

systems are routers for networking, ATM (is. Automated Teller Machine for bank transactional machines etc.

Large Scale/Complex:

- Embedded Systems built around high performance 32- or 64-bit RISC processors/controllers, RSoC or multi-core processors and PLD.
- It requires complex hardware and software.
- This system may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor.
- It contains RTOS for scheduling, prioritization and management.

Classification Based on deterministic behavior:

These are classified into two types

Event Triggered: Activities within the system (e.g., task run-times) are dynamic and depend upon occurrence of different events.

Time triggered: Activities within the system follow a statically computed schedule (i.e., they are allocated time slots during which they can take place) and thus by nature are predictable.

Major Application Areas of Embedded Systems:

Consumer Electronics: Camcorders, Cameras etc.

Household Appliances: Television, DVD players, washing machine, Fridge, Microwave Oven etc.

Home Automation and Security Systems: Air conditioners, sprinklers, Intruder detection alarms, Closed Circuit Television Cameras, Fire alarms etc.

Automotive Industry: Anti-lock breaking systems (ABS), Engine Control, Ignition Systems, Automatic Navigation Systems etc.

Telecom: Cellular Telephones, Telephone switches, Handset Multimedia pplications etc

Computer Peripherals: Printers, Scanners, Fax machines etc.

Computer Networking Systems: Network Routers, Switches, Hubs, Firewalls etc.

Health Care: Different Kinds of Scanners, EEG, ECG Machines etc.

Measurement & Instrumentation: Digital multi meters, Digital CROs, Logic Analyzers PLC systems etc.

Banking & Retail: Automatic Teller Machines (ATM) and Currency counters, Point of Sales (POS)

Card Readers: Barcode, Smart Card Readers, Hand held Devices etc.

Purpose of Embedded Systems:

Each Embedded Systems is designed to serve the purpose of any one or a combination of the following tasks.

- Data Collection/Storage/Representation
- Data Communication
- Data (Signal) Processing
- Monitoring
- Control
- Application Specific User Interface

Data Collection/Storage/Representation

Performs acquisition of data from the external world. The data may be text, audio, video or any physical quantities. The collected data can be either analog or digital. Data collection is usually done for storage, analysis, manipulation and transmission. The collected data may be stored directly in the system or may be transmitted to some other systems or it may be processed by the system or it may be deleted instantly.

Example: Digital Camera

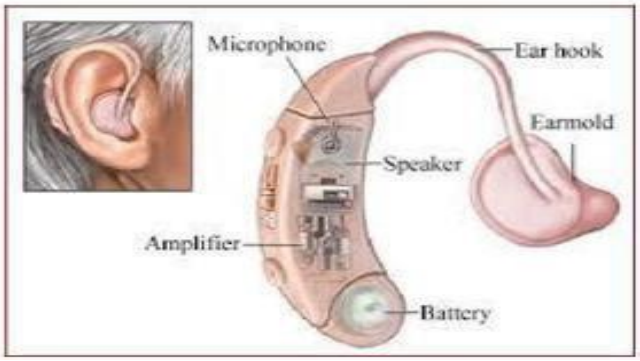
Data Communication:-

Embedded Data communication systems are deployed in applications ranging from complex satellite communication systems to simple home networking systems. Embedded Data communication systems are dedicated for data communication. The data communication can happen through a wired interface (like Ethernet, RS-232C/USB/IEEE1394 etc) or wireless interface (like Wi-Fi, GSM,/GPRS, Bluetooth, ZigBee etc)

- Network hubs, Routers, switches, Modems etc are typical examples for dedicated data transmission embedded systems

Data (Signal) Processing

- Embedded systems with Signal processing functionalities are employed in applications demanding signal processing like Speech coding, synthesis, audio video codec, transmission applications etc
- Computationally intensive systems Employs Digital Signal Processors (DSPs)



4. Monitoring:-

Embedded systems coming under this category are specifically designed for monitoring purpose. They are used for determining the state of some variables using input sensors.They cannot impose control over variables. Measuring instruments like Digital CRO, Digital Multi meter, Logic Analyzer etc used in Control & Instrumentation applications are also examples of embedded systems for monitoring purpose. Electro Cardiogram (ECG) machine for monitoring the heartbeat of a patient is a typical example for this. The sensors used in ECG are the different Electrodes connected to the patient's body

5. Control:

Embedded systems with control functionalities are used for imposing control over some variables according to the changes in input variables. Embedded system with control functionality contains both sensors and actuators. Sensors are connected to the input port for capturing the changes in environmental variable or measuring variable.The actuators connected to the output port are controlled according to the changes in input variable to put an impact on the controlling variable to bring the controlled variable to the specified range

- Air conditioner for controlling room temperature is a typical example for embedded system with „Control" functionality

Air conditioner contains a room temperature sensing element (sensor) which may be a thermistor and a handheld unit for setting up (feeding) the desired temperature.The air compressor unit acts as the actuator. The compressor is controlled according to the current room temperature and the desired temperature set by the end user.

Embedded systems possess certain specific characteristics and these are unique to each Embedded system.

Characteristics of Embedded systems:

- Application and domain specific
- Reactive and Real Time
- Operates in harsh environments
- Distributed
- Small Size and weight
- Power concerns

1. Application and Domain Specific:-

Each E. S has certain functions to perform and they are developed in such a manner to do the intended functions only. They cannot be used for any other purpose.

Ex – The embedded control units of the microwave oven cannot be replaced with AC's embedded control unit because the embedded control units of microwave oven and AC are specifically designed to perform certain specific tasks.

2. Reactive and Real Time: -

E.S are in constant interaction with the real world through sensors and user-defined input devices which are connected to the input port of the system. Any changes in the real world are captured by the sensors or input devices in real time and the control algorithm running inside the unit reacts in a designed manner to bring the controlled output variables to the desired level. E.S produce changes in output in response to the changes in the input, so they are referred as reactive systems.

Real Time system operation means the timing behavior of the system should be deterministic i.e the system should respond to requests in a known amount of time.

Example – E.S which are mission critical like flight control systems, Antilock Brake Systems (**ABS**) etc are Real Time systems.

3. Operates in Harsh Environment :–

The design of E.S should take care of the operating conditions of the area where the system is going to implement. Ex – If the system needs to be deployed in a high temperature zone, then all the components used in the system should be of high temperature grade.Also proper shock absorption techniques should be provided to systems which are going to be commissioned in places subject to high shock.

4. Distributed: –

It means that embedded systems may be a part of a larger system. Many numbers of such distributed embedded systems form a single large embedded control unit.

Ex – Automatic vending machine. It contains a card reader, a vending unit etc. Each of them are independent embedded units but they work together to perform the overall vending function.

5. Small Size and Weight:-

Product aesthetics (size, weight, shape, style, etc) is an important factor in choosing a product.

It is convenient to handle a compact device than a bulky product.

6. Power Concerns: -

Power management is another important factor that needs to be considered in designing embedded systems.

E.S should be designed in such a way as to minimize the heat dissipation by the system.

Quality Attributes of Embedded System:

Quality attributes are the non-functional requirements that need to be documented properly in any system design

Quality attributes can be classified as

1. Operational quality attributes

2. Non-operational quality attributes.

1. Operational quality attributes

The operational quality attributes represent the quality attributes related to the embedded system when it is in the operational mode or online mode

The Operational Quality Attributes are

- Response
- Throughput
- Reliability
- Maintainability
- Security
- Saftey

1. Response:

It is the measure of quickness of the system.It tells how fast the system is tracking the changes in input variables. Most of the E.S demands fast response which should be almost real time.

Ex – Flight control application

2. Throughput

It deals with the efficiency of a system. It can be defined as the rate of production or operation of a defined process over a stated period of time.The rates can be expressed in terms of products, batches produced or any other meaningful measurements.

Ex – In case of card reader throughput means how many transactions the reader can perform in a minute or in an hour or in a day.

3. Reliability:

It is a measure of how much we can rely upon the proper functioning of the system.

Mean Time between Failure (MTBF) and Mean Time To Repair (MTTR) are the terms used in determining system reliability.MTBF gives the frequency of failures in hours/weeks/months.

MTTR specifies how long the system is allowed to be out of order following a failure.For embedded system with critical application need, it should be of the order of minutes.

4. Maintainability:

It deals with support and maintenance to the end user or client in case of technical issues and product failure or on the basis of a routine system checkup.Reliability and maintainability are complementary to each other.A more reliable system means a system with less corrective maintainability requirements and vice versa. Maintainability can be broadly classified into two categories. Scheduled or Periodic maintenance (Preventive maintenance), Corrective maintenance to unexpected failures

5. Security:

Confidentiality, Integrity and availability are the three major measures of information security.

Confidentiality deals with protection of data and application from unauthorized disclosure.

Integrity deals with the protection of data and application from unauthorized modification.

Availability deals with protection of data and application from unauthorized users.

6. Saftey:

Safety deals with the possible damages that can happen to the operator, public and the environment due to the breakdown of an Embedded System. The breakdown of an embedded system may occur due to a hardware failure or a firmware failure. Safety analysis is a must in product engineering to evaluate the anticipated damages and determine the best course of action to bring down the consequences of damage to an acceptable level.

Non Operational Quality Attributes:

The quality attributes that needs to be addressed for the product not on the basis of operational aspects are grouped under this category.

The Non Operational Quality Attributes are

- Testability and Debugability
- Evolvability
- Portability
- Time to Prototype and Market
- Per Unit Cost and Revenue

1.Testability and Debugability:

Testability deals with how easily one can test the design, application and by which means it can be done.For an E.S testability is applicable to both the embedded hardware and firmware. Embedded hardware testing ensures that the peripherals and total hardware functions in the desired manner, whereas firmware testing ensures that the firmware is functioning in the expected way

Debug-ability is a means of debugging the product from unexpected behavior in the system

Debug-ability is two level process

1.Hardware level 2.software level

1. Hardware level: It is used for finding the issues created by hardware problems.

2. Software level: It is employed for finding the errors created by the flaws in the software

2. Evolvability:-

- It is a term which is closely related to Biology.
- It is referred as the non-heritable variation.
- For an embedded system evolvability refers to the ease with which the embedded product can be modified to take advantage of new firmware or hardware technologies.

3. **Portability: -**

- It is the measure of system independence.

An embedded product is said to be portable if the product is capable of functioning in various environments, target processors and embedded operating systems

4. **Time-to-Prototype and Market:-**

- It is the time elapsed between the conceptualization of a product and the time at which the product is ready for selling.
- The commercial embedded product market is highly competitive and time to market the product is critical factor in the success of commercial embedded product.
- There may be multiple players in embedded industry who develop products of the same category (like mobile phone).

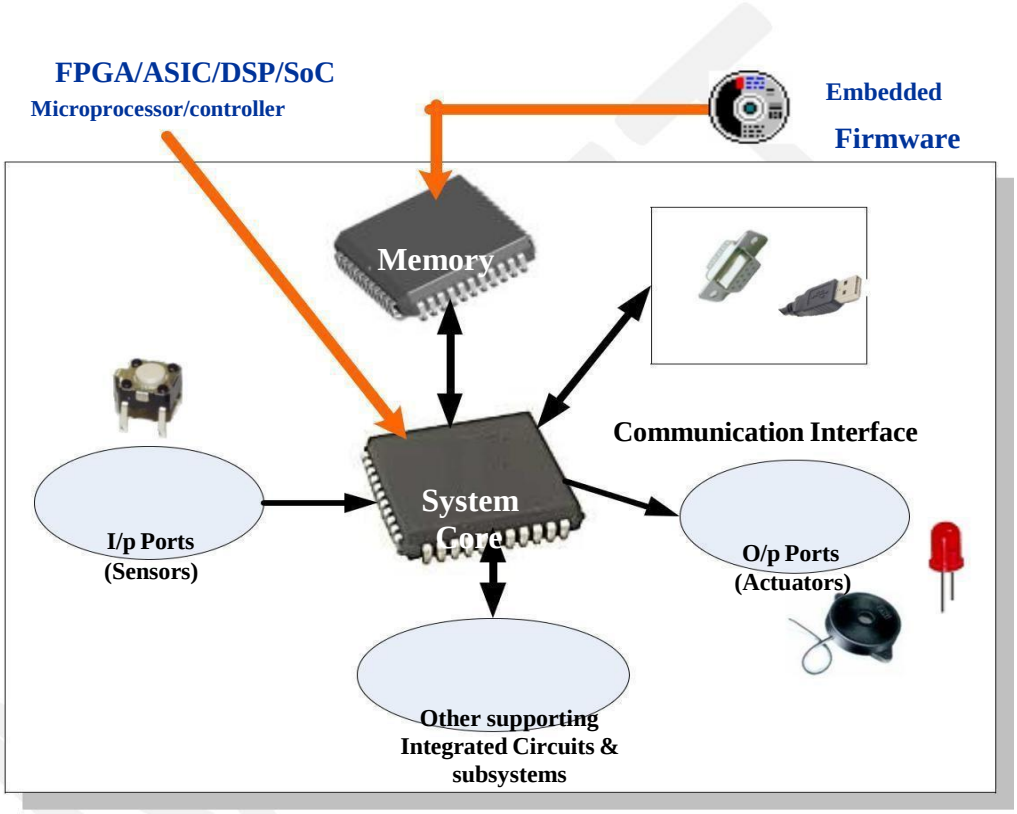
5. **Per Unit Cost and Revenue:-**

- Cost is a factor which is closely monitored by both end user and product manufacturer.
- Cost is highly sensitive factor for commercial products
- Any failure to position the cost of a commercial product at a nominal rate may lead to the failure of the product in the market.
- Proper market study and cost benefit analysis should be carried out before taking a decision on the per-unit cost of the embedded product.
- The ultimate aim of the product is to generate marginal profit so the budget and total cost should be properly balanced to provide a marginal profit.

UNIT-III
TYPICAL EMBEDDED SYSTEM

ELEMENTS OF EMBEDDED SYSTEMS:

An embedded system is a combination of 3 things, Hardware Software Mechanical Components and it is supposed to do one specific task only. A typical embedded system contains a single chip controller which acts as the master brain of the system. Diagrammatically an embedded system can be represented as follows:



Embedded System
Real World

Embedded systems are basically designed to regulate a physical variable (such Microwave Oven) or to manipulate the state of some devices by sending some signals to the actuators or devices connected to the output port system (such as temperature in Air Conditioner), in response to the input signal provided by the end users or sensors which are connected to the input ports.

The control is achieved by processing the information coming from the sensors and user interfaces and controlling some actuators that regulate the physical variable.

Keyboards, push button, switches, etc. are Examples of common user interface input devices and LEDs, LCDs, Piezoelectric buzzers, etc examples for common user interface output devices for a typical embedded system.The requirement of type of user interface changes from application to application based on domain.

Some embedded systems do not require any manual intervention for their operation. They automatically sense the input parameters from real world through sensors which are connected at input port. The sensor information is passed to the processor after signal conditioning and digitization. The core of the system performs some predefined operations on input data with the help of embedded firmware in the system and sends some actuating signals to the actuator connect connected to the output port of the system.

The memory of the system is responsible for holding the code (control algorithm and other important configuration details). There are two types of memories are used in any embedded system. Fixed memory (ROM) is used for storing code or program. The user cannot change the firmware in this type of memory. The most common types of memories used in embedded systems for control algorithm storage are OTP,PROM,UVEPROM,EEPROM and FLASH

An embedded system without code (i.e. the control algorithm) implemented memory has all the peripherals but is not capable of making decisions depending on the situational as well as real world changes.

Memory for implementing the code may be present on the processor or may be implemented as a separate chip interfacing the processor

In a controller based embedded system, the controller may contain internal memory for storing code such controllers are called Micro-controllers with on-chip ROM, eg. Atmel AT89C51.

The Core of the Embedded Systems: The core of the embedded system falls into any one of the following categories.

- **General Purpose and Domain Specific Processors**
  - Microprocessors
  - Microcontrollers
  - Digital Signal Processors
- **Programmable Logic Devices (PLDs)**
- **Application Specific Integrated Circuits (ASICs)**
- **Commercial off the shelf Components (COTS)**

GENERAL PURPOSE AND DOMAIN SPECIFIC PROCESSOR:

- Almost 80% of the embedded systems are processor/ controller based.
- The processor may be microprocessor or a microcontroller or digital signal processor, depending on the domain and application.

Microprocessor:

- A silicon chip representing a Central Processing Unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of Instructions, which is specific to the manufacturer
- In general the CPU contains the Arithmetic and Logic Unit (ALU), Control Unit and Working registers
- Microprocessor is a dependant unit and it requires the combination of other hardware like Memory, Timer Unit, and Interrupt Controller etc for proper functioning.
- Intel claims the credit for developing the first Microprocessor unit Intel 4004, a 4 bit processor which was released in Nov 1971

· Developers of microprocessors.

- Intel – Intel 4004 – November 1971(4-bit)
- Intel – Intel 4040.
- Intel – Intel 8008 – April 1972.
- Intel – Intel 8080 – April 1974(8-bit).
- Motorola – Motorola 6800.
- Intel – Intel 8085 – 1976.
- Zilog - Z80 – July 1976

Microcontroller:

- A highly integrated silicon chip containing a CPU, scratch pad RAM, Special and General purpose Register Arrays, On Chip ROM/FLASH memory for program storage, Timer and Interrupt control units and dedicated I/O ports
- Microcontrollers can be considered as a super set of Microprocessors
- Microcontroller can be general purpose (like Intel 8051, designed for generic applications and domains) or application specific (Like Automotive AVR from Atmel Corporation. Designed specifically for automotive applications)
- Since a microcontroller contains all the necessary functional blocks for independent working, they found greater place in the embedded domain in place of microprocessors
- Microcontrollers are cheap, cost effective and are readily available in the market
- Texas Instruments TMS 1000 is considered as the world"s first microcontroller

Microprocessor Vs Microcontroller:

Microprocessor	Microcontroller
A silicon chip representing a Central Processing Unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of Instructions	A microcontroller is a highly integrated chip that contains a CPU, scratch pad RAM, Special and General purpose Register Arrays, On Chip ROM/FLASH memory for program storage, Timer and Interrupt control units and dedicated I/O ports
It is a dependent unit. It requires the combination of other chips like Timers, Program and data memory chips, Interrupt controllers etc for functioning	It is a self contained unit and it doesn't require external Interrupt Controller, Timer, UART etc for its functioning
Most of the time general purpose in design and operation	Mostly application oriented or domain specific
Doesn"t contain a built in I/O port. The I/O Port functionality needs to be implemented with the help of external Programmable Peripheral Interface Chips like 8255	Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit Port or as individual port pins
Targeted for high end market where performance is important	Targeted for embedded market where performance is not so critical (At present this demarcation is invalid)
Limited power saving options compared to microcontrollers	Includes lot of power saving features

General Purpose Processor (GPP) Vs Application Specific Instruction Set Processor (ASIP)

- General Purpose Processor or GPP is a processor designed for general computational tasks
- GPPs are produced in large volumes and targeting the general market. Due to the high volume production, the per unit cost for a chip is low compared to ASIC or other specific ICs
- A typical general purpose processor contains an Arithmetic and Logic Unit (ALU) and Control Unit (CU)
- Application Specific Instruction Set processors (ASIPs) are processors with architecture and instruction set optimized to specific domain/application requirements like Network processing, Automotive, Telecom, media applications, digital signal processing, control applications etc.
- ASIPs fill the architectural spectrum between General Purpose Processors and Application Specific Integrated Circuits (ASICs)
- The need for an ASIP arises when the traditional general purpose processor are unable to meet the increasing application needs
- Some Microcontrollers (like Automotive AVR, USB AVR from Atmel), System on Chips, Digital Signal Processors etc are examples of Application Specific Instruction Set Processors (ASIPs)
- ASIPs incorporate a processor and on-chip peripherals, demanded by the application requirement, program and data memory

Digital Signal Processors (DSPs):

- Powerful special purpose 8/16/32 bit microprocessors designed specifically to meet the computational demands and power constraints of today's embedded audio, video, and communications applications
- Digital Signal Processors are 2 to 3 times faster than the general-purpose microprocessors in signal processing applications
- DSPs implement algorithms in hardware which speeds up the execution whereas general purpose processors implement the algorithm in firmware and the speed of execution depends primarily on the clock for the processors
- DSP can be viewed as a microchip designed for performing high speed computational

operations for „addition“, „subtraction“, „multiplication“ and „division“

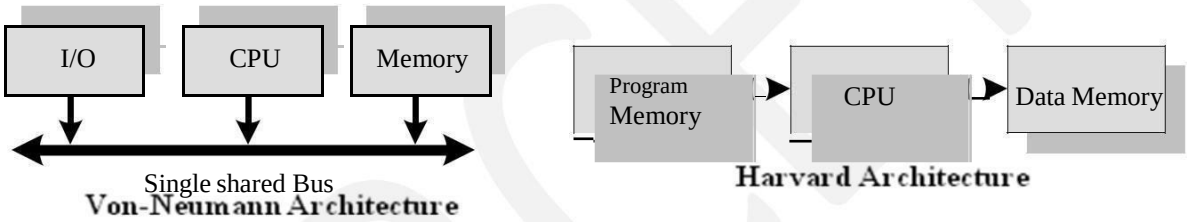
- A typical Digital Signal Processor incorporates the following key units
  - Program Memory
  - Data Memory
  - Computational Engine
  - I/O Unit
- Audio video signal processing, telecommunication and multimedia applications are typical examples where DSP is employed

RISC V/s CISC Processors/Controllers:

RISC	CISC
Lesser no. of instructions	Greater no. of Instructions
Instruction Pipelining and increased execution speed	Generally no instruction pipelining feature
Orthogonal Instruction Set (Allows each instruction to operate on any register and use any addressing mode)	Non Orthogonal Instruction Set (All instructions are not allowed to operate on any register and use any addressing mode. It is instruction specific)
Operations are performed on registers only, the only memory operations are load and store	Operations are performed on registers or memory depending on the instruction
Large number of registers are available	Limited no. of general purpose registers
Programmer needs to write more code to execute a task since the instructions are simpler ones	. A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using more simpler single instructions in RISC
Single, Fixed length Instructions	Variable length Instructions
Less Silicon usage and pin count	More silicon usage since more additional decoder logic is required to implement the complex instruction decoding.
With Harvard Architecture	Can be Harvard or Von-Neumann Architecture

Harvard V/s Von-Neumann Processor/Controller Architecture

- The terms Harvard and Von-Neumann refers to the processor architecture design.
- Microprocessors/controllers based on the **Von-Neumann** architecture shares a single common bus for fetching both instructions and data. Program instructions and data are stored in a common main memory
- Microprocessors/controllers based on the **Harvard** architecture will have separate data bus and instruction bus. This allows the data transfer and program fetching to occur simultaneously on both buses
- With Harvard architecture, the data memory can be read and written while the program memory is being accessed. These separated data memory and code memory buses allow one instruction to execute while the next instruction is fetched ("Pre-fetching")

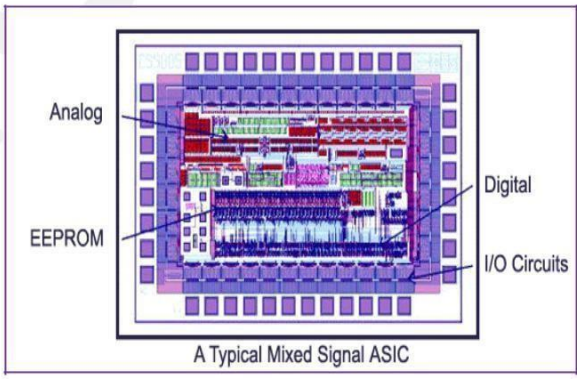


Harvard V/s Von-Neumann Processor/Controller Architecture:

Harvard Architecture	Von-Neumann Architecture
Separate buses for Instruction and Data fetching	Single shared bus for Instruction and Data fetching
Easier to Pipeline, so high performance can be achieved	Low performance Compared to Harvard Architecture
Comparatively high cost	Cheaper
No memory alignment problems	Allows self modifying codes†
Since data memory and program memory are stored physically in different locations, no chances for accidental corruption of program memory	Since data memory and program memory are stored physically in same chip, chances for accidental corruption of program memory

Application Specific Integrated Circuit (ASIC):

- A microchip designed to perform a specific or unique application. It is used as replacement to conventional general purpose logic chips.
- ASIC integrates several functions into a single chip and thereby reduces the system development cost
- Most of the ASICs are proprietary products. As a single chip, ASIC consumes very small area in the total system and thereby helps in the design of smaller systems with high capabilities/functionalities.
- ASICs can be pre-fabricated for a special application or it can be custom fabricated by using the components from a re-usable „*building block*" library of components for a particular customer application



A Typical Mixed Signal ASIC

- Fabrication of ASICs requires a non-refundable initial investment (Non Recurring Engineering (NRE) charges) for the process technology and configuration expenses
- If the Non-Recurring Engineering Charges (NRE) is born by a third party and the Application Specific Integrated Circuit (ASIC) is made openly available in the market, the ASIC is referred as Application Specific Standard Product (ASSP)
- The ASSP is marketed to multiple customers just as a general-purpose product , but to a smaller number of customers since it is for a specific application.

- Some ASICs are proprietary products , the developers are not interested in revealing the internal details.

Programmable Logic Devices (PLDs):

- Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform.
- Logic devices can be classified into two broad categories - Fixed and Programmable. The circuits in a fixed logic device are permanent, they perform one function or set of functions - once manufactured, they cannot be changed
- Programmable logic devices (PLDs) offer customers a wide range of logic capacity, features, speed, and voltage characteristics - and these devices can be re-configured to perform any number of functions at any time
- Designers can use inexpensive software tools to quickly develop, simulate, and test their logic designs in PLD based design. The design can be quickly programmed into a device, and immediately tested in a live circuit
- PLDs are based on re-writable memory technology and the device is reprogrammed to change the design

Programmable Logic Devices (PLDs) – CPLDs and FPGA

- Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs) are the two major types of programmable logic devices

FPGA:

- FPGA is an IC designed to be configured by a designer after manufacturing.
- FPGAs offer the highest amount of logic density, the most features, and the highest performance.
- Logic gate is Medium to high density ranging from **1K to 500K** system gates

- These advanced FPGA devices also offer features such as built-in hardwired processors (such as the IBM Power PC), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies

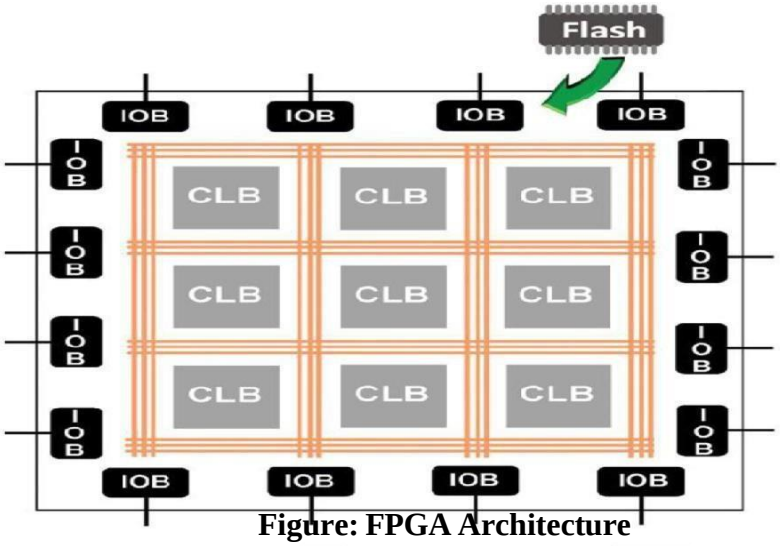


Figure: FPGA Architecture

- These advanced FPGA devices also offer features such as built-in hardwired processors, substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies.

- FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing

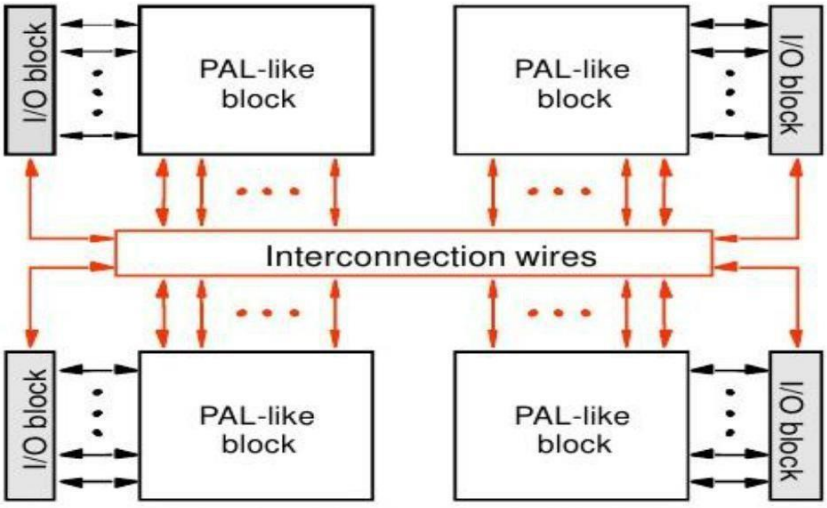
CPLD:

- A **complex programmable logic device** (**CPLD**) is a programmable logic device with complexity between that of PALs and FPGAs, and architectural features of both.

  CPLDs, by contrast, offer much smaller amounts of logic - up to about 10,000 gates.

- CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications.

Structure of a CPLD



- CPLDs such as the Xilinx **CoolRunner** series also require extremely low amounts of power and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

ADVANTAGES OF PLDs:

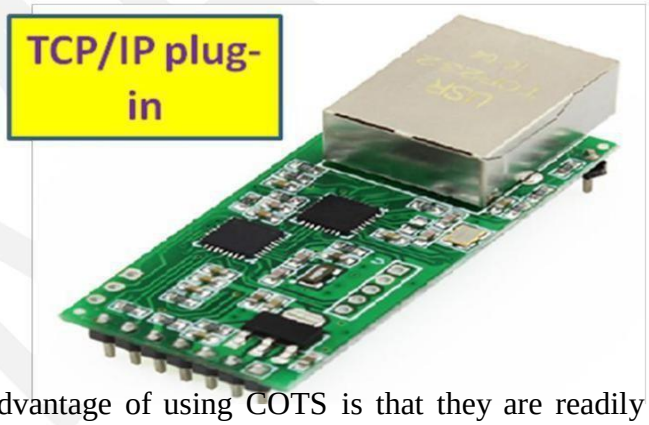
- PLDs offer customer much more flexibility during design cycle
- PLDSs do not require long lead times for prototype or production-the PLDs are already on a distributor‟s self and ready for shipment
- PLDs do not require customers to pay for large NRE costs and purchase expensive mask sets
- PLDs allow customers to order just the number of parts required when they need them. allowing them to control inventory.
- PLDs are reprogrammable even after a piece of equipment is shipped to a customer.
- The manufacturers able to add new features or upgrade the PLD based products that are in the field by uploading new programming file

Commercial off the Shelf Component (COTS):

- A Commercial off-the-shelf (COTS) product is one which is used „*as-is*‟
- COTS products are designed in such a way to provide easy integration and interoperability with existing system components

- Typical examples for the COTS hardware unit are Remote Controlled Toy Car control unit including the RF Circuitry part, High performance, high frequency microwave electronics (2 to 200 GHz), High bandwidth analog-to-digital converters, Devices and components for operation at very high temperatures, Electro-optic IR imaging arrays, UV/IR Detectors etc

- A COTS component in turn contains a General Purpose Processor (GPP) or Application Specific Instruction Set Processor (ASIP) or Application Specific Integrated Chip (ASIC)/Application Specific Standard Product (ASSP) or Programmable Logic Device (PLD)



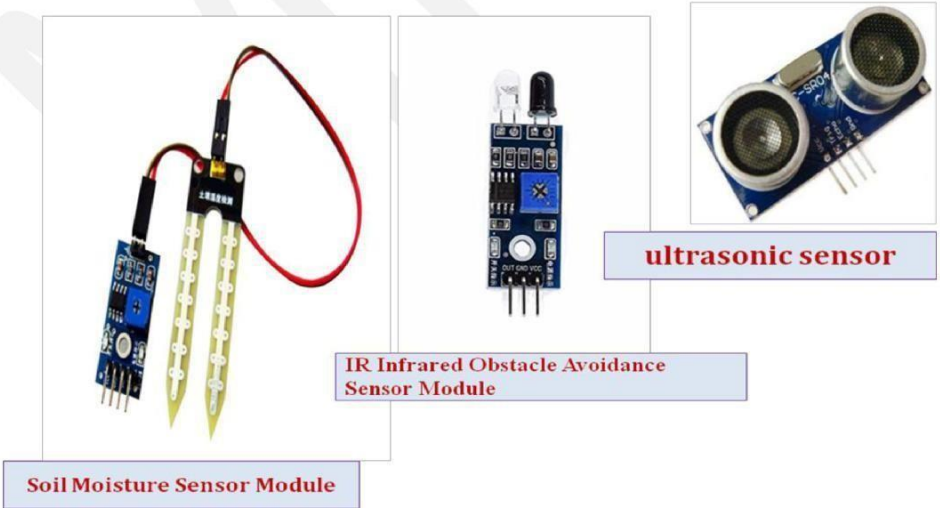
- The major advantage of using COTS is that they are readily available in the market, cheap and a developer can cut down his/her development time to a great extend.
- There is no need to design the module yourself and write the firmware .
- Everything will be readily supplied by the COTs manufacturer.

Sensors & Actuators:

- Embedded system is in constant interaction with the real world
- Controlling/monitoring functions executed by the embedded system is achieved in accordance with the changes happening to the Real World.
- The changes in the system environment or variables are detected by the sensors connected to the input port of the embedded system.
- If the embedded system is designed for any controlling purpose, the system will produce some changes in controlling variable to bring the controlled variable to the desired value.
- It is achieved through an actuator connected to the out port of the embedded system.

Sensor:

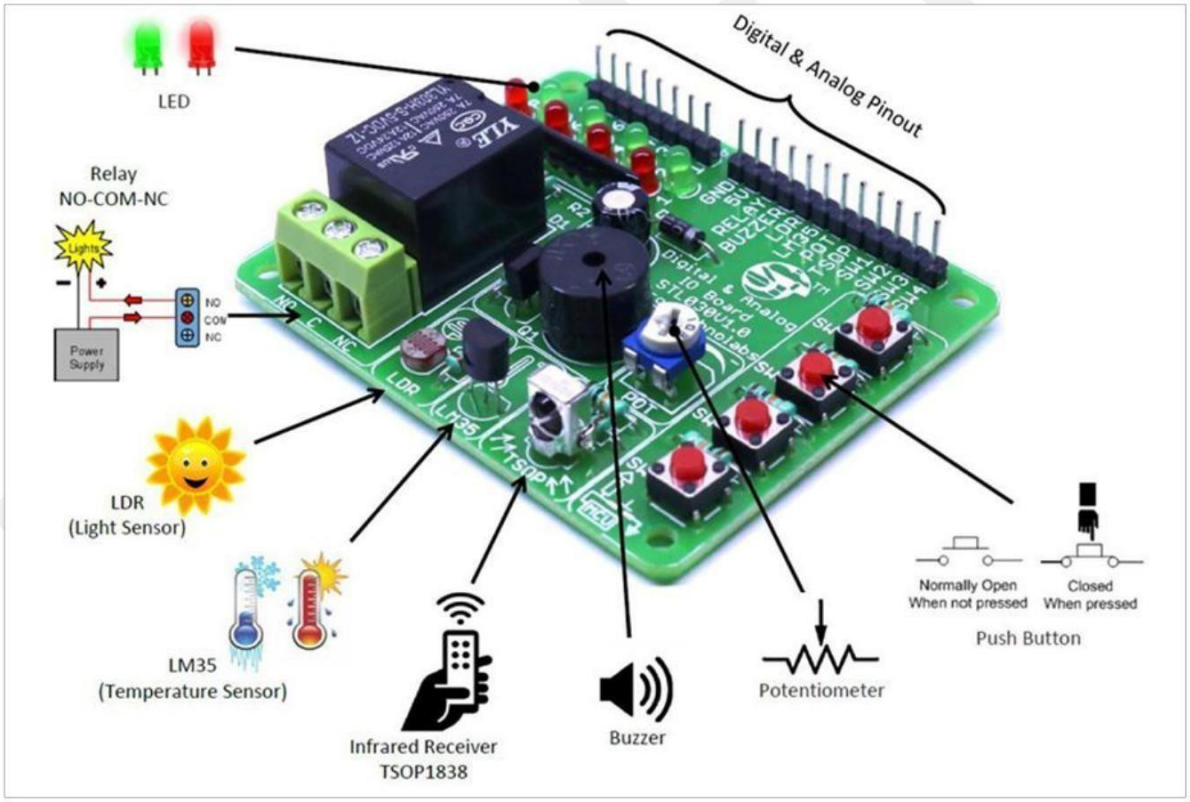
- A transducer device which converts energy from one form to another for any measurement or control purpose. Sensors acts as input device
- Eg. Hall Effect Sensor which measures the distance between the cushion and magnet in the Smart Running shoes from adidas
- **Example: IR, humidity , PIR(passive infra red) , ultrasonic , piezoelectric , smoke sensors**



Actuator:

- A form of transducer device (mechanical or electrical) which converts signals to corresponding physical action (motion). Actuator acts as an output device

  Eg. Micro motor actuator which adjusts the position of
- the cushioning element in the Smart Running shoes from adidas



Silicon TechnoLabs Digital Analog Arduino Starter kit

Communication Interface:

- Communication interface is essential for communicating with various subsystems of the embedded system and with the external world

- The communication interface can be viewed in two different perspectives; namely;

1. Device/board level communication interface (Onboard Communication Interface)

2. Product level communication interface (External Communication Interface)

1. Device/board level communication interface (Onboard Communication Interface):

The communication channel which interconnects the various components within an embedded product is referred as Device/board level communication interface (Onboard Communication Interface)

- Examples: Serial interfaces like I2C, SPI, UART, 1-Wire etc and Parallel bus interface

2. Product level communication interface (External Communication Interface):

The „Product level communication interface" (External Communication Interface) is responsible for data transfer between the embedded system and other devices or modules. The external communication interface can be either wired media or wireless media and it can be a serial or parallel interface.

- Examples for wireless communication interface: Infrared (IR), Bluetooth (BT), Wireless LAN (Wi-Fi), Radio Frequency waves (RF), GPRS etc.
- Examples for wired interfaces: RS-232C/RS-422/RS 485, USB, Ethernet (TCP-IP), IEEE 1394 port, Parallel port etc.

1. Device/board level or On board communication interfaces: The

Communication channel which interconnects the various components within an embedded product is referred as Device/board level communication interface (Onboard Communication Interface)

These are classified into

I2C (Inter Integrated Circuit) Bus

SPI (Serial Peripheral Interface) Bus

UART (Universal Asynchronous Receiver Transmitter)

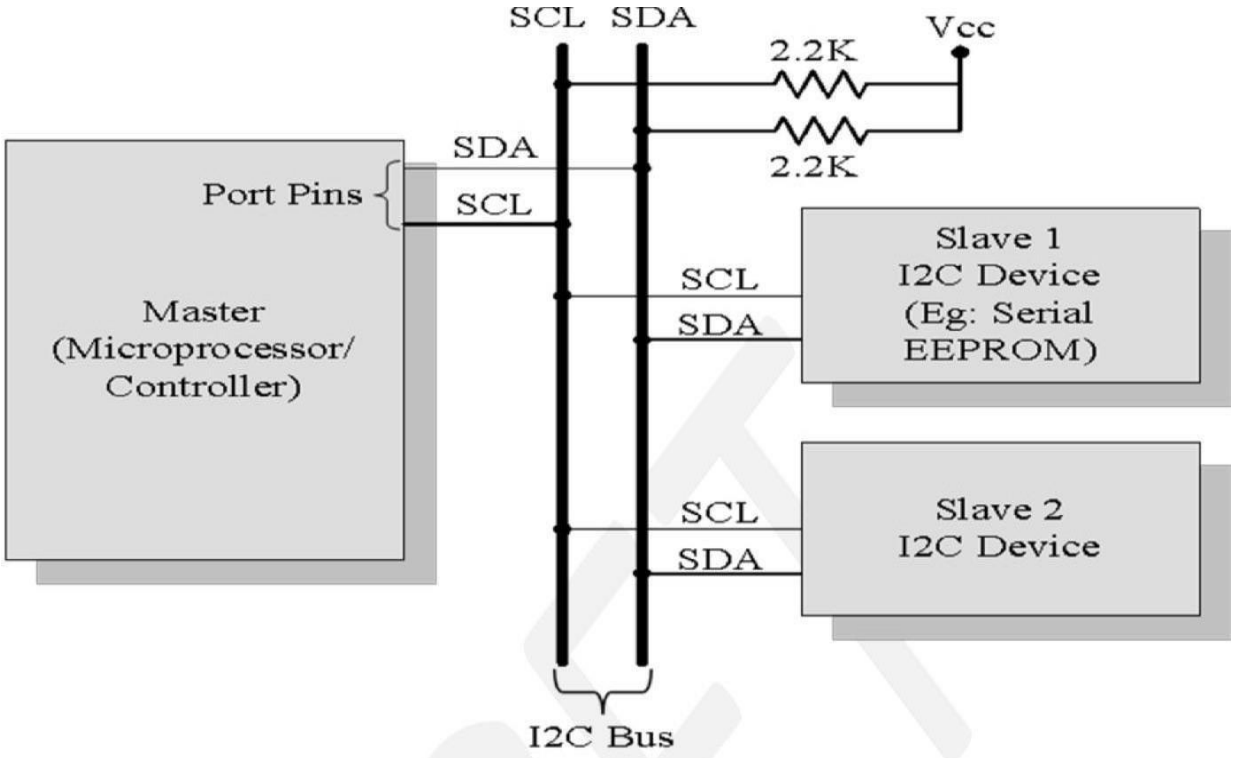
1-Wires Interface

Parallel Interface

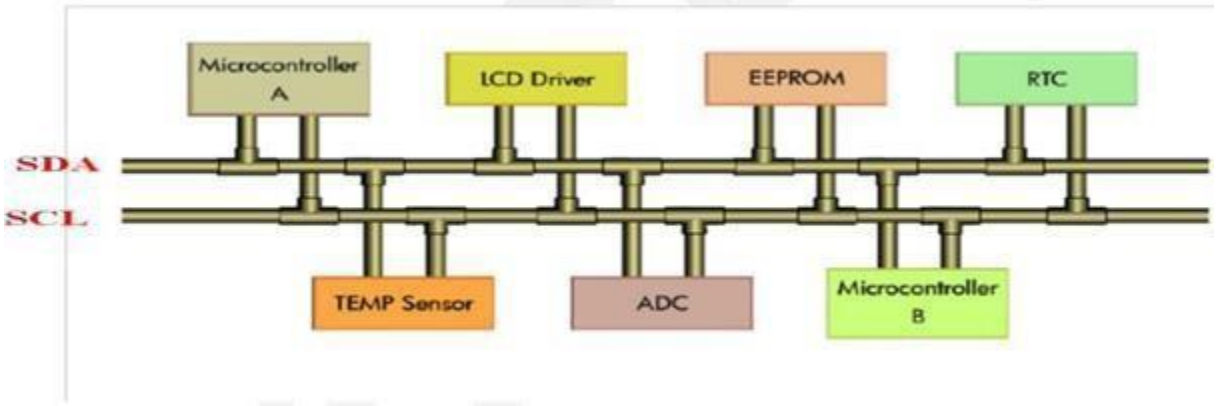
1 I2C (Inter Integrated Circuit) Bus:

Inter Integrated Circuit Bus (I2C - Pronounced „I square C") is a synchronous bi-directional half duplex (one-directional communication at a given point of time) two wire serial interface bus.The concept of I2C bus was developed by „Philips Semiconductors" in the early 1980"s. Theoriginal intention of I2C was to provide an easy way of connection between a microprocessor/microcontroller system and the peripheral chips in Television sets.

The I2C bus is comprised of two bus lines, namely; Serial Clock – SCL and Serial Data – SDA.



I2C Bus Interfacing



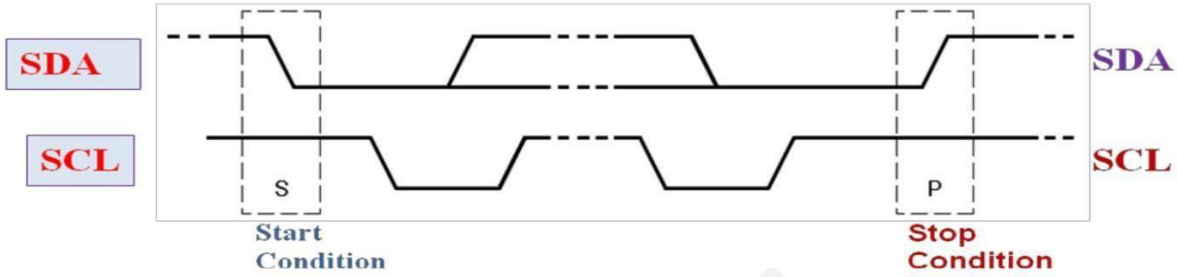
SCL line is responsible for generating synchronization clock pulses and SDA is responsible for transmitting the serial data across devices.I2C bus is a shared bus system to which many number of I2C devices can be connected. Devices connected to the I2C bus can act as either „Master" device or „Slave" device.

The „Master" device is responsible for controlling the communication by initiating/terminating data transfer, sending data and generating necessary synchronization clock pulses.

Slave devices wait for the commands from the master and respond upon receiving the commands. Master and „Slave" devices can act as either transmitter or receiver. Regardless whether a master is acting as transmitter or receiver, the synchronization clock signal is generated by the „Master" device only.I2C supports multi masters on the same bus.

The sequence of operation for communicating with an I2C slave device is:

1. Master device pulls the clock line (SCL) of the bus to „HIGH"
2. Master device pulls the data line (SDA) „LOW", when the SCL line is at logic „HIGH" (This is the „Start" condition for data transfer)



3. Master sends the address (7 bit or 10 bit wide) of the „Slave" device to which it wants to communicate, over the SDA line.
4. Clock pulses are generated at the SCL line for synchronizing the bit reception by the slave device.
5. The MSB of the data is always transmitted first.
6. The data in the bus is valid during the „HIGH" period of the clock signal
7. In normal data transfer, the data line only changes state when the clock is low.

R/Wr

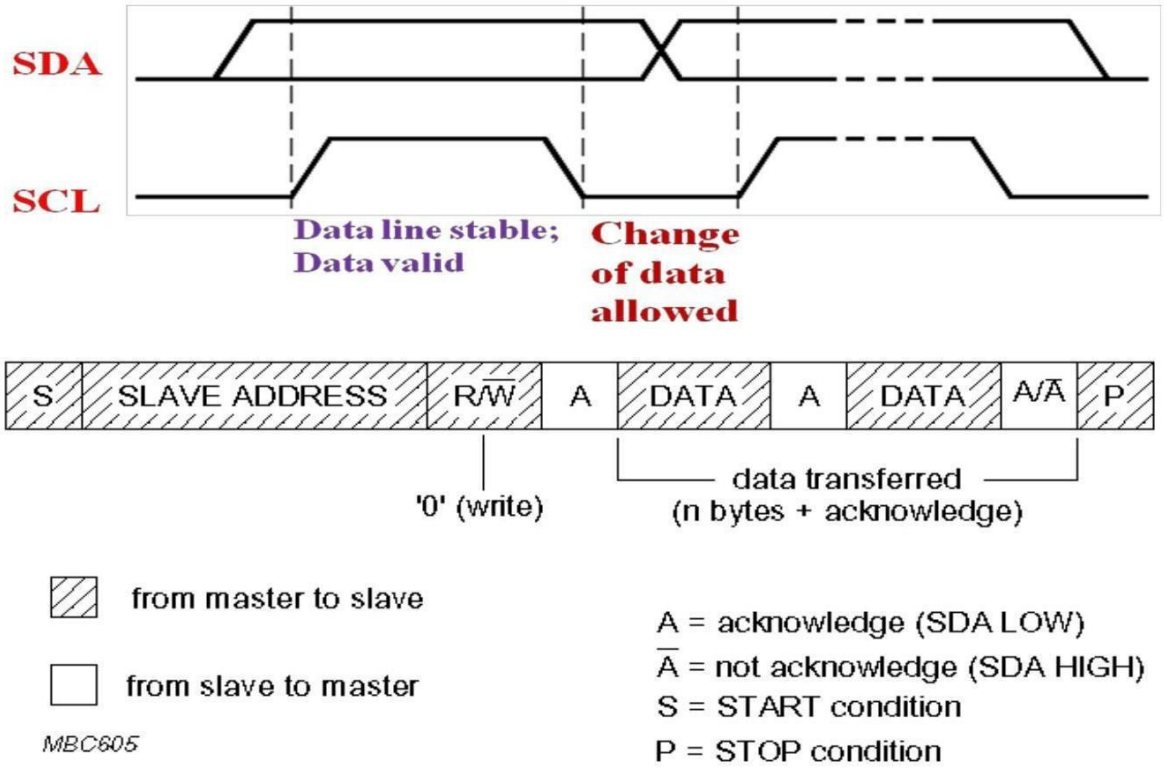
0 – Master writes to the slave

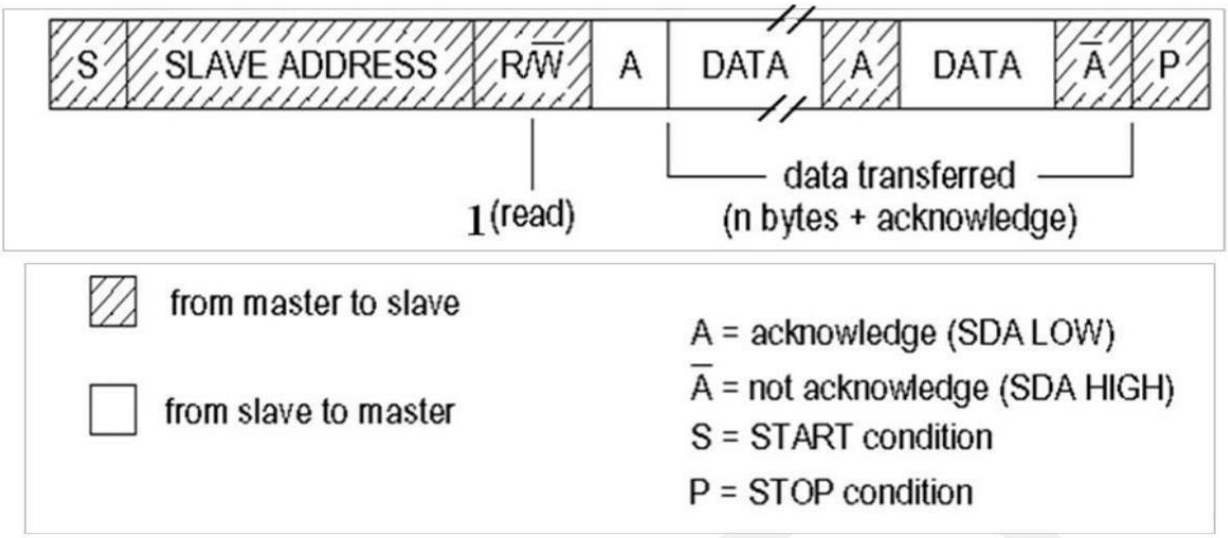
1 – Master read from slave

ACK – Generated by the slave whose address has been output.

8. Master waits for the acknowledgement bit from the slave device whose address is sent on the bus along with the Read/Write operation command.

9. Slave devices connected to the bus compares the address received with the address assigned to them
10. The Slave device with the address requested by the master device responds by sending an acknowledge bit (Bit value =1) over the SDA line
11. Upon receiving the acknowledge bit, master sends the 8bit data to the slave device over SDA line, if the requested operation is „Write to device".
12. If the requested operation is „Read from device", the slave device sends data to the master over the SDA line.
13. Master waits for the acknowledgement bit from the device upon byte transfer complete for a write operation and sends an acknowledge bit to the slave device for a read operation
14. Master terminates the transfer by pulling the SDA line „HIGH" when the clock line SCL is at logic „HIGH" (Indicating the „STOP" condition).

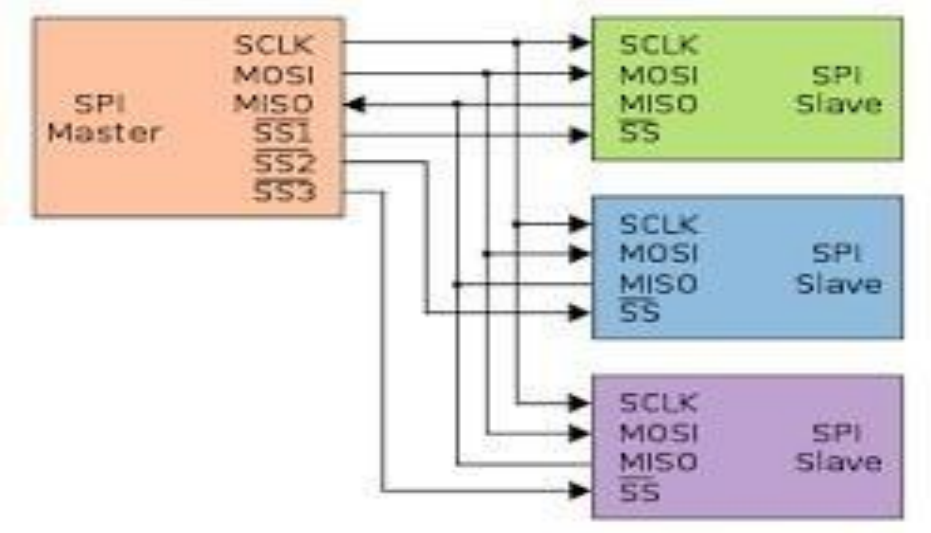




1.2 Serial Peripheral Interface (SPI) Bus:

The Serial Peripheral Interface Bus (SPI) is a synchronous bi-directional full duplex four wire serial interface bus. The concept of SPI is introduced by Motorola.SPI is a single master multi-slave system.

- It is possible to have a system where more than one SPI device can be master, provided the condition only one master device is active at any given point of time, is satisfied.
- SPI is used to send data between Microcontrollers and small peripherals such as shift registers, sensors, and SD cards.



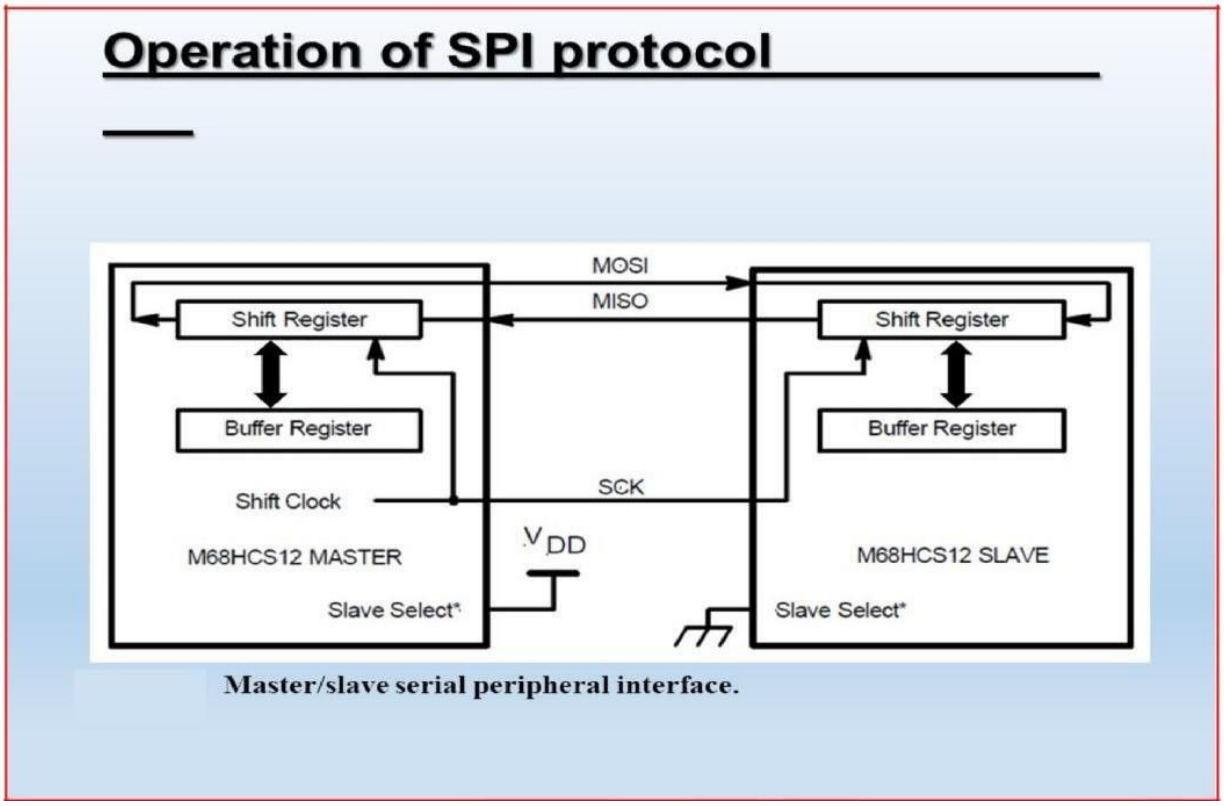
SPI requires four signal lines for communication. They are:

Master Out Slave In (MOSI): Signal line carrying the data from master to slave device. It is also known as Slave Input/Slave Data In (SI/SDI)

Master In Slave Out (MISO): Signal line carrying the data from slave to master device. It is also known as Slave Output (SO/SDO)

Serial Clock (SCLK): Signal line carrying the clock signals
Slave Select (SS): Signal line for slave device select. It is an active low signal.
The master device is responsible for generating the clock signal.
Master device selects the required slave device by asserting the corresponding slave devices slave select signal „LOW".

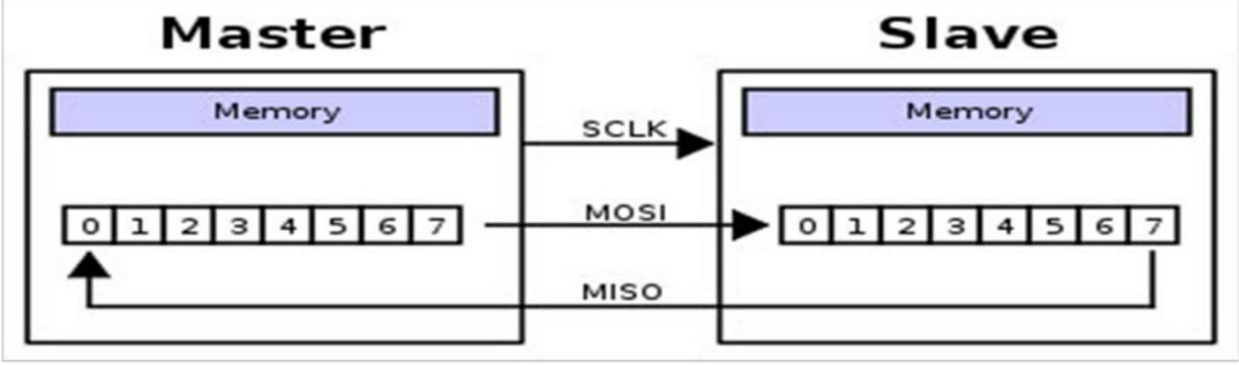
- The data out line (MISO) of all the slave devices when not selected floats at high impedance state
- The serial data transmission through SPI Bus is fully configurable.
- SPI devices contain certain set of registers for holding these configurations.
- The Serial Peripheral Control Register holds the various configuration parameters like master/slave selection for the device, baudrate selection for communication, clock signal control etc.
- The status register holds the status of various conditions for transmission and reception.SPI works on the principle of „Shift Register".
- The master and slave devices contain a special shift register for the data to transmit or receive.
- The size of the shift register is device dependent. Normally it is a multiple of 8.



Master/slave serial peripheral interface.

- During transmission from the master to slave, the data in the master"s shift register is shifted out to the MOSI pin and it enters the shift register of the slave device through the MOSI pin of the slave device.

- At the same time the shifted out data bit from the slave device's shift register enters the shift register of the master device through MISO pin



Master shifts out data to Slave, and shift in data from Slave

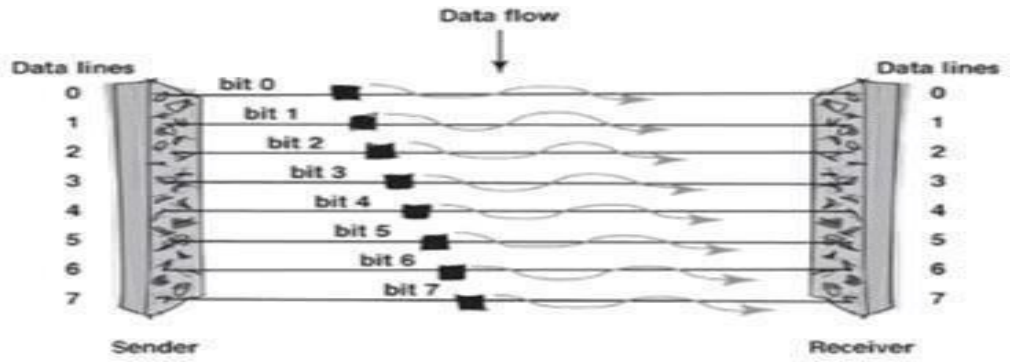
I2C V/S SPI:

I2C	SPI
Speed limit varies from 100kbps, 400kbps, 1mbps, 3.4mbps depending on i2c version.	More than 1mbps, 10mbps till 100mbps can be achieved.
Half duplex synchronous protocol	Full Duplex synchronous protocol
Support Multi master configuration	Multi master configuration is not possible
Acknowledgement at each transfer	No Acknowledgement
Require Two Pins only SDA, SCL	Require separate MISO, MOSI, CLK & CS signal for each slave.
Addition of new device on the bus is easy	Addition of new device on the bus is not much easy a I2C
More Overhead (due to acknowledgement, start, stop)	Less Overhead
Noise sensitivity is high	Less noise sensitivity

PARALLEL COMMUNICATION:

In data transmission, parallel communication is a method of conveying multiple binary digits (bits) simultaneously. It contrasts with communication. The communication channel is the number of electrical conductors used at the physical layer to convey bits.

Parallel communication implies more than one such conductor. For example, an 8-bit parallel channel will convey eight bits (or a byte) simultaneously, whereas a serial channel would convey those same bits sequentially, one at a time. Parallel communication is and always has been widely used within integrated circuits, in peripheral buses, and in memory devices such as RAM.



2. Product level communication interface (External Communication Interface):

The Product level communication interface‟ (External Communication Interface) is responsible for data transfer between the embedded system and other devices or modules

It is classified into two types

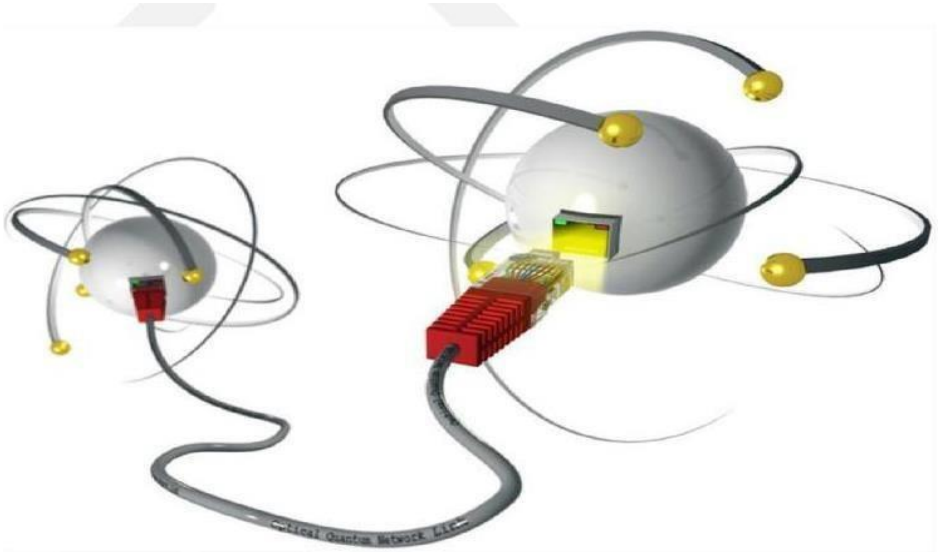
1. Wired communication interface
2. Wireless communication interface:

1. Wired communication interface: Wired communication interface is an interface used to transfer information over a wired network.

It is classified into following types.

1. **RS-232C/RS-422/RS 485**

2. **USB**



RS-232C:

- RS-232 C (Recommended Standard number 232, revision C from the Electronic Industry Association) is a legacy, full duplex, wired, asynchronous serial communication interface
- RS-232 extends the UART communication signals for external data communication.
- UART uses the standard TTL/CMOS logic (Logic „High" corresponds to bit value 1 and Logic „LOW" corresponds to bit value 0) for bit transmission whereas RS232 use the EIA standard for bit transmission.
- As per EIA standard, a logic „0" is represented with voltage between +3 and +25V and a logic „1" is represented with voltage between -3 and -25V.
- In EIA standard, logic „0" is known as „Space" and logic „1" as „Mark".

The RS232 interface define various handshaking and control signals for communication apart from the „Transmit" and „Receive" signal lines for data communication

RS-232 supports two different types of connectors, namely; DB-9: 9-Pin connector and DB-25: 25-Pin connector.

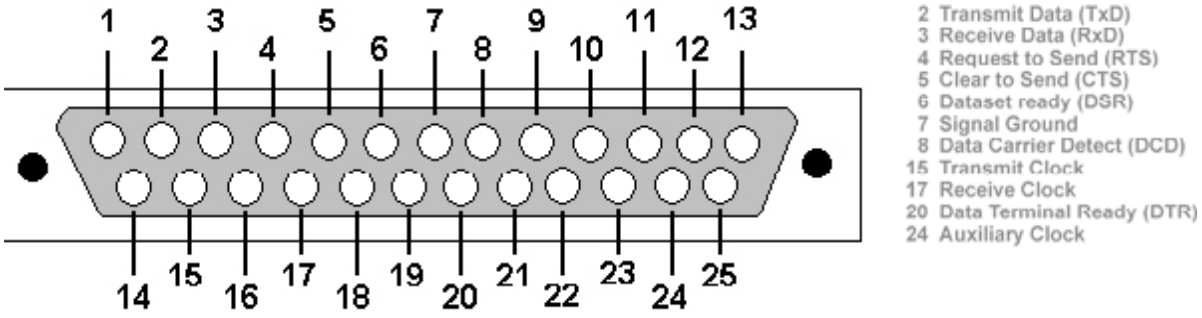
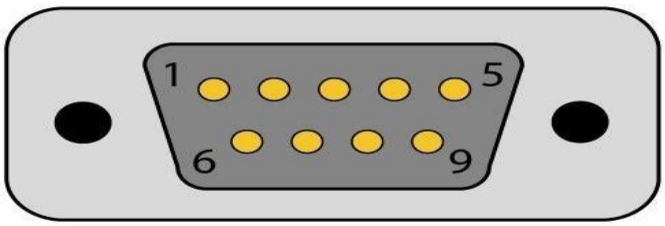


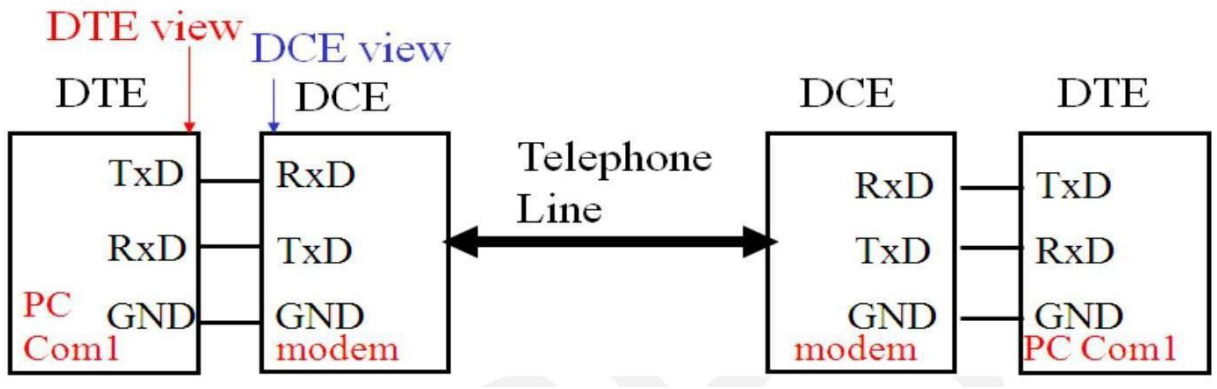
Fig: DB-25:25-Pin connector.



RS232 Pin Out

Pin #	Signal
1	DCD
2	RX
3	TX
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

Fig: DB-9:9-Pin connector.



- RS-232 is a point-to-point communication interface and the devices involved in RS-232 communication are called „Data Terminal Equipment (DTE)" and „Data Communication Equipment (DCE)".
- If no data flow control is required, only TXD and RXD signal lines and ground line (GND) are required for data transmission and reception.
- The RXD pin of DCE should be connected to the TXD pin of DTE and vice versa for proper data transmission.
- If hardware data flow control is required for serial transmission, various control signal lines of the RS-232 connection are used appropriately.
- The control signals are implemented mainly for modem communication and some of them may be irrelevant for other type of devices.
- The Request to Send (RTS) and Clear To Send (CTS) signals co-ordinate the communication between DTE and DCE.
- Whenever the DTE has a data to send, it activates the RTS line and if the DCE is ready to accept the data, it activates the CTS line.
- The Data Terminal Ready (DTR) signal is activated by DTE when it is ready to accept data.
- The Data Set Ready (DSR) is activated by DCE when it is ready for establishing a communication link.
- DTR should be in the activated state before the activation of DSR.
- The Data Carrier Detect (DCD) is used by the DCE to indicate the DTE that a good signal is being received.

- Ring Indicator (RI) is a modem specific signal line for indicating an incoming call on the telephone line.
- As per the EIA standard RS-232 C supports baudrates up to 20Kbps (Upper limit 19.2Kbps).
- The commonly used baudrates by devices are 300bps, 1200bps, 2400bps, 9600bps, 11.52Kbps and 19.2Kbps.
- The maximum operating distance supported in RS-232 communication is 50 feet at the highest supported baudrate.
- Embedded devices contain a UART for serial communication and they generate signal levels conforming to TTL/CMOS logic.
- A level translator IC like MAX 232 from Maxim Dallas semiconductor is used for converting the signal lines from the UART to RS-232 signal lines for communication.
- On the receiving side the received data is converted back to digital logic level by a converter IC.
- Converter chips contain converters for both transmitter and receiver.
- RS-232 uses single ended data transfer and supports only point-to-point communication and not suitable for multi-drop communication.

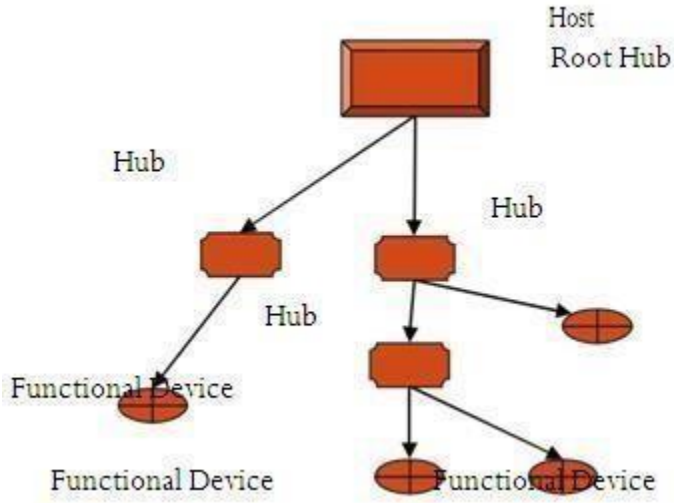
USB (UNIVERSAL SERIAL BUS):

- External Bus Standard.
- Allows connection of peripheral devices.
- Connects Devices such as keyboards, mice, scanners, printers, joysticks, audio devices, disks.
- Facilitates transfers of data at 480 (USB 2.0 only), 12 or 1.5 Mb/s (mega-bits/second).
- Developed by a Special Interest Group including Intel, Microsoft, Compact, DEC, IBM, Northern Telecom and NEC originally in 1994.
- Low-Speed: 10 – 100 kb/s
- 1.5 Mb/s signaling bit rate
- Full-Speed: 500 kb/s – 10 Mb/s 12 Mb/s signaling bit rate
- High-Speed: 400 Mb/s

- 480 Mb/s signaling bit rate
- NRZI with bit stuffing used
- SYNC field present for every packet
- There exist two pre-defined connectors in any USB system - Series "A" and Series "B" Connectors.
- Series "A" cable: Connects USB devices to a hub port.
- Series "B" cable: Connects detachable devices (hot- swappable)

Bus Topology:

- Connects computer to peripheral devices.
- Ultimately intended to replace parallel and serial ports
- Tiered Star Topology
- All devices are linked to a common point referred to as the root hub.
- Specification allows for up to 127 (2^7 -1) different devices.



- Four wire cable serves as interconnect of system - power, ground and two differential signaling lines.
- USB is a polled bus-all transactions are initiated by host.

USB HOST: Device that controls entire system usually a PC of some form. Processes data arriving to and from the USB port.

USB HUB: Tests for new devices and maintains status information of child devices.Serve as repeaters, boosting strength of up and downstream signals. Electrically isolates devices from one another - allowing an expanded number of devices.

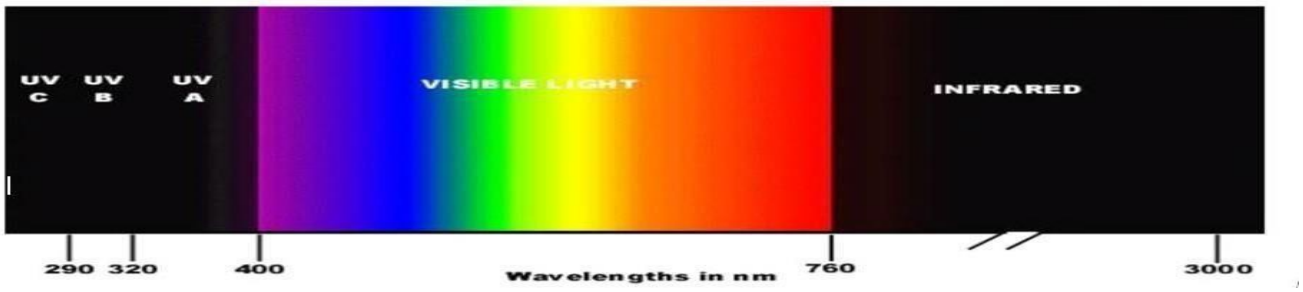
2.Wireless communication interface : Wireless communication interface is an interface used to transmission of information over a distance without help of wires, cables or any other forms of electrical conductors.

They are basically classified into following types

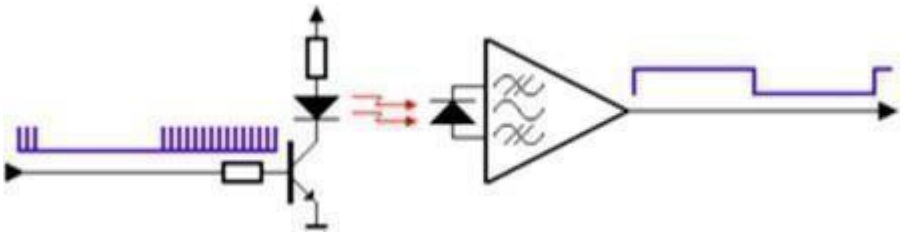
1. **Infrared**
2. **Bluetooth**
3. **Wi-Fi**
4. **Zigbee**
5. **GPRS**

INFRARED:

- Infrared is a certain region in the light spectrum
- Ranges from .7µ to 1000µ or .1mm
- Broken into near, mid, and far infrared
- One step up on the light spectrum from visible light
- Measure of heat



Most of the thermal radiation emitted by objects near room temperature is infrared. Infrared radiation is used in industrial, scientific, and medical applications. Night-vision devices using active near-infrared illumination allow people or animals to be observed without the observer being detected.



IR transmission:

The transmitter of an IR LED inside its circuit, which emits infrared light for every electric pulse given to it. This pulse is generated as a button on the remote is pressed, thus completing the circuit, providing bias to the LED.

The LED on being biased emits light of the wavelength of 940nm as a series of pulses, corresponding to the button pressed. However since along with the IR LED many other sources of infrared light such as us human beings, light bulbs, sun, etc, the transmitted information can be interfered. A solution to this problem is by modulation. The transmitted signal is modulated using a carrier frequency of 38 KHz (or any other frequency between 36 to 46 KHz). The IR LED is made to oscillate at this frequency for the time duration of the pulse. The information or the light signals are pulse width modulated and are contained in the 38 KHz frequency.

IR supports data rates ranging from 9600bits/second to 16Mbps

Serial infrared: 9600bps to 115.2 kbps

Medium infrared: 0.576Mbps to 1.152 Mbps

Fast infrared: 4Mbps

BLUETOOTH:

Bluetooth is a wireless technology standard for short distances (using short-wavelength UHF band from 2.4 to 2.485 GHz)for exchanging data over radio waves in the ISM and mobile devices, and building personal area networks (PANs).Invented by telecom vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS- 232 data cables.

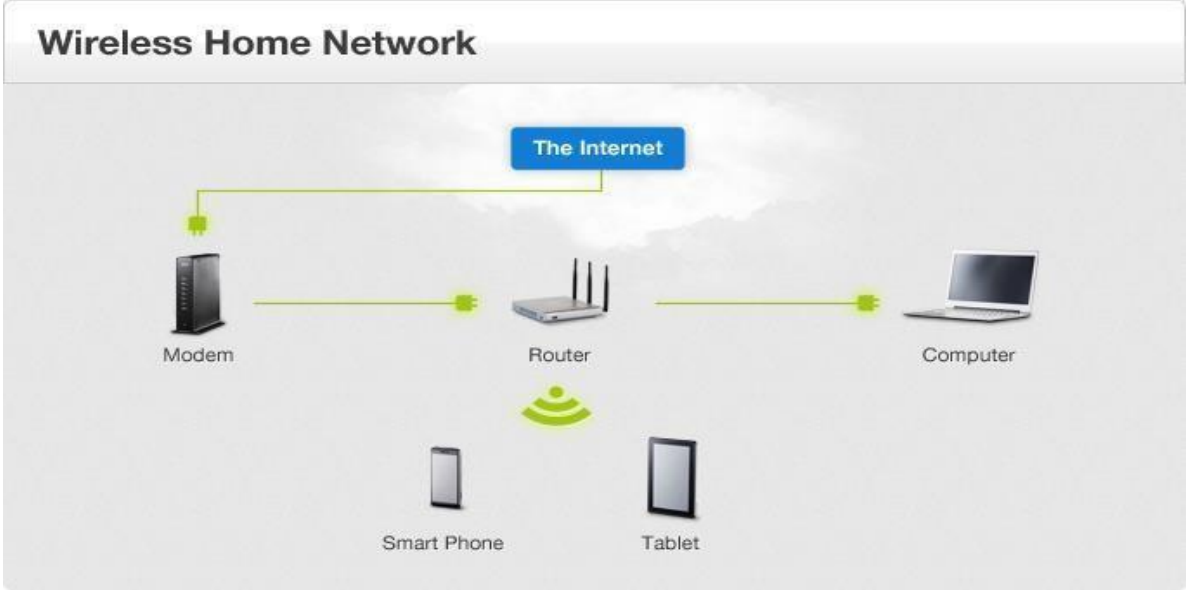
Bluetooth uses a radio technology called frequency- hopping spread spectrum. Bluetooth divides transmitted data into packets, and transmits each packet on one of 79 designated Bluetooth channels. Each channel has a bandwidth of 1 MHz. It usually performs 800 hops per second, with Adaptive Frequency-Hopping (AFH) enabled

Originally, Gaussian frequency-shift keying (GFSK) modulation was the only modulation scheme available. Since the introduction of Bluetooth 2.0+EDR, π/4-DQPSK (Differential Quadrature Phase Shift Keying) and 8DPSK modulation may also be used between compatible devices. Bluetooth is a packet-based protocol with a master- slave structure. One master may communicate with up to seven slaves in a piconet. All devices share the master's clock. Packet exchange is based on the basic clock, defined by the master, which ticks at312.5 µs intervals.

A master BR/EDR Bluetooth device can communicate with a maximum of seven devices in a piconet (an ad-hoc computer network using Bluetooth technology), though not all devices reach this maximum. The devices can switch roles, by agreement, and the slave can become the master (for example, a headset initiating a connection to a phone necessarily begins as master—as initiator of the connection—but may subsequently operate as slave).

Wi-Fi:

- Wi-Fi is the name of a popular wireless networking technology that uses radio waves to provide wireless high-speed Internet and network connections
- Wi-Fi follows the IEEE 802.11 standard
- Wi-Fi is intended for network communication and it supports Internet Protocol (IP) based communication
- Wi-Fi based communications require an intermediate agent called Wi-Fi router/Wireless Access point to manage the communications.
- The Wi-Fi router is responsible for restricting the access to a network, assigning IP address to devices on the network, routing data packets to the intended devices on the network.

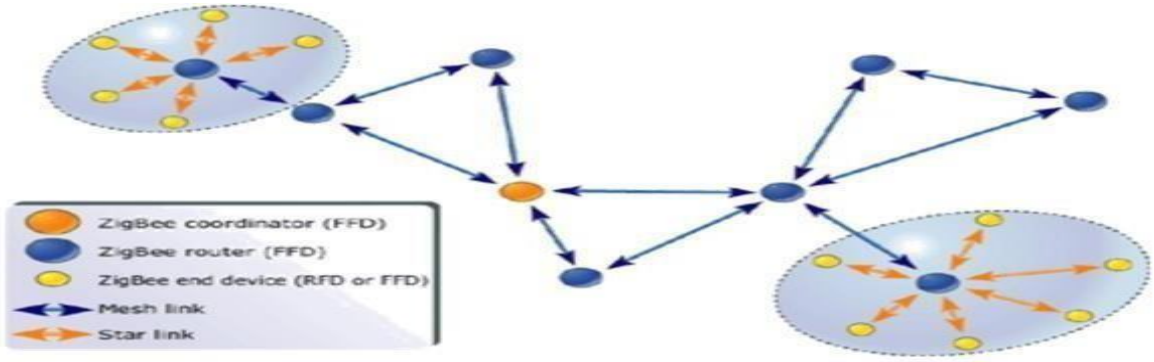


- Wi-Fi enabled devices contain a wireless adaptor for transmitting and receiving data in the form of radio signals through an antenna.
- Wi-Fi operates at 2.4GHZ or 5GHZ of radio spectrum and they co-exist with other ISM band devices like Bluetooth.
- A Wi-Fi network is identified with a Service Set Identifier (SSID). A Wi-Fi device can connect to a network by selecting the SSID of the network and by providing the credentials if the network is security enabled
- Wi-Fi networks implements different security mechanisms for authentication and data transfer.
- Wireless Equivalency Protocol (WEP), Wireless Protected Access (WPA) etc are some of the security mechanisms supported by Wi-Fi networks in data communication.

ZIGBEE:

Zigbee is an IEEE 802.15.4-based specification for a suite of high- level communication protocols used to create personal area networks with small, low-power digital radios, such as for home automation, medical device data collection, and other low-power low-bandwidth needs, designed for small scale projects which need wireless connection.Hence, zigbee is a low-power, low data rate, and close proximity (i.e., personal area) wireless ad hoc network.The technology

defined by the zigbee specification is intended to be simpler and less expensive than other wireless personal area networks (WPANs), such as Bluetooth or Wi-Fi . Applications include wireless light switches, electrical meters with in-home-displays, traffic management systems, and other consumer and industrial equipment that require short-range low- rate wireless data transfer. Its low power consumption limits transmission distances to 10– 100 meters line-of-sight, depending on power output and environmental characteristics. Zigbee devices can transmit data over long distances by passing data through a mesh network of intermediate devices to reach more distant ones.



Zigbee Coordinator: The zigbee coordinator acts as the root of the zigbee network. The ZC is responsible for initiating the Zigbee network and it has the capability to store information about the network.

Zigbee Router: Responsible for passing information from device to another device or to another ZR.

Zigbee end device:End device containing zigbee functionality for data communication. It can talk only with a ZR or ZC and doesn't have the capability to act as a mediator for transferring data from one device to another.

Zigbee supports an operating distance of up to 100 metres at a data rate of 20 to 250 Kbps.

General Packet Radio Service(GPRS):

General Packet Radio Service (**GPRS**) is a packet oriented mobile data service on the 2G and 3G cellular communication system's global system for mobile communications (GSM).GPRS was originally standardized by European Telecommunications Standards Institute (ETSI) GPRS usage is typically charged based on volume of data transferred, contrasting with circuit switched data, which is usually billed per minute of connection time. Sometimes billing time is broken down to every third of a minute. Usage above the bundle cap is charged per megabyte, speed limited, or disallowed.

Services offered:

- GPRS extends the GSM Packet circuit switched data capabilities and makes the following services possible:
- SMS messaging and broadcasting
- "Always on" internet access
- Multimedia messaging service (MMS)
- Push-to-talk over cellular (PoC)
- Instant messaging and presence-wireless village Internet applications for smart devices through wireless application protocol (WAP).
- Point-to-point (P2P) service: inter-networking with the Internet (IP).
- Point-to-multipoint (P2M) service]: point-to- multipoint multicast and point-to-multipoint group calls.

Text Book:-

1. Introduction to Embedded Systems – Shibu K.V Mc Graw Hill

Introduction

The control algorithm (Program instructions) and or the configuration settings that an embedded system developer dumps into the code (Program) memory

- of the embedded system
- It is an un-avoidable part of an embedded system.

The embedded firmware can be developed in various methods like

- Write the program in high level languages like Embedded C/C++ using an Integrated Development Environment (The IDE will contain an editor, compiler, linker, debugger, simulator etc. IDEs are differentfor different family of processors/controllers.
- Write the program in Assembly Language using the Instructions Supported by your application's target processor/controller

Embedded Firmware Design & Development:

- The embedded firmware is responsible for controlling the various peripherals of the embedded hardware and generating response in accordance with the functional requirements of the product.
- The embedded firmware is the master brain of the embedded system.
- The embedded firmware imparts intelligence to an Embedded system.
- It is a onetime process and it can happen at any stage.
- The product starts functioning properly once the intelligence imparted to the product by embedding the firmware in the hardware.
- The product will continue serving the assigned task till hardware breakdown occurs or a corruption in embedded firmware.
- In case of hardware breakdown, the damaged component may need to be replaced and for firmware corruptions the firmware should be re-loaded, to bring back the embedded product to the normal functioning.
- The embedded firmware is usually stored in a permanent memory (ROM)

and it is non alterable by end users.

- Designing Embedded firmware requires understanding of the particular embedded product hardware, like various component interfacing, memory map details, I/O port details, configuration and register details of various hardware chips used and some programming language (either low level Assembly Language or High level language like C/C++ or a combination of the two)
- The embedded firmware development process starts with the conversion of the firmware requirements into a program model using various modeling tools.
- The firmware design approaches for embedded product is purely dependent on the complexity of the functions to be performed and speed of operation required.
- There exist two basic approaches for the design and implementation of embedded firmware, namely;
  - **The Super loop based approach**
  - **The Embedded Operating System based approach**
- The decision on which approach needs to be adopted for firmware development is purely dependent on the complexity and system requirements

1. Embedded firmware Design Approaches – The Super loop:

- The Super loop based firmware development approach is Suitable for applications that are not time critical and where the response time is not so important (Embedded systems where missing deadlines are acceptable).
- It is very similar to a conventional procedural programming where the code is executed task by task

  The tasks are executed in a never ending loop.
-

- The task listed on top on the program code is executed first and the tasks just below the top are executed after completing the first task

- A typical super loop implementation will look like:

    1. Configure the common parameters and perform initialization for various hardware components memory, registers etc.
    2. Start the first task and execute it
    3. Execute the second task
    4. Execute the next task
    5. :
    6. :
    7. Execute the last defined task
    8. Jump back to the first task and follow the same flow.

The 'C' program code for the super loop is given below

```
void main ()
{
Configurations ();
Initializations ();
while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

Pros:

- Doesn't require an Operating System for task scheduling and monitoring and free from OS related overheads
- Simple and straight forward design
- Reduced memory footprint

Cons:

- Non Real time in execution behavior (As the number of tasks increases the frequency at which a task gets CPU time for execution also increases)
- Any issues in any task execution may affect the functioning of the product (This can be effectively tackled by using Watch Dog Timers for task execution monitoring)

Enhancements:

- Combine Super loop based technique with interrupts
- Execute the tasks (like keyboard handling) which require Real time attention as Interrupt Service routines.

2. Embedded firmware Design Approaches – Embedded OS based Approach:

- The embedded device contains an Embedded Operating System which can be one of:
  - **A Real Time Operating System (RTOS)**
  - **A Customized General Purpose Operating System (GPOS)**

- The Embedded OS is responsible for scheduling the execution of user tasks and the allocation of system resources among multiple tasks
- It Involves lot of OS related overheads apart from managing and executing user defined tasks
- Microsoft® Windows XP Embedded is an example of GPOS for embedded devices
- Point of Sale (PoS) terminals, Gaming Stations, Tablet PCs etc are examples of embedded devices running on embedded GPOSs
- '*Windows CE*', '*Windows Mobile*','*QNX*', '*VxWorks*', '*ThreadX*', '*MicroC/OS-II*', '*Embedded Linux*', '*Symbian*' etc are examples of RTOSs employed in Embedded Product development
- Mobile Phones, PDAs, Flight Control Systems etc are examples of embedded devices that runs on RTOSs

Embedded firmware Development Languages/Options

- **Assembly Language**
- **High Level Language**
  - Subset of C (Embedded C)
  - Subset of C++ (Embedded C++)
  - Any other high level language with supported Cross-compiler
- **Mix of Assembly & High level Language**
  - Mixing High Level Language (Like C) with Assembly Code
  - Mixing Assembly code with High Level Language (Like C)
  - Inline Assembly

1. Embedded firmware Development Languages/Options – Assembly Language

- '***Assembly Language***' is the human readable notation of '***machine language***'
- 'M***achine language***' is a processor understandable language
- Machine language is a binary representation and it consists of 1s and 0s
- Assembly language and machine languages are processor/controller dependent
- An Assembly language program written for one processor/controller family will not work with others
- Assembly language programming is the process of writing processor specific machine code in mnemonic form, converting the mnemonics into actual processor instructions (machine language) and associated data using an assembler
- The general format of an assembly language instruction is an Opcode followed by Operands
- The Opcode tells the processor/controller what to do and the Operandsprovide the data and information required to perform the action specified by the opcode
- It is not necessary that all opcode should have Operands following them. Some of the Opcode implicitly contains the operand and in such situation no operand is required. The operand may be a single operand, dual operand or more

The 8051 Assembly Instruction

MOV A, #30

Moves decimal value 30 to the 8051 Accumulator register. Here *MOV A* is the Opcode and 30 is the operand (single operand). The same instruction when written in machine language will look like

01110100 00011110

The first 8 bit binary value 01110100 represents the opcode *MOV A* and the second 8 bit binary value 00011110 represents the operand 30.

- Assembly language instructions are written one per line
- A machine code program consists of a sequence of assembly language instructions, where each statement contains a mnemonic (Opcode + Operand)
- Each line of an assembly language program is split into four fields as:

LABEL OPCODE OPERAND COMMENTS

- LABEL is an optional field. A 'LABEL' is an identifier used extensively in programs to reduce the reliance on programmers for remembering where data or code is located. LABEL is commonly used for representing

  ❖ A memory location, address of a program, sub-routine, code portion etc.

  ❖ The maximum length of a label differs between assemblers. Assemblers insist strict formats for labeling. Labels are always suffixed by a colon and begin with a valid character. Labels can contain number from 0 to 9 and special character _ (underscore).

```
;#################################################################
;   SUBROUTINE FOR GENERATING DELAY
;   DELAY PARAMETR PASSED THROUGH REGISTER R1
;   RETURN VALUE NONE,REGISTERS USED: R0,  R1
;#################################################################
#####   DELAY:    MOV R0, #255     ; Load Register R0 with 255

          DJNZ R1, DELAY; Decrement R1 andloop till      R1= 0

          RET               ; Return to calling program
```

- The symbol ; represents the start of a comment. Assembler ignores the text in a line after the ; symbol while assembling the program
- DELAY is a label for representing the start address of the memory location where the piece of code is located in code memory
- The above piece of code can be executed by giving the label DELAY as part of the instruction. E.g. LCALL DELAY; LMP DELAY

2. Assembly Language – Source File to Hex File Translation:

- The Assembly language program written in assembly code is saved as *.asm* (Assembly file) file or a *.src* (source) file or a format supported by the assembler
- Similar to 'C' and other high level language programming, it is possible to have multiple source files called modules in assembly language programming. Each module is represented by a '***.asm***' or '***.src***' file or the assembler supported file format similar to the '***.c***' files in C programming
- The software utility called 'Assembler' performs the translation of assembly code to machine code
- The assemblers for different family of target machines are different. A51 Macro Assembler from Keil software is a popular assembler for the 8051 family micro controller

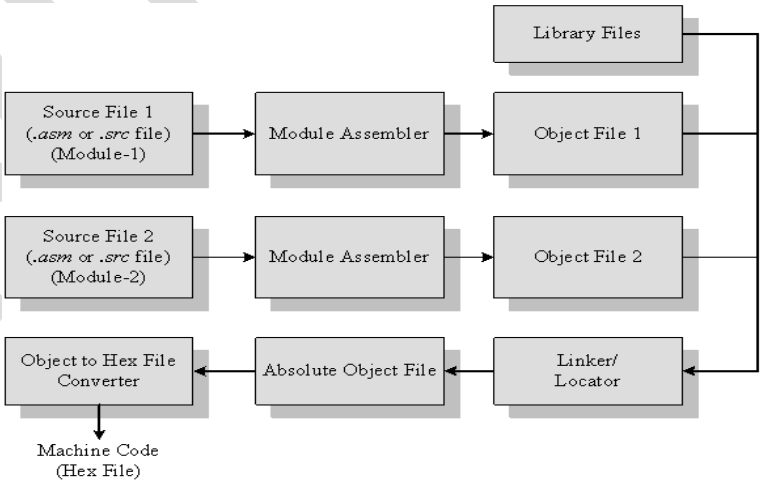


Figure 5: Assembly Language to machine language conversion process

- Each source file can be assembled separately to examine the syntax errors and incorrect assembly instructions
- Assembling of each source file generates a corresponding object file. The object file does not contain the absolute address of where the generated code needs to be placed (a re-locatable code) on the program memory
- The software program called linker/locater is responsible for assigning absolute address to object files during the linking process
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory
- A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file)

Advantages:

✶ **1.Efficient Code Memory & Data Memory Usage (Memory Optimization):**

- ➢ The developer is well aware of the target processor architecture and memory organization, so optimized code can be written for performing operations.
- ➢ This leads to less utilization of code memory and efficient utilization of data memory.

✶ **2.High Performance:**

- ➢ Optimized code not only improves the code memory usage but also improves the total system performance.
- ➢ Through effective assembly coding, optimum performance can be achieved for target processor.

✶ **3.Low level Hardware Access:**

- ➢ Most of the code for low level programming like accessing external device specific registers from OS kernel ,device drivers, and low level interrupt routines, etc are making use of direct assembly coding.

✶ 4.Code Reverse Engineering:

- It is the process of understanding the technology behind a product by extracting the information from the finished product.
- It can easily be converted into assembly code using a dis-assembler program for the target machine.

Drawbacks:

✶ 1.High Development time:

- The developer takes lot of time to study about architecture ,memory organization, addressing modes and instruction set of target processor/controller.
- More lines of assembly code is required for performing a simple action.

✶ 2.Developer dependency:

- There is no common written rule for developing assembly language based applications.

✶ 3.Non portable:

- Target applications written in assembly instructions are valid only for that particular family of processors and cannot be re-used for another target processors/controllers.
- If the target processor/controller changes, a complete re-writing of the application using assembly language for new target processor/controller is required.

2. Embedded firmware Development Languages/Options – High Level Language

- The embedded firmware is written in any high level language like C, C++
- A software utility called ‘cross-compiler’ converts the high level language to target processor specific machine code

- The cross-compilation of each module generates a corresponding object file. The object file does not contain the absolute address of where the generated code needs to be placed (a re-locatable code) on the program memory
- The software program called linker/locater is responsible for assigning absolute address to object files during the linking process
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory
- A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file)

Embedded firmware Development Languages/Options – High Level Language – Source File to Hex File Translation

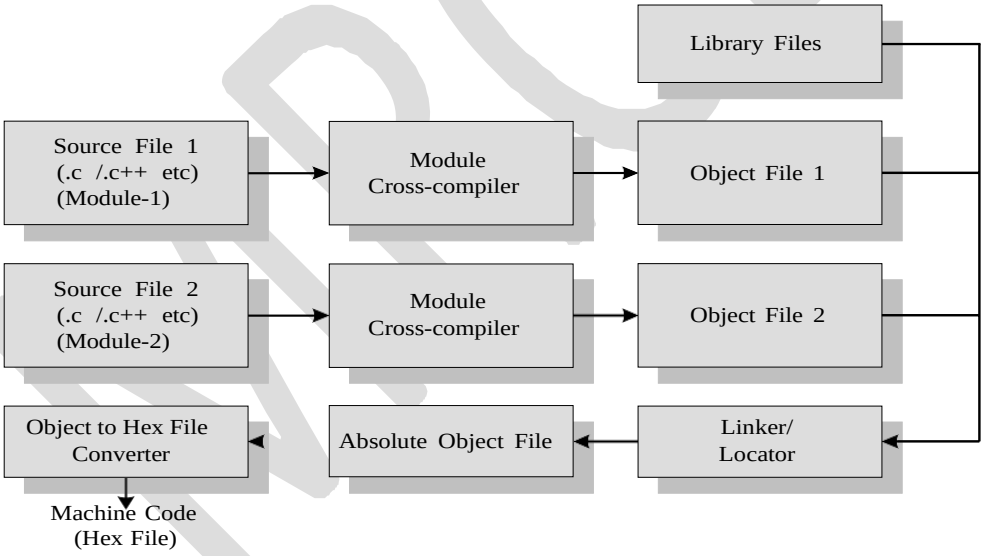


Figure 6: High level language to machine language conversion process

Advantages:

- **Reduced Development time:** Developer requires less or little knowledge on internal hardware details and architecture of the target processor/Controller.
- **Developer independency:** The syntax used by most of the high level languages are universal and a program written high level can easily understand by a second person knowing the syntax of the language
- **Portability:** An Application written in high level language for particular target processor /controller can be easily be converted to another target processor/controller specific application with little or less effort

Drawbacks:

- The cross compilers may not be efficient in generating the optimized target processor specific instructions.
- Target images created by such compilers may be messy and non-optimized in terms of performance as well as code size.
- The investment required for high level language based development tools (IDE) is high compared to Assembly Language based firmware development tools.

Embedded firmware Development Languages/Options – Mixing of Assembly Language with High Level Language

- Embedded firmware development may require the mixing of Assembly Language with high level language or vice versa.
- Interrupt handling, Source code is already available in high level language\Assembly Language etc are examples

- High Level language and low level language can be mixed in three different ways
  - ✓ Mixing Assembly Language with High level language like 'C'
  - ✓ Mixing High level language like 'C' with Assembly Language
  - ✓ In line Assembly
- The passing of parameters and return values between the high level and low level language is cross-compiler specific

1. Mixing Assembly Language with High level language like 'C' (Assembly Language with 'C'):

- Assembly routines are mixed with 'C' in situations where the entire program is written in 'C' and the cross compiler in use do not have built in support for implementing certain features like ISR.
- If the programmer wants to take advantage of the speed and optimized code offered by the machine code generated by hand written assembly rather than cross compiler generated machine code.
- For accessing certain low level hardware ,the timing specifications may be very critical and cross compiler generated machine code may not be able to offer the required time specifications accurately.
- Writing the hardware/peripheral access routine in processor/controller specific assembly language and invoking it from 'C' is the most advised method.
- Mixing 'C' and assembly is little complicated.
- The programmer must be aware of how to pass parameters from the 'C' routine to assembly and values returned from assembly routine to 'C' and how Assembly routine is invoked from the 'C' code.

- Passing parameter to the assembly routine and returning values from the assembly routine to the caller 'C' function and the method of invoking the assembly routine from 'C' code is cross compiler dependent.
- There is no universal written rule for purpose.
- We can get this information from documentation of the cross compiler.
- Different cross compilers implement these features in different ways depending on GPRs and memory supported by target processor/controller

2. Mixing High level language like 'C' with Assembly Language ('C' with Assembly Language)

- The source code is already available in assembly language and routinewritten in a high level language needs to be included to the existing code.
- The entire source code is planned in Assembly code for various reasons like optimized code, optimal performance, efficient code memory utilization and proven expertise in handling the assembly.
- The functions written in 'C' use parameter passing to the function and returns values to the calling functions.
- The programmer must be aware of how parameters are passed to the function and how values returned from the function and how function is invoked from the assembly language environment.
- Passing parameter to the function and returning values from the function using CPU registers , stack memory and fixed memory.
- Its implementation is cross compiler dependent and varies across compilers.

3.In line Assembly:

- Inline assembly is another technique for inserting the target processor/controller specific assembly instructions at any location of source code written in high level language 'C'
- Inline Assembly avoids the delay in calling an assembly routine from a 'C' code.
- Special keywords are used to indicate the start and end of Assembly instructions
- ***E.g #pragma asm***

  Mov A,#13H

  #pragma ensasm
- Keil C51 uses the keywords *#pragma asm* and *#pragma endasm* to indicate a block of code written in assembly.

Text Books:

1. Introduction to Embedded Systems – Shibu K.V Mc Graw Hill

2. Embedded System Design-Raj Kamal TMH

EMBEDDED PROGRAMMING

Assembly language is introduced for providing **mnemonics** or symbols for the machine level code instructions. Assembly language program is consisting of mnemonics that is translated into machine code. A program that is used for this conversion is known as **assembler**.

Assembly language is also called as low-level language because it directly works with the internal structure of CPU. For programming in assembly language, a programmer must have the knowledge of all the registers in a CPU.

Different programming languages like C, C++, Java and various other languages are called as high-level languages because they are not dealing with the internal details of CPU.

In contrast, an assembler is used to translate an assembly language program into machine code (sometimes also called **object code** or **opcode**). Similarly, a compiler translates a high-level language into machine code. For example, to write a program in C language, one must use a C compiler to translate the program into machine language.

Structure of Assembly Language

An assembly language program is a series of statements, which are either assembly language instructions such as ADD and MOV, or statements called **directives**.

An **instruction** tells the CPU what to do, while a **directive** (also called **pseudo-instructions**) gives instruction to the assembler. For example, ADD and MOV instructions are commands which the CPU runs, while ORG and END are assembler directives. The assembler places the opcode to the memory location 0 when the ORG directive is used, while END indicates to the end of the source code. A program language instruction consists of the following four fields –

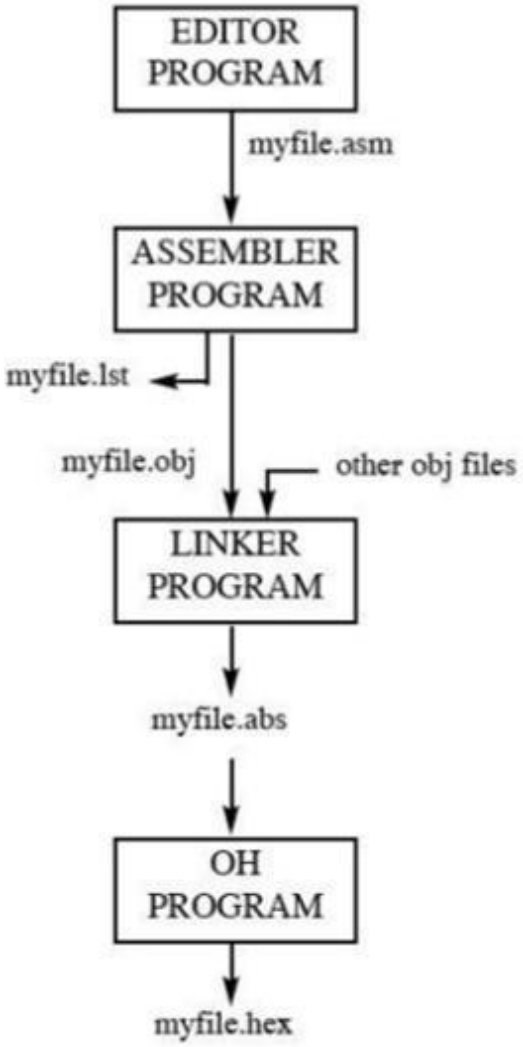
	[label:]	mnemonics	[operands]	[;comment]
1.	0000		ORG 0H	;start (origin) at location 0
2.	0000	7D25	MOV R5,#25H	;load 25H into R5
3.	0002	7F34	MOV R7,#34H	;load 34H into R7
4.	0004	7400	MOV A,#0	;load 0 into A
5.	0006	2D	ADD A,R5	;add contents of R5 to A
6.	0007	2F	ADD A,R7	;add contents of R7 to A
7.	0008	2412	ADD A,#12H	;add to A value 12 H
8.	000A	80FE	HERE: SJMP HERE	;stay in this loop
9.	000C		END	;end of asm source file

A square bracket ([ ]) indicates that the field is optional.

- The **label field** allows the program to refer to a line of code by name. The label fields cannot exceed a certain number of characters.
- The **mnemonics** and **operands fields** together perform the real work of the program and accomplish the tasks. Statements like ADD A , C & MOV C, #68 where ADD and MOV are the mnemonics, which produce opcodes ; "A, C" and "C, #68" are operands. These two fields could contain directives. Directives do not generate machine code and are used only by the assembler, whereas instructions are translated into machine code for the CPU to execute.
- The **comment field** begins with a semicolon which is a comment indicator.
- Notice the Label "HERE" in the program. Any label which refers to an instruction should be followed by a colon.

Assembling and Running of 8051 Program

Let's see the steps for creating, assembling and running an assembly language program are as follows:



- **Editor Program** : At first, we use an editor for type in a program. Editors like MS-DOS program that comes with all Microsoft operating systems can be used for creating or edit a program. The editor produces an ASCII file. The ?asm? extension for a source file is used by an assembler during next step.
- **Assembler Program**: The "asm" source file contain the code created in Step 1. It is transferred to an 8051 assembler. The assembler is used for converting the assembly language instructions into machine code instructions and it produced the **.obj file (object file) and .lst file (list file)**. It is also called as source file because some assembler requires that this file must have "src" extension.
- **Linker Program**: The linker program is used for generating one or more object files and produces an absolute object file with an extension "abs".
- **OH Program**: The OH program fetches the "abs" file and fed it to a program called "OH". OH is called as object to hex converter it creates a file with an extension "hex" thatis ready for burn in to the ROM.

Labels in assembly Language

All labels used in assembly language follow the certain rules as given below:

- Each label name should be unique. The name used as label in assembly language programming consist of alphabetic letters in both lowercase and uppercase, numbers from 0 to 9, and special characters such as at the rate (@), question mark (?), underscore(_), and dollar ($) etc.
- Reserved words are not allowed to be used as a label in the program. For example, MOV and ADD words are reserved words.
- The first character must be an alphabetical character, it cannot be a number.

Data Type

The 8051 microcontroller contains a single data type of 8-bits, and each register is also of 8-bits size. The programmer has to break down data larger than 8-bits (00 to FFH, or to 255 in decimal) so that it can be processed by the CPU.

DB (Define Byte)

The DB directive is the most widely used data directive in the assembler. It is used to define the 8-bit data. It can also be used to define decimal, binary, hex, or ASCII formats data. For decimal, the "D" after the decimal number is optional, but it is required for "B" (binary) and "Hl" (hexadecimal).

To indicate ASCII, simply place the characters in quotation marks ('like this'). The assembler generates ASCII code for the numbers/characters automatically. The DB directive is the only directive that can be used to define ASCII strings larger than two characters; therefore, it should be used for all the ASCII data definitions. Some examples of DB are given below −

```
        ORG  500H
DATA1:  DB   28                    ;DECIMAL (1C in hex)
DATA2:  DB   00110101B             ;BINARY (35 in hex)
DATA3:  DB   39H                   ;HEX
        ORG  510H
DATA4:  DB   "2591"                ;ASCII  NUMBERS
        ORG  520H
DATA6:  DA   "MY NAME IS Michael"  ;ASCII CHARACTERS
```

Either single or double quotes can be used around ASCII strings. DB is also used to allocate memory in byte-sized chunks.

Assembler Directives

Some of the directives of 8051 are as follows −

- **ORG (origin)** − The origin directive is used to indicate the beginning of the address. It takes the numbers in hexa or decimal format. If H is provided after the number, the number is treated as hexa, otherwise decimal. The assembler converts the decimal numberto hexa.
- **EQU (equate)** − It is used to define a constant without occupying a memory location. EQU associates a constant value with a data label so that the label appears in the program,its constant value will be substituted for the label. While executing the instruction "MOV R3, #COUNT", the register R3 will be loaded with the value 25 (notice the # sign). The advantage of using EQU is that the programmer can change it once and the assembler will change all of its occurrences; the programmer does not have to search the entire program.
- **END directive** − It indicates the end of the source (asm) file. The END directive is the last line of the program; anything after the END directive is ignored by the assembler.

Labels in assembly Language

All labels used in assembly language follow the certain rules as given below:

- Each label name should be unique. The name used as label in assembly language programming consist of alphabetic letters in both lowercase and uppercase, numbers from 0 to 9, and special characters such as at the rate (@), question mark (?), underscore(_), and dollar ($) etc.
- Reserved words are not allowed to be used as a label in the program. For example, MOV and ADD words are reserved words.

- The first character must be an alphabetical character, it cannot be a number.

Embedded Systems - Tools & Peripherals

Compilers and Assemblers

Compiler

A compiler is a computer program (or a set of programs) that transforms the source code written in a programming language (the source language) into another computer language (normally binary format). The most common reason for conversion is to create an executable program. The name "compiler" is primarily used for programs that translate the source code from a highlevel programming language to a low-level language (e.g., assembly language or machine code).

Cross-Compiler

If the compiled program can run on a computer having different CPU or operating system than the computer on which the compiler compiled the program, then that compiler is known as a cross-compiler.

Decompiler

A program that can translate a program from a low-level language to a high-level language is called a decompiler.

Language Converter

A program that translates programs written in different high-level languages is normally called a language translator, source to source translator, or language converter.

A compiler is likely to perform the following operations −

- Preprocessing
- Parsing
- Semantic Analysis (Syntax-directed translation)
- Code generation
- Code optimization

Assemblers

An assembler is a program that takes basic computer instructions (called as assembly language) and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. An assembler creates object code by translating assembly instruction mnemonics into opcodes, resolving symbolic names to memory locations. Assembly language uses a mnemonic to represent each low-level machine operation (opcode).

Debugging Tools in an Embedded System

Debugging is a methodical process to find and reduce the number of bugs in a computer program or a piece of electronic hardware, so that it works as expected. Debugging is difficult when subsystems are tightly coupled, because a small change in one subsystem can create bugs in another. The debugging tools used in embedded systems differ greatly in terms of their development time and debugging features. We will discuss here the following debugging tools −

- Simulators
- Microcontroller starter kits
- Emulator

Simulators

Code is tested for the MCU / system by simulating it on the host computer used for code development. Simulators try to model the behavior of the complete microcontroller in software.

Functions of Simulators

A simulator performs the following functions −

- Defines the processor or processing device family as well as its various versions for the target system.
- Monitors the detailed information of a source code part with labels and symbolic arguments as the execution goes on for each single step.
- Provides the status of RAM and simulated ports of the target system for each single step execution.
- Monitors system response and determines throughput.
- Provides trace of the output of contents of program counter versus the processor registers.
- Provides the detailed meaning of the present command.
- Monitors the detailed information of the simulator commands as these are entered from the keyboard or selected from the menu.
- Supports the conditions (up to 8 or 16 or 32 conditions) and unconditional breakpoints.
- Provides breakpoints and the trace which are together the important testing and debugging tool.
- Facilitates synchronizing the internal peripherals and delays.