

ANNAMACHARYA **INSTITUTE OF TECHNOLOGY AND SCIENCES** **(AUTONOMOUS)**

Approved by AICTE, New Delhi & Permanent Affiliation to JNTUA, Anantapur.

Three B. Tech Programmes (CSE , ECE & CE) are accredited by NBA, New Delhi, Accredited by NAAC with 'A' Grade , Bangalore.

A-grade awarded by AP Knowledge Mission. Recognized under sections 2(f) & 12(B) of UGC Act 1956.

Venkatapuram Village, Renigunta Mandal, Tirupati, Andhra Pradesh-517520.

Department of Computer Science and Engineering(CIC)



Academic Year 2023-24

III. B.Tech I Semester

**EMBEDDED SYSTEMS AND INTERNET OF
THINGS**

(20APC3616)

Prepared By

Mrs C Hemavathy
Assistant Professor
Department of CSE, AITS
hemasathwik1@gmail.com

UNIT-I

Introduction to Embedded Systems

Embedded System Introduction:

We live in an era where pervasive (spreading) computing exists everywhere, right from a small handheld device such as a mobile phone to the electronic control units within automobiles or avionics. Today, large volumes of information is getting processed and communicated over the Internet every microsecond. Buzz words such as Cloud Computing, Big Data Mining and Internet of Things are everywhere.

There are two broad classifications of computing systems - general purpose computing system and embedded computing systems. If we define these in simple words, general purpose computing systems are those used in desktop or laptop computers, which can process several different applications. An embedded system refers to any device that has some computational intelligence in it. It is generally used as a standalone system that repeatedly performs a specific task or as part of a large system to perform multiple tasks with the required hardware and software embedded within. Systems used in printers, washing machines, mp3 players; CT scan machines etc. are great examples of embedded systems.

An embedded system is a constrained system and its design goals vary from a general purpose system. The constraints are: high performance, low power consumption, small size and low cost of the system.

The basic components of an embedded system include hardware, software and some mechanical parts. Embedded hardware includes a processing unit, block of memory and I/O sub-unit which are called as the system resources. The embedded software can be thought of as the application software in a small computing system or both the system and the application software in case of a large complex system. The system software mentioned here is the real time operating system (RTOS) used to manage the usage of system resources by application software.

What is an Embedded System?

- An embedded system is a system that has software embedded into computer-hardware, which makes a system dedicated for an application or specific part of an application or product or part of a larger system.
- An embedded system is designed to do a specific/few task only.

❖ **Examples:**

- A Washing machine can only wash clothes.
- A Digital camera can only capture the images.
- An Air conditioner can control the temperature in the room.

Host and Target concept:

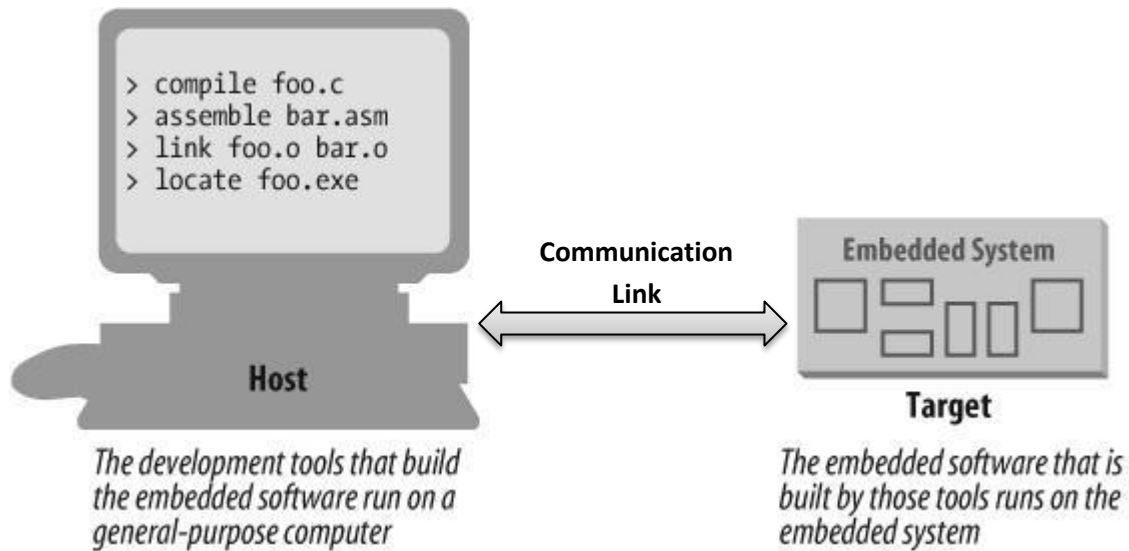


Figure: Embedded System using Host and Target Machine

Performance of Host machine:

The application program developed runs on the host computer. The host computer is also called as Development Platform. It is a general purpose computer. It has a higher capability processor and more memory. It has different input and output devices. The compiler, assembler, linker, and locator run on a host computer rather than on the embedded system itself. These tools are extremely popular with embedded software developers because they are freely available (even the source code is free) and support many of the most popular embedded processors. It contains many development tools to create the output binary image. Once a program has been written, compiled, assembled and linked, it is moved to the target platform.

Performance of Target machine:

The output binary image is executed on the target hardware platform. It consists of two entities - the target hardware (processor) and runtime environment (OS). It is needed only for final output. It is different from the development platform and it does not contain any development tools.

Embedded Applications:

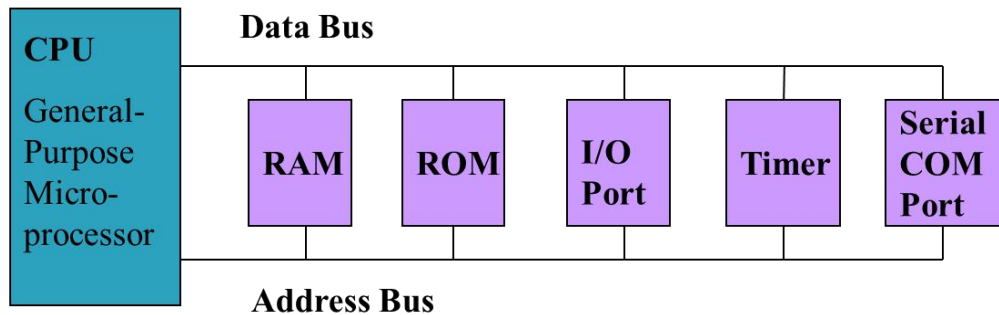
Where Embedded Systems?

- Avionics: Navigation systems, Inertial Guidance Systems, GPS Receivers etc
- Automotives: Automatic Breaking System, Air Bag System, Ignition Systems etc.

- Consumer Electronics: MP3 Players, Mobile Phones, Video Game Consoles, Digital Cameras, Set top Boxes, Camcorders etc.
- Telecommunication: Cell Phones, Telephone Switches etc.
- Medical Systems: ECG, EEG, MRI, CT scan, BP Monitors, etc.
- Military Applications
- Industries
- Home Appliances: AC, TV, DVD Players, Washing machine, Refrigerators, Microwave Oven, etc.
- Computer Peripherals: Printers, Scanners, Fax Machines.
- Computer Networking Systems: Network Routers, Switches, Hubs etc.
- Security Systems: Intruder Detection Alarms, CC Cameras, Fire Alarms
- Measurement & Instrumentation: Digital Multimeters, Digital CRO's, Logic Analyzers, PLC Systems.
- Banking and Retail: ATM, Currency Counters.
- Card readers: Barcode, Smart Card Readers, Hand Held Devices.
- And Many More....



Features and Architecture Considerations for Embedded Systems:



CPU/Processor: It act as a brain to the system, it is used to execute the program and to sending and receiving signals. The processing unit could be a microprocessor, a microcontroller, embedded processor, DSP, ASIC or FPGA selected for an embedded system based on the application requirements.

ROM for the program: Nonvolatile (read-only memory, ROM); meaning that it retains its contents even when power is off. It also called Program memory. In embedded systems, the application program after being compiled is saved in the ROM. The processing unit accesses the ROM to fetch instructions sequentially and executes them within the CPU. There are different categories of ROM such as: programmable read only memory (PROM), erasable programmable read only memory (EPROM), electrically erasable programmable read only memory (EEPROM) etc. There is also flash memory which is the updated version of EEPROM and extensively used in embedded systems.

RAM for data: Random access memory (RAM) is volatile i.e. it does not retain the contents after the power goes off. It is used as the data memory in an embedded system. It holds the variables declared in the program, the stack and intermediate data or results during program run time. The Processing unit accesses the RAM for instruction execution to save or retrieve data. There are different variations of RAM such as: static RAM (SRAM), dynamic RAM (DRAM), pseudo static RAM (PSRAM), non-volatile RAM (NVRAM), synchronous DRAM, (SDRAM) etc.

I/O PORTS: To provide digital communication with the outside world. These Ports are two types Parallel Port and Serial Port, Parallel are used to communicate with Parallel I/O devices like LEDs, LCDs, 7-Segment Displays, Keypads etc., Serial Port is used to communicate with Serial devices like Bluetooth module, Wi-Fi, Zigbee, GSM Modules, PCs etc.

Address and Data buses: To link these subsystems to transfer data and instructions. The system bus consists of three different bus systems: address bus, data bus and control bus. Processor sends the address of the destination through the address bus. So address bus is unidirectional from processor to the external end. Data can be sent or received from any unit to any other unit in the diagram. So data bus is bidirectional. Control bus is basically a group of control signals from the processing unit to the external units

Timers: Timers are a fundamental concept in embedded systems and they have many use cases such as creating accurate delays, executing a periodic task, implementing a PWM (Pulse With Modulation) output or capturing the elapsed time between two events, to vary the

speed of data transfer rate i.e. Baud rate (bps-bits per second) in case serial communication etc.

Clock: To keep the whole system synchronized. It may be generated internally or obtained from a crystal or external source; modern MCUs offer considerable choice of clocks.

Watchdog timer: This is a safety feature, which resets the processor if the program becomes stuck in an infinite loop/hang.

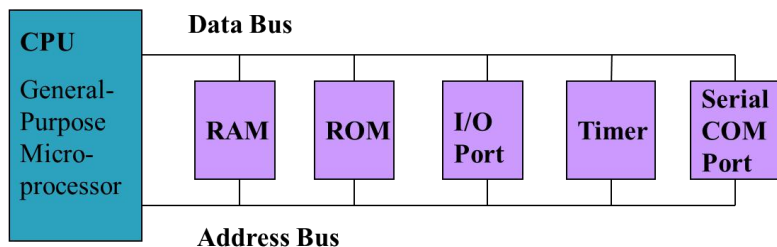
Communication interfaces: A wide choice of interfaces is available to exchange information with another IC or System. They include Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I²C or IIC), Universal Serial Bus (USB), Controller Area Network (CAN), Ethernet, and many others.

Non-volatile Memory for data: This is used to store data whose value must be retained even when power is removed. Serial numbers for identification and network addresses are two obvious candidates.

Microprocessor vs. Microcontroller

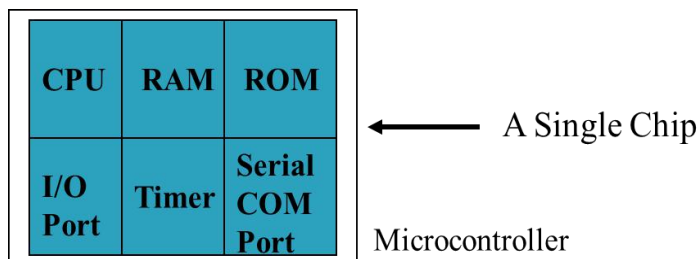
Microprocessor Based Embedded System:

- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example: Intel's 8085,8086, Motorola's 680xx



Microcontroller Based Embedded System:

- A Small Computer or System on Chip (SoC)
- On-chip RAM, ROM, I/O ports...
- Example: TI's MSP430, Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC 16X



S.No.	Microprocessor	Microcontroller
1	Contains Only CPU (μP); RAM, ROM, I/O Ports, Serial Port, Timers are Separately Interfaced	CPU (μP), RAM, ROM, I/O Ports, Serial Port, Timers are all on a Single Chip.
2	It has Many Instructions to Move data between Memory and μP	It has One or Two Instructions to Move data between Memory and μC
3	It has Few Bit Manipulation Instructions	It has Many Bit Manipulation Instructions
4	It has Less Number of Multifunctional Pins	It has More Number of Multifunctional Pins
5	General Purpose	Single (Specific) Purpose
6	High Speed	Low Speed
7	High Power Consumption	Low Power Consumption
8	μP Based System Requires More Hardware & High Cost	μC Based System Requires Less Hardware & Less Cost
9	Designer can decide on the amount of ROM, RAM and I/O ports.	Fixed amount of on-chip ROM, RAM, I/O ports

Embedded Processor and their types:

Microprocessor

Microprocessor is a programmable digital device which has high computational capability to run a number of applications in general purpose systems. It does not have memory or I/O ports built within its architecture. So, these devices need to be added externally to make a system functional. In embedded systems, the design is constrained with limited memory and I/O features. So microprocessors are used where system capability needs to be expanded by adding external memory and I/O.

Microcontroller

A microcontroller has a specific amount of program and data memory, as well as I/O ports built within the architecture along with the CPU core, making it a complete system. As a result, most embedded systems are microcontroller based, where are used to run one or limited number of applications.

Embedded Processor

Embedded processors are specifically designed for embedded systems to meet design constraints. They have the potential to handle multitasking applications. The performance and power efficiency requirements of embedded systems are satisfied by the use of embedded processors.

DSP

Digital signal processors (DSP) are used for signal processing applications such as voice or video compression, data acquisition, image processing or noise and echo cancellation.

ASIC

Application specific integrated circuit (ASIC) is basically a proprietary device designed and used by a company for a specific line of products (for example Samsung cell phones or Cisco routers etc.). It is specifically an algorithm called intellectual property core implemented on a chip.

FPGA

Field programmable gate arrays (FPGA) have programmable macro cells and their interconnects are configured based on the design. They are used in embedded systems when it is required to enhance

Memory Types:

The semiconductor memory can be classified into two types

- Volatile Memory
- Non-volatile memory

Volatile memory

- RAM
 - ⇒ Static RAM
 - ⇒ Dynamic RAM

Non-volatile memory

- ROM
 - ⇒ Masked ROM
 - ⇒ OTPROM
 - ⇒ EPROM
 - ⇒ EEPROM
 - ⇒ FLASH
 - NOR FLASH
 - NAND FLASH
- NVRAM (Battery Backup RAM)

Volatile Memory:

RAM (Random Access Memory):

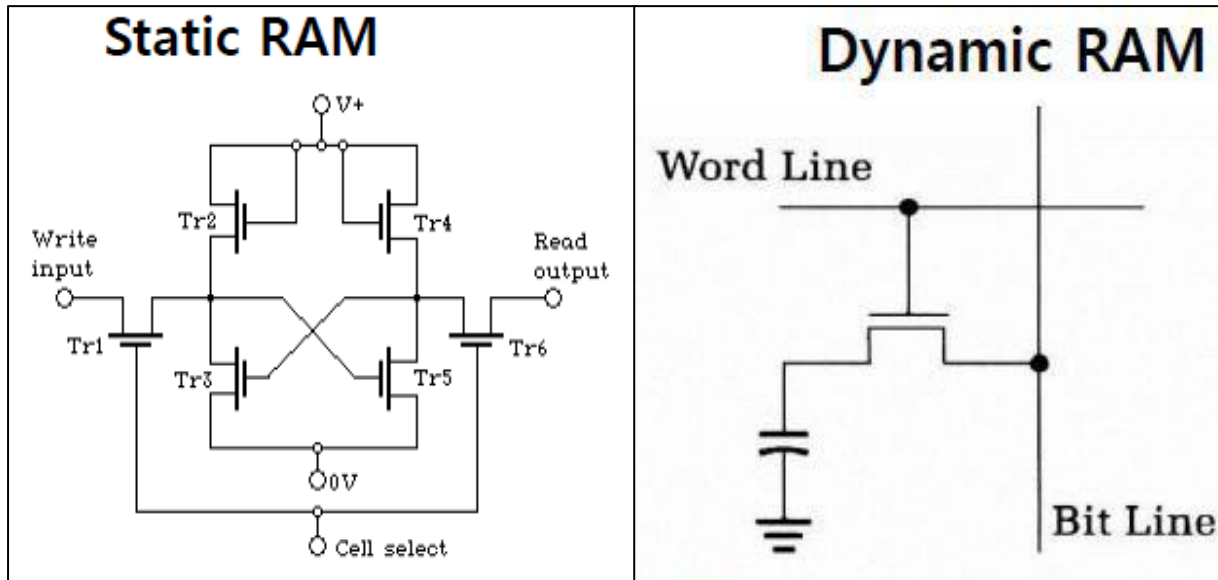
Loses its contents when power is removed, it is usually called Random-Access Memory-RAM. The vital feature is that data can be read or written with equal ease. It has ability to access any memory cell directly. RAM is much faster than ROM. It used to write and read data values while program running. Local variables, pointers, functions, recursive functions results in using large amounts of RAM. Volatile memory is used for data, and small microcontrollers often have very little RAM.

Static RAM:

Means that it retains its data even if the clock is stopped (provided that power is maintained, of course). A single cell of static RAM needs six transistors. RAM therefore takes up a large area of silicon, which makes it expensive.

Dynamic RAM:

This needs only one transistor per cell but must be refreshed regularly to maintain its contents, so it is not used in small microcontrollers. Most memory in a desktop computer is dynamic RAM.



Static RAM Vs Dynamic RAM:

Static RAM (SRAM)	Dynamic RAM (DRAM)
Made from Flip-Flops.	Made from Capacitors
High cost (per bit)	Low cost (per bit)
High using Power	Low using Power
Fast	Slow
Used in Cache Memory	Used in Main Memory
Large in Size	Low in Size
Will Retain State Forever	Automatically Discharges after sometime, Need Refreshing

Nonvolatile Memory:

ROM (Read-Only Memory):

Retains its contents when power is removed and is therefore used for the program and constant data. It is usually called Read-Only Memory-ROM. It is used as Program Memory in Microcontroller. It can't be written or modified at run time. There are many types of nonvolatile memory in use:

Masked ROM:

The data are encoded into one of the masks used for photolithography and written into the IC during manufacture. This memory really is read-only. It is used for the high-volume production of stable products, because any change to the data requires a new mask to be produced at great expense.

OTP ROM (One-Time Programmable ROM):

This is just PROM (Programmable ROM) in a normal package without a window, which means that it cannot be erased. It can be programmed one time only. Used when the firmware is stable and the product is shipping in bulk to customers. Devices with OTP ROM are still widely used and the first family of the MSP430 used this technology.

EPROM (Erasable Programmable ROM):

As its name implies, it can be programmed electrically but not erased. Devices must be exposed to ultraviolet (UV) light for about ten to twenty minutes to erase them. Erasable devices need special packages with quartz windows, which are expensive.

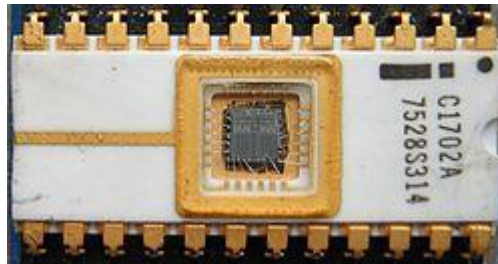


Figure: Example of EPROM IC with quartz window

EEPROM:

EEPROM (also E²PROM) stands for **Electrically Erasable Programmable ROM**. EEPROMs can be programmed and erased in-circuit i.e. without removing from hardware kit, by applying electrical signals. The contents of this memory may be changed during run time (similar to RAM), but remains permanently saved even if the power supply is off (similar to ROM). EEPROM is often used to read and store values, created during operation, which must be permanently saved.

FLASH Memory:

This can be both programmed and erased electrically and is now by far the most common type of memory. Flash Memory is designed for high speed and high density, at the expense of large erase blocks (typically 512 bytes or larger). The practical difference is that individual bytes of EEPROM can be erased but flash can be erased only in blocks. Most MSP430 devices use flash memory, shown by an **F** in the part number. Microcontrollers use NOR flash, which is slower to write but permits random access. NAND flash is used in bulk storage devices and can be accessed only serially in rows.

Many microcontrollers include both: Flash Memory for the firmware (embedded program), and a small EEPROM for parameters and history.

Overview of Design Process of Embedded Systems:

The below figure shows the Design Process of Embedded Systems, In the first stage, according to user requirements the designer chooses the electronic chip which is suitable. In the next stage the designer will work on S/W side and H/W side separately. After that they will integrate the both S/W and H/W to design the Embedded System.

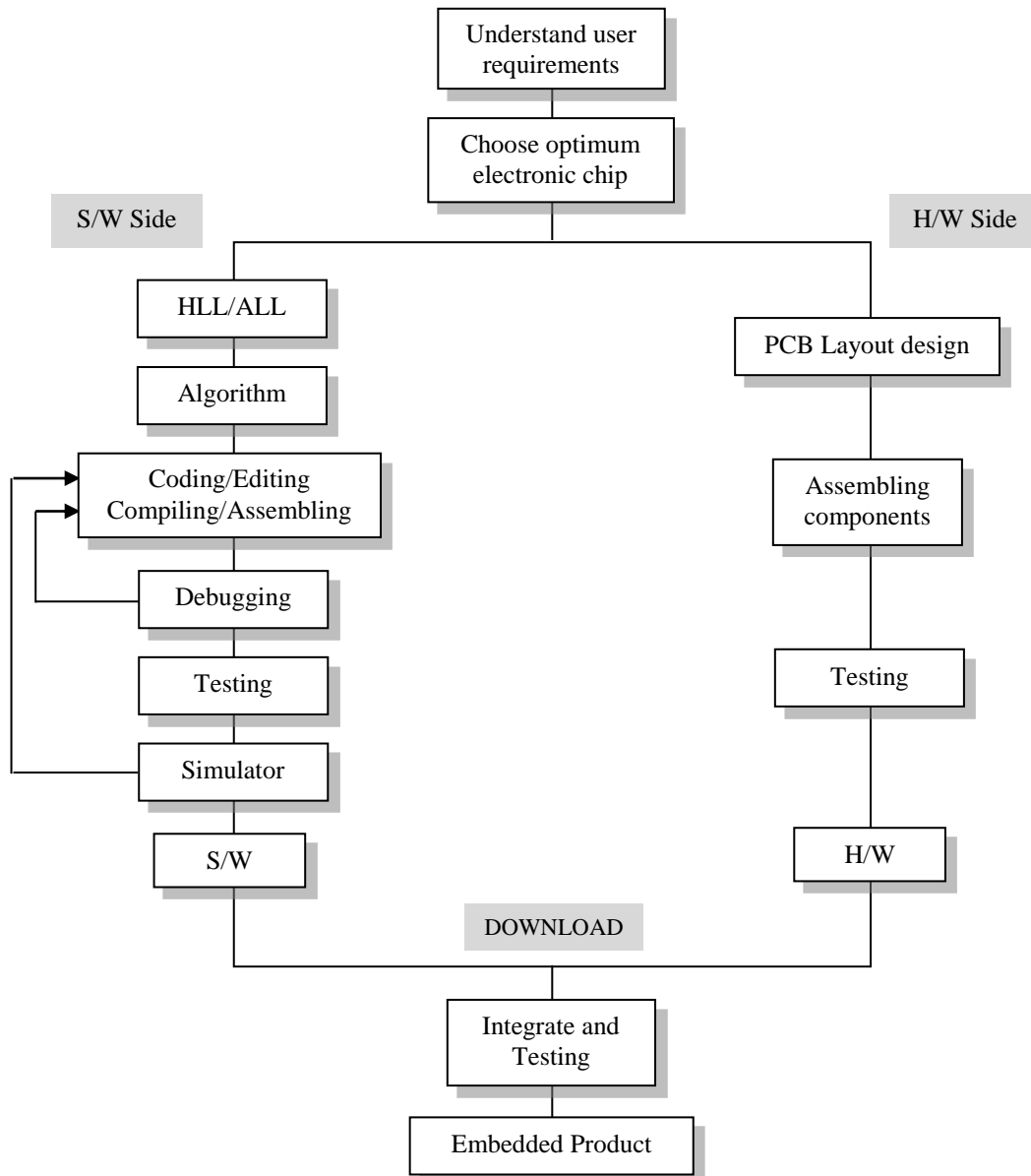


Figure: Design Process of Embedded Systems

An example of an embedded system:

Here is a simple application to introduce a small embedded system – a stepper motor controller for a robotics system. The stepper motor mentioned here is an electromechanical device that rotates in discrete step angles when electrical pulses are applied to it. Suppose in an industrial environment, a robot arm is employed to pick-up components from one container and deposit to another container. The robot arm is operated by three stepper motors,

one to move the arm from one container to the other and other two to make a grip to tightly hold the component. To control these stepper motors, a small embedded system can be designed as shown in below figure. The hardware components are a microcontroller, three stepper motors and a robot arm. To make this system functional, three program modules are required.

- ⇒ Module1 for the stepper motor1 to rotate counter clockwise with definite angle so that robot arm can move from container1 to container 2 also to rotate clockwise to move back to container1 when the component is picked.
- ⇒ Module 2 for the stepper motors 2 & 3 to make the motor 2 to rotate in counter clockwise and motor 3 to rotate in clockwise both simultaneously so that the robot arm can pick and make a grip on the component.

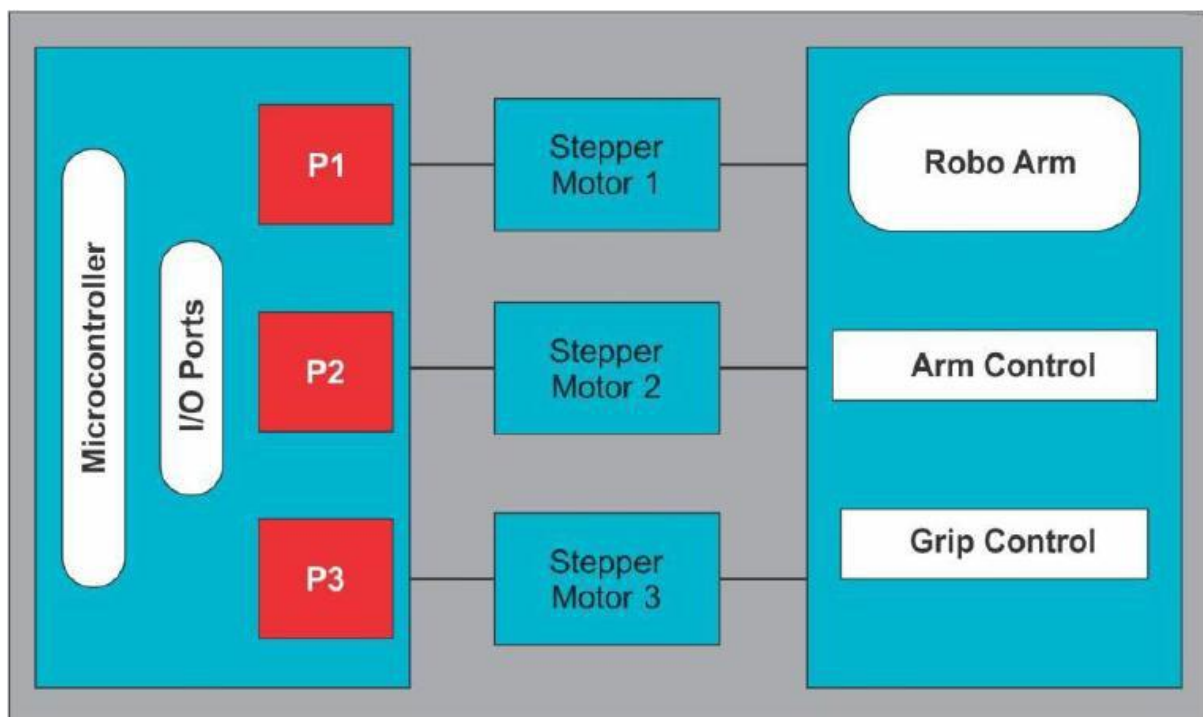


Figure: Stepper Motor Control Embedded System

Programming Languages and Tools for Embedded Design:

Programming Languages:

- ⇒ Code is typically written in C or C++, but various high-level programming languages, such as Python and JavaScript, are now also in common use to target microcontrollers and embedded systems. Ada is used in some military and aviation projects.

Machine code: The binary data that the processor itself understands. Each instruction has a binary value called an opcode. It is unrecognizable to humans, unless you spent a very long time on low-level debugging. Some very early computers had to be programmed in machine code, but that was long ago, thank goodness. You will see it, however, because the contents of memory are shown in the debugger and machine code is included in the “disassembly”.

Assembly language: Little more than machine code translated into English. The instructions are written as words called mnemonics rather than binary values and a program called an assembler translates the mnemonics into machine code. It does a little more than direct translation, but not a lot, nothing like a compiler for a high-level language.

A major disadvantage of assembly language is that it is intimately tied to a processor and is therefore different for each architecture. Even worse, the detailed usage varies between development environments for the same processor. Most programming of small microcontrollers was done in assembly language until recently, despite these problems, mainly because compilers for C produced less-efficient code. Now the compilers are better and modern processors are designed with compilers in mind, so assembly language has been pushed to the fringes. A few operations, notably bitwise rotations, cannot be written directly in C, and for these assembly language may be much more efficient. However, the main argument for learning assembly language is for debugging. There is no escape if you need to check the operation of the processor, one instruction at a time. Disassembly is the opposite process to assembly, the translation of machine code to assembly language.

C: The most common choice for small microcontrollers nowadays. A compiler translates C into machine code that the CPU can process. This brings all the power of a high-level language—data structures, functions, type checking and so on—but C can usually be compiled into efficient code. Compilation used to go through assembly language but this is now less common and the compiler produces machine code directly. A disassembler must then be used if you wish to review the assembly language.

C++: An object-oriented language that is widely used for larger devices. A restricted set can be used for small microcontrollers but some features of C++ are notorious for producing highly inefficient code. Embedded C++ is a subset of the language intended for embedded systems. Java is another object-oriented language, but it is interpreted rather than compiled and needs a much more powerful processor.

BASIC: Available for a few processors, of which the Parallax Stamp is a well-known example. The usual BASIC language is extended with special instructions to drive the peripherals. This enables programs to be developed very rapidly, without detailed understanding of the peripherals. Disadvantages are that the code often runs very slowly and the hardware is expensive if it includes an interpreter.

Programming Tools:

The microcontrollers can be programmed using various Programming/Development tools like Embedded Software from Mentor Graphics, IAR Systems, Keil MicroVision from ARM Ltd., and Code Composer Studio (CCS) from Texas Instruments.

Introduction to Code Composer Studio:

At this juncture of the book, it is important to concentrate on a single development tool for programming the hardware. In this book, Code Composer Studio is used because it is

free and is supported by a large TI community commonly known as E2E. This forum is a congregation of engineers and hobbyist working on TI products across the world.

Code Composer Studio (CCS):

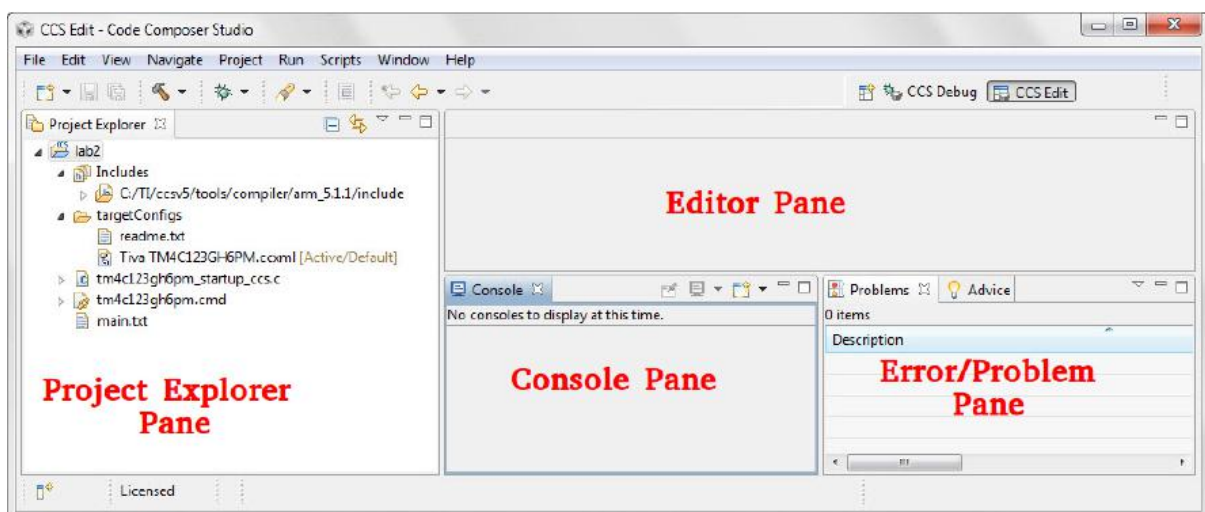
CCS is a TI proprietary cross platform IDE used to convert C and assembly language code to executable for TI processors, involving Digital Signal Processors, ARM, and other microcontrollers. CCS combines editing, debugging and analysis tools into a single IDE based on Eclipse open source development tool.

Compilation (Build Process):

CCS uses a Code Generation Tool (often called CGT) for compilation process also called as build process. It is used to generate executables for a target device. This process takes source codes written in C, C++ and/or Assembly Language and produces executable at the output called binaries. This process undergoes many intermediated stages. A normal development flow of a compilation process takes the source files written in C/C++ and compiles them to create assembly language files using a compiler. For projects where source code is written in assembly language, this process is bypassed. In many projects, some of the logic functions are written in assembly languages along with other C/C++ source codes. After the assembly language files are generated by the compiler, an assembler converts them to relocatable object files. These object files are linked to the runtime libraries included in the project by a linker. The linker produces executables from these files using protocols mentioned in the command file. A command file typically consists of a list of all memories of the part and the type of software compatible to them.

CCS uses C/C++ compiler and other compilation tools which are developed by engineers at TI over the years and packaged into TI's code generation tools. They are commonly amalgamation (combination) of various tools used in compilation processes, namely, Compiler, Assembler, Linker, Optimizer, Code Generator, Parser, Linear Assembler, Archiver, Disassembler and many more.

Review the CCS Editing GUI – Figure below shows various panes of CCS which appear after project is created. Understanding their nomenclature and functionality with help in process of application development



Debugger:

The CCS debugger depends on a configuration file and a general extension language (GEL) file. The debugger initializes and loads the software on a target device using information provided by these files. A target configuration file specifies

- (i) Connection type to the target device,
- (ii) Target device, and
- (iii) About a startup script

Table: Programming/Development Tools for Tiva C Series

Product	License	Compiler	IDE	Debugger	JTAG
Embedded Software, Mentor Graphics	30-day full function	GNU C/C++	Gdb	Eclipse	
KEIL MicroVision, ARM Ltd.	Full function. Onboard emulation limited	Real View C/C++	µVision	µVision	U-Link, 199 USD
CCS, Texas Instruments	32KB code size limited. Upgradeable	TI C/C++	Eclipse	CCS	XDS100 79 USD
IAR Systems	32KB code size limited. Upgradeable	IAR C/C++	Embedded Workbench	C-SPY	J-Link, 299 USD

In CCS, startup scripts are specified to setup the memory map for debugger. It is also used to setup any initial target state that is necessary for connection to the debugger using memory or register writes. These scripts are known as GEL script file. „OnStartup()“ function in the GEL file runs when the debugger is launched. After the target is connected, „OnTargetConnect()“ defined in the GEL file is executed.

Organization and Building a CCS Project:

In CCS, designs are organized in workspaces and projects which are merely folders in the file system. When CCS is launched, it prompts the users to provide a folder path to the workspace. This folder consists of individual project folders and a folder named as ‘.metadata’. The .metadata folder consists of CCS settings and preferences for the particular workspace. Along with source files, header files and library files, each project folder contains the build and the tool settings for the project. It also contains the target configuration file and the command file required by the debugger. CCS has two predefined build configurations, namely, debug and release. It also provides custom build configurations.

Debug – It is generally used when it is required to operate in debugging mode. It includes the symbol tables and executes compilation without any optimization.

Release – Building project in this mode is suited when the user requires performance. It discards all symbol tables and implements the full code optimization. It is therefore a noted convention to use this mode only when the final version of a project is to be deployed on the hardware. For all other intermediate versions, debug mode is rather preferred.

Custom Configuration – CCS also provide its users to add custom build configurations for a particular project. It can be done by going to processor options under ‘properties -> build -> ARM Compilers’.

UNIT-II Embedded Processor Architecture

CISC Vs RISC design philosophy, Von-Neumann Vs Harvard architecture. Introduction to ARM architecture and Cortex – M series, Introduction to the TM4C family viz. TM4C123x & TM4C129x and its targeted applications. TM4C block diagram, address space, on-chip peripherals (analog and digital) Register sets, addressing modes and instruction set basics

Microprocessor Architecture Classification

The microprocessor architecture can be classified using different aspects. Figure1 shows two different possible classifications of the processor architecture.

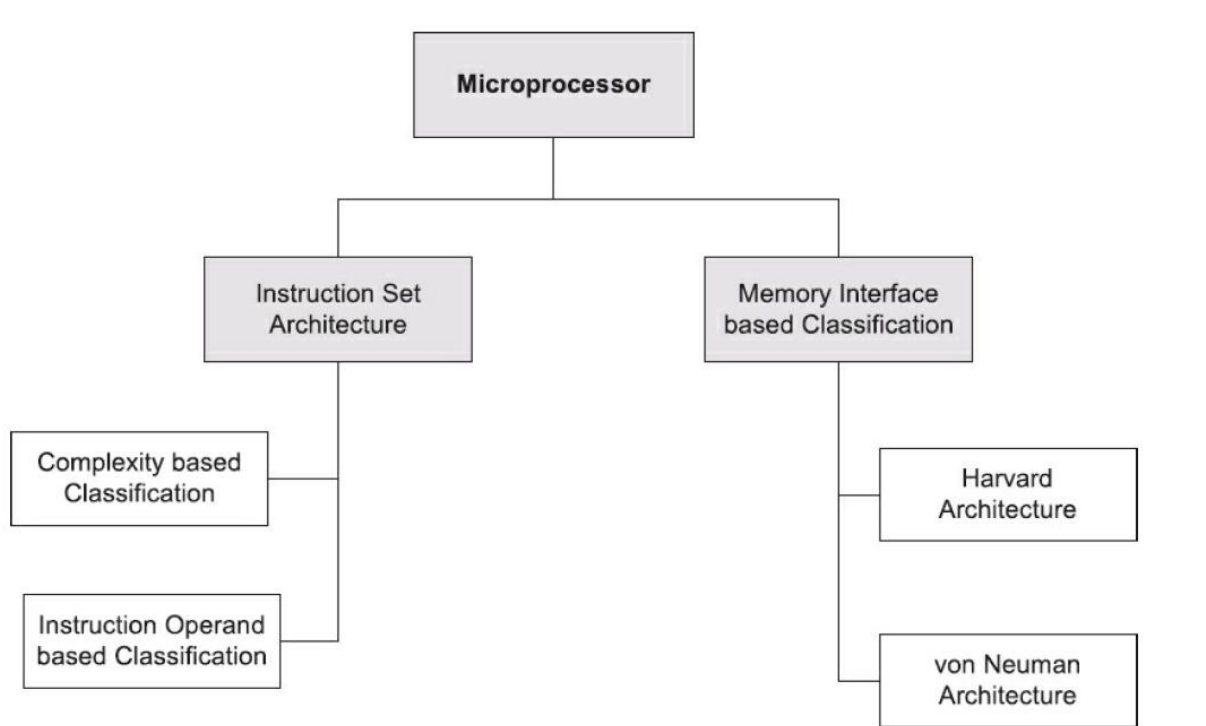


Fig 1: Classifications of the processor architecture

Instruction Set Architecture

Now let us discuss the processor architecture classification based on instruction set architecture (ISA). There are two important aspects that can be used to define ISA classification. One classification can be done based on the complexity of instructions, while the other possibility is based on the instruction operands. Next, we discuss each of these ISA classifications.

Complexity-Based ISA Classification

Using the ISA complexity as the classification measure, we can categorize the microprocessors into two groups: complex instruction set computer (CISC) and reduced instruction set computer (RISC). In reality, there is a spectrum of architectures that we can classify as CISC or RISC. Figure 2 depicts the major differences between the two architectures. We make the following general observations when deciding whether to call a computer CISC or RISC.

Complex instruction set computers (CISC):

- One possible use of complex instruction set architecture was in early computers where the processors were much faster than available memories.
- Fetching an instruction from memory used to become the performance bottleneck.
- One of the advantages of CISC architecture is that a single complex instruction can perform many operations. For instance, find the zeros of a polynomial.
- However, complex instructions require many processor clock cycles to complete and most of the instructions can access memory.
- A program running on a CISC architecture-based machine involves a relatively small number of complex instructions, which can provide high code density.
- In addition, many instruction types with varying instruction length are available supporting different addressing modes while requiring fewer and specialized registers.

- In CISC the complexity is embedded in the processor hardware, making the compilation tools design simpler. Some of the example processors based on CISC architecture are Intel (x86) and Freescale 9S12.

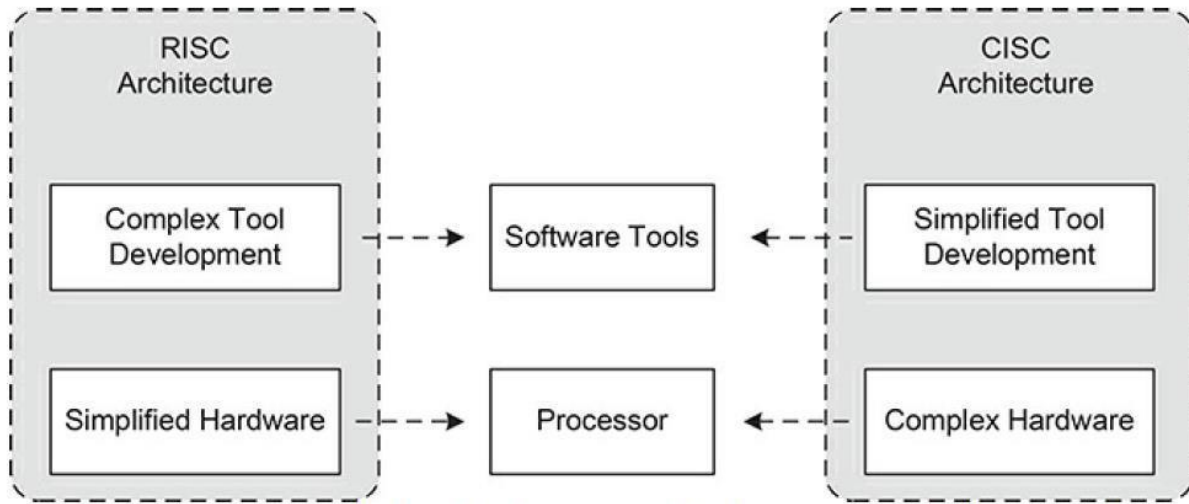


Fig 2: depicts the major differences between the two architectures

Reduced Instruction Set Computers (RISC):

- This architecture is suited for those scenarios where the processor speeds match that of memories. This speed matching reduces the penalty for instruction or parameter fetching from the memory.
- Another associated advantage with RISC is the simplified instructions used by this architecture. Each complex operation is broken into multiple simplified operations and dedicated instructions are provided for these operations. For instance, load and store instructions are provided for memory read and write operations. The other instructions cannot access memory directly.
- The simplified instructions make it possible to execute an instruction in a single processor clock cycle. However, this is not true in general and some of the instructions in RISC may require more than one clock cycle for its execution completion.
- A task when run on a RISC computer requires a relatively larger number of simplified instructions and results in low code density. An associated advantage is fewer memory addressing modes resulting in reduced complexity.

- The simplicity in the hardware architecture in RISC is complemented by the increased complexity in the generation of assembly code by the tools (compiler) or by the programmer.
- Some of the processor architectures based on RISC are MIPS, ARM, SPARC, and PowerPC.

Memory Interface-Based Architecture Classification

There are two widely used memory interface architectures, namely, von Neumann architecture and Harvard architecture. Both of them are shown in Figure 3.

- The von Neumann architecture uses a common bus for both data as well as code memory. As a result either an instruction can be fetched from memory or data can be read/written to/from memory during each memory access cycle. Instructions and data are stored in the same memory subsystem and share a common bus to the processor.
- The Harvard architecture, on the other hand, utilizes separate buses for accessing code and data memories. Using separate buses for code and data memories allows instructions and data to be accessed simultaneously.
- In addition, the next instruction may be fetched from memory at the time when the previous instruction is about to finish its execution, allowing for a primitive form of pipelining.
- Pipelining decreases the per instruction execution time; however, main memory access time is a major bottleneck in the overall performance of the system.
- Mostly, microprocessors are implemented using von Neumann architecture, while microcontrollers use Harvard architecture. ARM based microcontrollers use Harvard architecture.

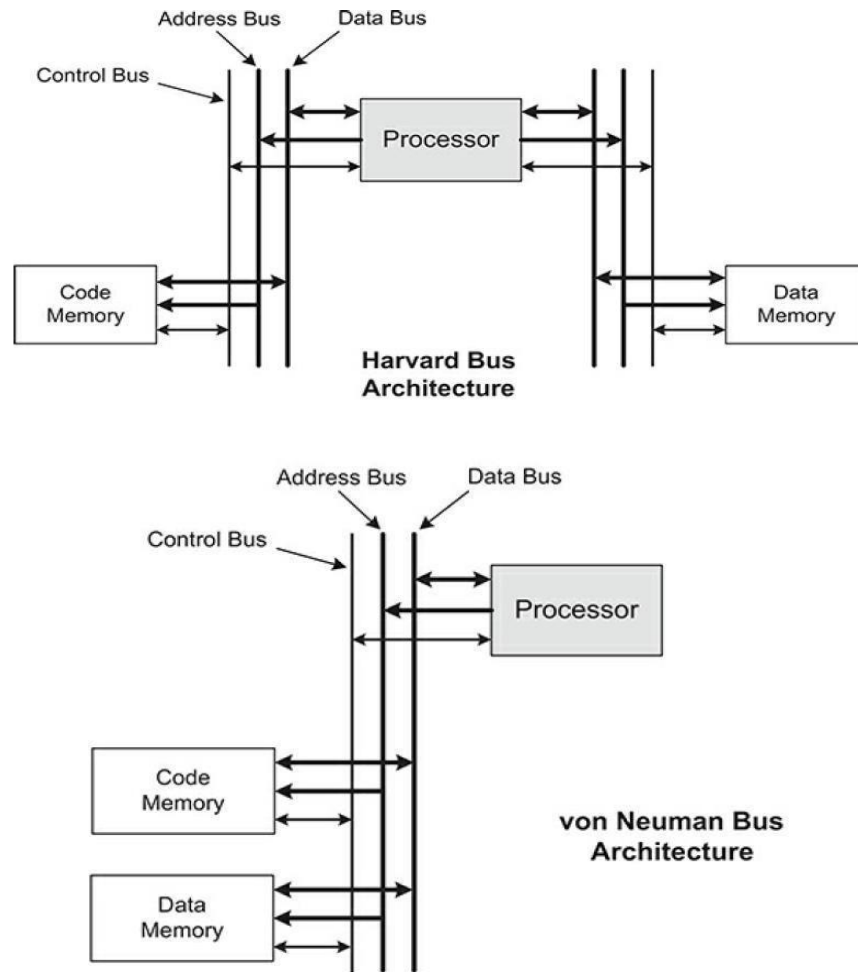


Fig 3: Neumann architecture and Harvard architecture.

ARM Microcontroller Architectures

OVERVIEW AND INTRODUCTION

Embedded system is generally composed of a group of programmable devices with some memory devices and a set of peripheral device I/O ports. In fact, an embedded system can be considered as an integrated system by embedding some Central Processing Units (CPUs) with some Memory subsystems, I/O Ports, and maybe several Peripheral Devices together into a single semiconductor chip to get an intelligent control unit, called a Microcontroller Unit (MCU). Therefore, a typical embedded system can be thought of as an MCU.

Most important components involved in this MCU include:

- ARM CPU or Processor
- Memory (SRAM and Flash Memory)

- Parallel Input and Output (PIO) Ports

- Some internal devices (Timer/Counters) or peripheral devices (ADC and USB) Different embedded systems or MCUs have been developed and built by different vendors in recent years. One of the most popular MCUs is the ARM Cortex-M MCU family.
- This kind of MCU provides multifunction's and control abilities, low power consumptions, high-efficiency signal processing functionality, low cost, and easy-to-use advantages. The latest product of the ARM Cortex-M family is Cortex-M4 MCU.

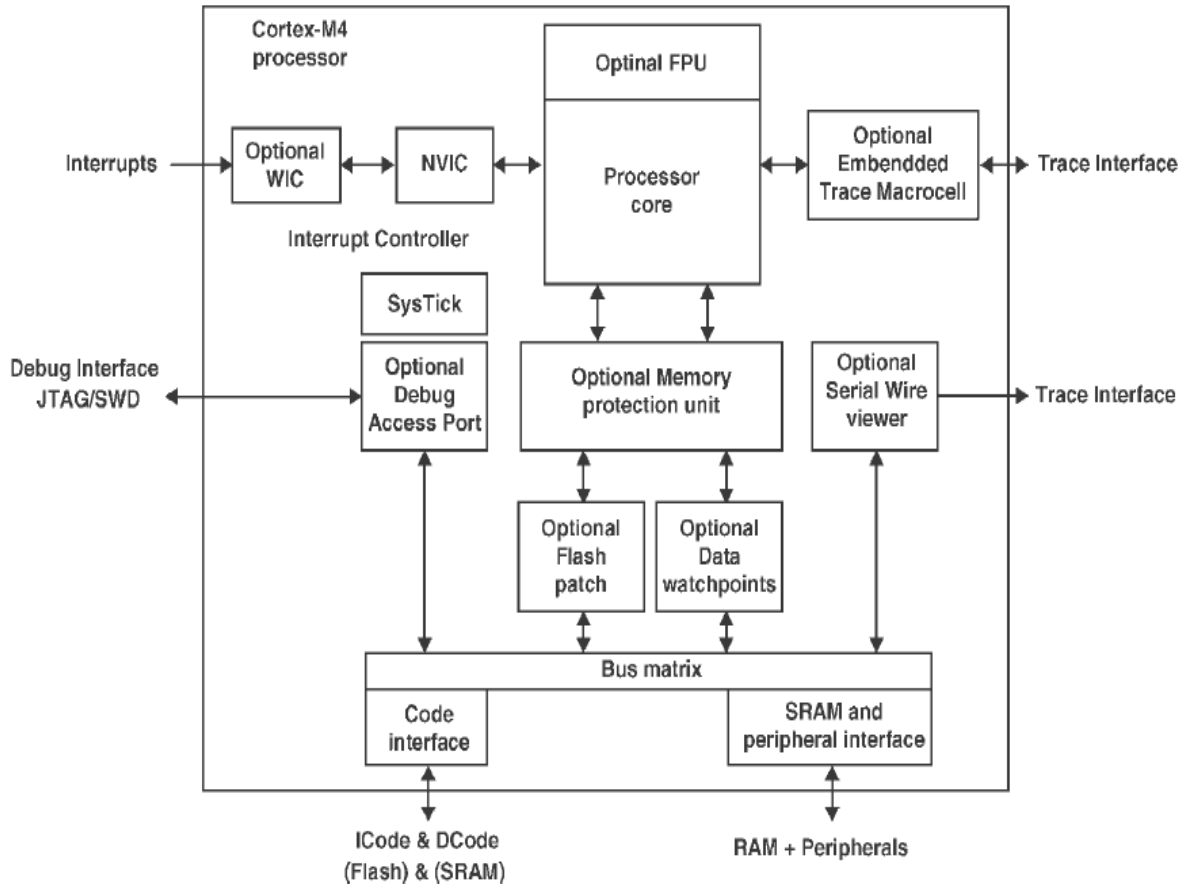
INTRODUCTION TO ARM CORTEX -M4 MCU

- The ARM Cortex-M is a group of 32-bit Reduced Instruction Set Computing (RISC) ARM processor
- The ARM Cortex-M4 processor is the latest embedded processor by ARM specifically developed to address digital signal control markets that demand an efficient, easy-to-use blend of control and signal processing capabilities.
- The combination of high efficiency signal processing functionality with the low-power, low cost, and ease-of-use benefits of the Cortex-M family of processors is designed to satisfy the emerging category of flexible solutions specifically targeting the motor control, automotive, power management, embedded audio, and industrial automation markets.

The ARM Cortex-M4 MCU provides the following specific functions:

- Although the Cortex-M4 processor is a 32-bit MCU, it can also handle 8-bit, 16-bit, and 32-bit data efficiently.
- The Cortex-M4 MCU itself does not include any memory, but it provides different memory interfaces to the external Flash Memory and SRAMs.
- Because of its 32-bit data length, the maximum searchable memory space is up to 4 GB.

- In order to effectively manage and access this huge memory space, different regions are created to store system instructions and data, users' instructions, data, and mapped peripheral device registers and related interfaces.
- The internal bus system used in Cortex-M4 MCU is 32-bit and is based on the so-called Advanced Microcontroller Bus Architecture (AMBA) standard. The AMBA standard provides efficient operations and low power cost on the hardware.
- The main bus interface between the MCU and external components is the Advanced High-performance Bus (AHB), which provides interfaces for memory and system bus as well as for peripheral devices.
- A Nested Vectored Interrupt Controller (NVIC) is used to provide all supports and managements to the interrupt responses and processing to all components in the system.
- The Cortex-M4 MCU also provides standard and extensive debug features and supports to enable users to easily check and trace their program with breakpoints and steps.



The functional block diagram for the ARM® Cortex®-M4 MCU.

Fig 4: Functional Block Diagram of ARM Cortex-M4

The Cortex-M4 Core or Center Processing Unit (CPU)

This is a center control unit for the entire Cortex-M4 MCU. All instructions are fetched into, decoded, and executed inside this CPU. **This CPU or processor consists of three key components, namely, Register Bank, Internal Data Path, and control unit.** This is the core for the normal operations of the entire MCU.

Floating Point Unit (FPU)

One of the important differences between the Cortex-M4 MCU and Cortex-M3 MCU is that an optional Floating-Point Unit (FPU) is added into the Cortex-M4 Core to enhance the floating-point data operations. The Cortex-M4 FPU It provides floating-point computation functionality that is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic.*

A System Timer SysTick

The ARM Cortex-M4 MCU provides an integrated system timer, SysTick (System Tick), which includes a **24-bit counter with clear-on-writing, decrementing, and reloading-on-zero control mechanism**. The main purpose of using this timer is to provide a periodic interrupt to ensure that the OS kernel can invoke regularly.

If you do not need to use any embedded OS in your application, the SysTick timer can work as a simple timer peripheral for periodic interrupt generator, delay generator, or timing measurement device. The reason for installing this timer inside the MCU is to make software portable, and any program built with a Cortex-M processor can run any OS written for Cortex-M4 MCU.

Nested Vectored Interrupt Controller (NVIC)

The Nested Vectored Interrupt Controller (NVIC) is closely integrated with the processor core to achieve low latency interrupt processing. These features include the following:

- Monitor and pre-process any exception or interrupt occurred during the normal running of the processor. All exceptions and interrupts are categorized into different emergency levels based on their priority levels, from 1 to 15 for exceptions and from 16 to 240 for interrupts.
- Identify the exception or interrupt source if an exception/interrupt occurred, and direct the main control to the entry point (address) of the related exception/interrupt Interrupt Service Routine (ISR) to process the exception/interrupt.
- Dynamically manage all exceptions and interrupts based on their priority levels.
- Automatically store the processor states on interrupt entry, and restore it on interrupt exit, with no instruction overhead.
- Optional *Wake-up Interrupt Controller* (WIC) supports the ultra-low-power sleep mode.

Bus Matrix and Bus Interfaces

The Bus Matrix and Bus Interfaces provide:

- Three Advanced High-Performance Bus-Lite (AHB-Lite) interfaces:
 - ICode interface to flash ROM,

- DCode interface to SRAM and peripheral interfaces, as well as System bus interfaces, including the internal control bus and debug components.
 - Private Peripheral Bus (PPB) based on Advanced Peripheral Bus (APB) interface.
- Bit-Band support that includes atomic bit-band writing and reading operations.
 - Memory access alignment and Write buffer for buffering of write data.
 - Exclusive access transfers for multiprocessor systems.

Memory Protection Unit (MPU)

This is an optional MPU used for memory protection purpose and it includes:

- Eight memory regions.
- Sub-Region Disable (SRD), enabling efficient use of memory regions.
- The ability to enable a background region that implements the default memory map attributes.

System Control Block (SCB)

This block is located in the System Control Space (SCS) in the memory map, and it is integrated with the NVIC unit together to provide the following features:

- Monitor and control the processor configurations, such as low power modes.
- Provide fault detection information via fault status register.
- Relocate the Vector Table in the memory map by adjusting the content of the Vector Table Offset Register (VTOR).

Debug Access Port (DAP)

The DAP is an optional unit and it provides the following features:

- Debug access to all memory and registers in the system, including access to memory mapped devices, access to internal core registers when the core is halted, and access to debug control registers even while SYSRESETn is asserted.
- Serial Wire Debug Port (SW-DP)/Serial Wire JTAG Debug Port (SWJ-DP) debug access.
- Optional Flash Patch and Breakpoint (FPB) unit for implementing breakpoints and code patches.

- Optional Data Watchpoint and Trace (DWT) unit for implementing watchpoints, data tracing, and system profiling.
- Optional Instrumentation Trace Macro cell (ITM) for support of printf() style debugging.
- Optional Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer (TPA), including Single Wire Output (SWO) mode.
- Optional Embedded Trace Macro cell (ETM) for instruction trace.

Introduction to the TM4C family viz. TM4C123x & TM4C129x and its targeted applications:

TIVA TM4C123GH6PM Microcontroller:



Fig 5: TM4C123GH6PM Microcontroller Block Diagram

The TM4C microcontroller block diagram shown in above figure has six functional units. The cortex M4F core, on-chip memory, analog block, serial interface, motion control and system integration.

Features:

- TM4C123GH6PM microcontroller has 32-bit ARM Cortex M4 CPU core with 80 MHz clock rate.
- Memory protection unit (MPU) provides protected operating system functionality and floating-point unit (FPU) supports IEEE 754 single precision operations.
- JTAG/SWD/ETM for serial wire debugs and traces (SWD/T).
- Nested vector interrupt controller (NVIC) reduces interrupt response latency.
- System control block (SCB) holds the system configuration information.
- The microcontroller has a set of memory integrated in it: 256 KB flash memory, 32 KB SRAM, 2 KB EEPROM and ROM loaded with TIVA software library and boot loader.
- Serial communications peripherals such as: 2 CAN controllers, full speed USB controller, 8 UARTs, 4 I2C modules and 4 Synchronous Serial Interface (SSI) modules.
- On chip voltage regulator, two analog comparators and two 12 channel 12-bit analog to digital converter with sample rate 1 million samples per second (1MSPS) are the analog functions in built to the device.
- Two quadrature encoder interface (QEI) with index module and two PWM modules are the advanced motion control functions integrated into the device that facilitate wheel and motor controls.
- Various system functions integrated into the device are: Direct Memory Access controller, clock and reset circuitry with 16 MHz precision oscillator, six 32-bit timers, six 64-bit timers, twelve 32/64-bit captures compare PWM (CCP), battery backed hibernation module and RTC hibernation module, 2 watchdog timers and 43 GPIOs.

Few Applications:

- Building automation system
- Lighting control system
- Data acquisition system
- Motion control
- IoT and Sensor networks.

TIVA TM4C129CNCZAD Microcontroller:

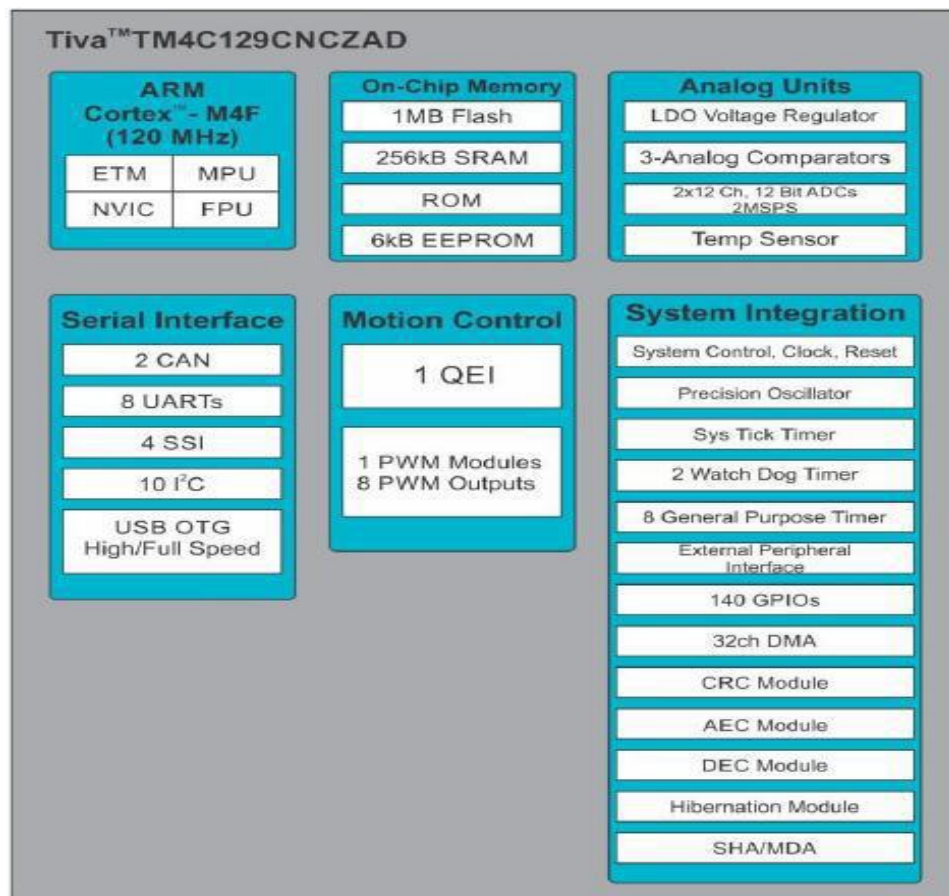


Fig 6: TIVA TM4C129CNCZAD Microcontroller Block Diagram

Features:

- TM4C129CNCZAD microcontroller has 32-bit ARM Cortex M4F CPU core with 120 MHz clock rate.
- Memory protection unit (MPU) provides a privileged mode for protected operating system functionality and floating-point unit (FPU) supports IEEE 754 compliant single precision operations.
- JTAG/SWD/ETM for serial wire debug and trace.

- Nested vector interrupt controller (NVIC) reduces interrupt response latency and high-performance interrupt handling for time critical applications.
- The microcontroller has a set of memory integrated in it: 1MB flash memory, 256 KB SRAM, 6 KB EEPROM and ROM loaded with TIVA ware software library and boot loader.
- Serial communications peripherals such as: 2 CAN controllers, full speed and high-speed USB controller, 8 UARTs, 10 I2C modules and 4 Synchronous Serial Interface (SSI) modules.
- On chip voltage regulator, three analog comparators and two 12 channel 12-bit analog to digital converter with sample rate 2 million samples per second (2MSPS) and temperature sensor are the analog functions in built to the device.
- One quadrature encoder interface (QEI) and one PWM module with 8 PWM outputs are the advanced motion control functions integrated into the device that facilitate wheel and motor controls.
- Various system functions integrated into the device are: Micro Direct Memory Access controller, clock and reset circuitry with 16 MHz precision oscillator, eight 32-bit timers, and low power battery backed hibernation module and RTC hibernation module, 2 watchdog timers and 140 GPIOs.
- Cyclic Redundancy Check (CRC) computation module is used for message transfer and safety system checks. CRC module can be used in combination with AES and DES modules.
- Advanced Encryption Standard (AES) and Data Encryption Standard (DES) accelerator module provides hardware accelerated data encryption and decryption functions.
- Secure Hash Algorithm/ Message Digest Algorithm (SHA/MDA) provides hardware accelerated hash functions for secured data applications.

Applications:

- Low power, hand-held smart devices
- Gaming equipment
- Home and commercial site monitoring and control

- Motion control
- Medical instrumentation
- Test and measurement equipment
- Factory automation
- Fire and security
- Smart Energy/Smart Grid solutions
- Intelligent lighting control
- Transportation

Cortex-M-Based TM4C123 Microcontroller

The Cortex-M microcontroller selected for hardware-based programming and interfacing illustrations is TM4C123 from Texas Instruments. This microcontroller belongs to the high-performance ARM Cortex-M4F based architecture and has a broad set of peripherals integrated. The key features of the TM4C123 microcontroller are listed below.

Clock frequency: Processor clock frequency up to 80 MHz with floating point unit (FPU).

System timer SysTick: SysTick is 24-bit, clear-on-write, decrementing timer. Its flexible control allows its use for the purpose of system time base generation.

Nested vectored interrupt controller: The microcontroller TM4C123 also includes a nested vectored interrupt controller (NVIC). The NVIC along with Cortex-M processor can prioritize and handle the interrupts in handler mode. On the occurrence of an interrupt, the state of the processor is automatically stored onto the system stack and is restored from the stack when exiting from the interrupt service routine. The fetching of the interrupt vector and saving of the state (i.e., storing the registers onto stack) are performed simultaneously, which provides

efficient interrupt entry and as a result the interrupt latency is reduced. The Cortex-M4F processor in TM4C123 also supports the tail-chaining functionality that further reduces interrupt latency. A collection of 7 system exceptions and 65 peripheral interrupts are supported by TM4C123. The microcontroller supports 8 priority levels, which can be configured for these exceptions and interrupts.

Debugging interface using JTAG/SWD: The TM4C123 microcontroller provides a JTAG and SWD (serial wire debug) based debugging interface for programming and debugging purpose.

TM4C123 Microcontroller Block Diagram

A high-level block diagram of the TM4C123 microcontroller is shown in Figure 7.

Note that there are two on-chip buses, AHB and APB, which connect the processor core to the peripherals. These buses are constructed from the system bus using a bus matrix. It should be recalled that this bus matrix is different from the one used inside the processor block. The bus matrix inside the processor block is responsible for connecting the instruction and data buses from the processor core to I-Code, D-Code, and System buses.

The advanced peripheral bus (APB) is the low speed bus. The more complex advanced high-performance bus (AHB) gives improved performance than the APB bus and should be used for interfacing those peripherals, which require faster data transfer speeds. The block diagram also shows different peripheral modules connected to these two buses. It should be noted that only the GPIO and direct memory access (DMA) modules have connectivity available to both the buses.

TM4C123 Microcontroller Peripherals

The TM4C123 microcontroller is equipped with a variety of peripherals, which are integrated with the ARM Cortex-M core. The TM4C123 on chip peripherals can be categorized into the following four groups.

System integration peripherals: Different system integration peripherals include GPIOs, system clock management, direct memory access as well as hibernation module. The GPIOs can be used for parallel interfacing. For example, external memory can be interfaced using

parallel interface. In addition, the GPIO pins on TM4C123 can also be configured as interrupt inputs for external binary events. The system clock management module is responsible for generating clock for system as well as for other peripherals. It allows the user to run the system at different operating frequencies. In many battery-operated systems, the power conservation is of extreme importance. For that purpose, TM4C123G microcontroller has an integrated battery-backed hibernation module to efficiently keep the microcontroller in power down mode for reducing the power usage during intervals of inactivity.

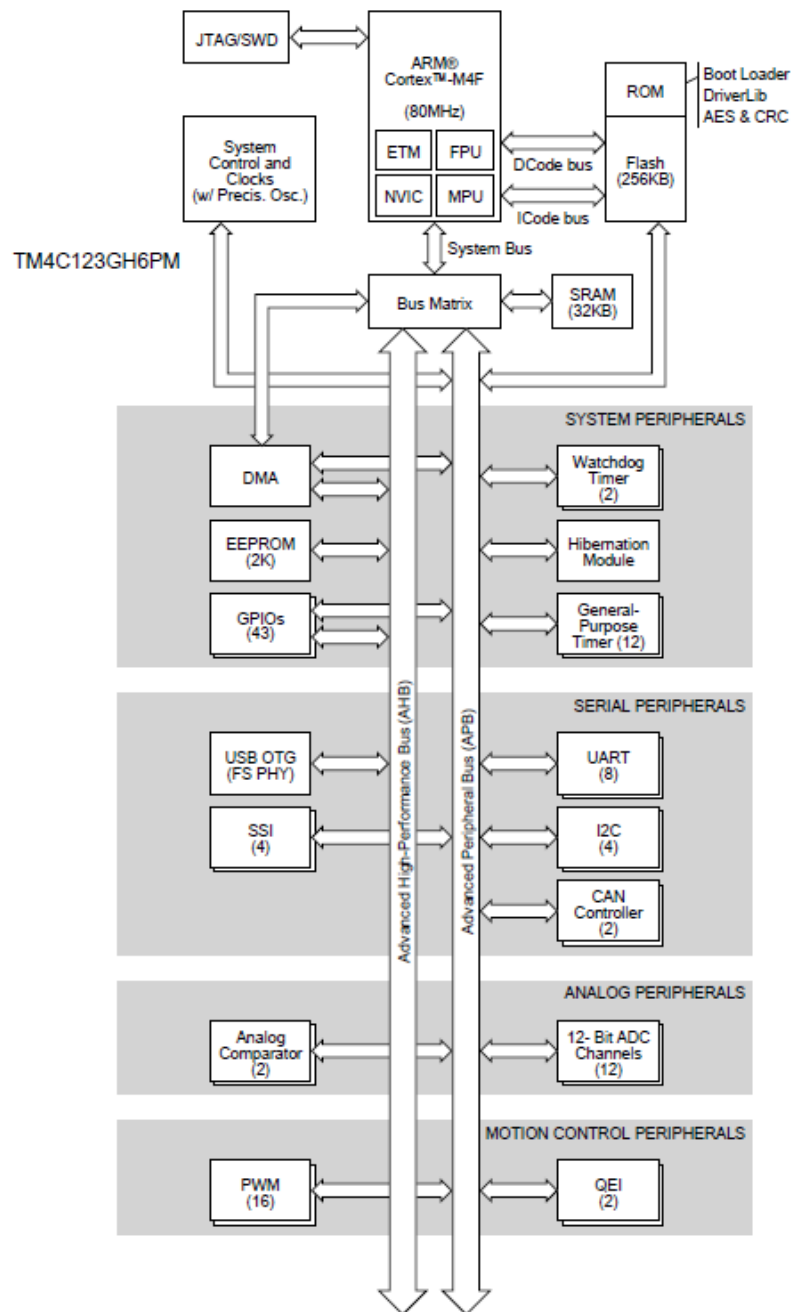


Fig 8: TIVA TM4C Block Diagram

Timing interfaces: The TM4C123 has 12 general purpose timers, two watchdog timers, and one systick timer. Systick timer is integrated as part of the ARM processor core, while all other timers are integrated as peripheral modules. There are six 32-bit general purpose timers and each of these timers can be split into two 16-bit timers. Similarly, there are six 64-bit timers and each of these timers is made by concatenating two 32-bit timers. In addition to counter mode, a timer can be configured in input capture or output compare modes. Input capture and output compare will be used to create periodic interrupts and measure period, pulse width, phase as well as frequency. In addition, there are two 32-bit watchdog timers that can be used for regaining system control.

Communication interfaces: The communication interfacing peripherals primarily consist of serial interfaces. The TM4C123 has eight Universal Asynchronous Receiver/Transmitter (UART) interfaces, which can be used for asynchronous point to point serial communication between two devices. The UART interface allows simultaneous communication in both directions making its communication full-duplex. The TM4C123 has four Synchronous Serial Interfaces (SSI), which is also called Serial Peripheral Interface (SPI). The SPI interface is also full-duplex. The microcontroller also supports four I2C interfaces, which is a serial bus interface that can be used to connect multiple devices. The I2C bus is half-duplex. There are two CAN (controller area network) interfaces and one USB device interface available as well on the TM4C123 microcontroller. The CAN bus is commonly used in automotive as well as distributed control systems.

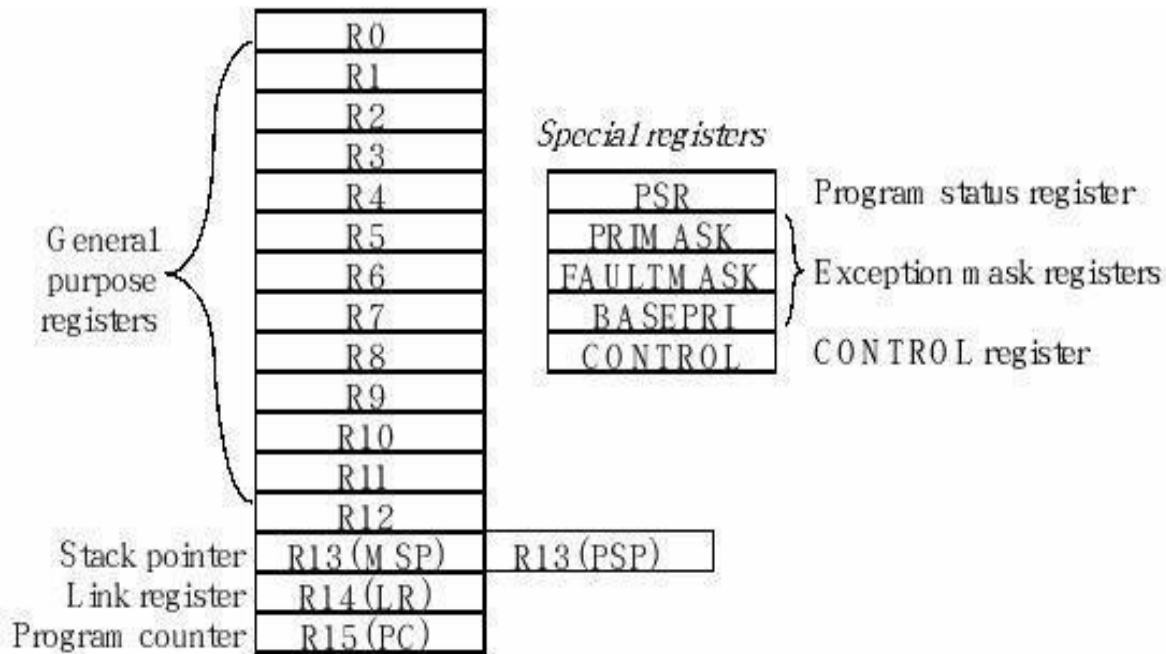
Analog interfacing peripherals: There are two 12-bit analog-to-digital converters (ADCs) available in TM4C123. The ADC is used for converting the analog signals to their digital equivalent, which is required for many data acquisition applications. In addition to ADCs, there are analog comparators available on the microcontroller. These analog comparators can be used for converting an analog signal to binary logical signal based on the thresholding principle. In addition, it can also be used for comparing two analog input signals. The TM4C123 peripherals discussed above are summarized in Table 1.

Peripheral	Description
GPIOs	43 configurable pins for this purpose.
System Clock	80 MHz maximum, can be configured arbitrarily using PLL
Hibernate	Operates in lower power mode using backup battery
UART	Eight UART modules with maximum baud rate of 10 Mbps
SPI	Four SPI modules with transmit & receive FIFOs
I2C	Four I2C bus interfaces configurable as master or slave
CAN	Two CAN modules supporting protocol versions 2.0 A/B
USB	One USB device interface
32-bit Timers	Six 32-bit timers with each timer constructed from two 16-bit timers concatenated. Can be used as 16-bit timer
64-bit Timers	Six 64-bit timers with each timer constructed from two 32-bit timers concatenated. Can be used as 32-bit timer
Watchdog Timers	Two watchdog 32-bit timer modules
ADC	Two analog to digital converters with 12-bit resolution
Comparators	Two analog comparators
QEI	Two quadrature encoder inputs
EEPROM	2 KB on chip EEPROM for storing configuration and other data

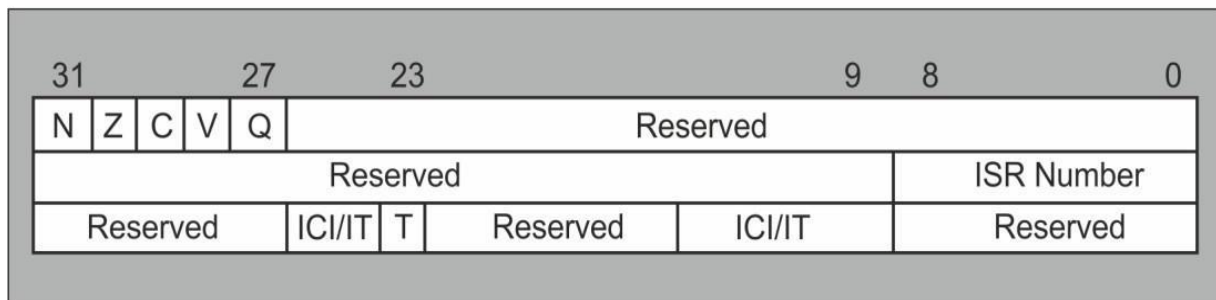
Table 1: Description of TM4C123 peripherals

Register Set:

Registers are high-speed storage inside the processor. The registers are depicted in below figure. R0 to R12 are general purpose registers and contain either data or addresses. Register R13 (also called the stack pointer, SP) points to the top element of the stack. Register R14 (also called the link register, LR) is used to store the return location for functions. The LR is also used in a special way during exceptions, Register R15 (also called the program counter, PC) points to the next instruction to be fetched from memory. The processor fetches an instruction using the PC and then increments the PC.



There are three status registers named Application Program Status Register (APSR), the Interrupt Program Status Register (IPSR), and the Execution Program Status Register (EPSR) as shown below. These registers can be accessed individually or in combination as the Program Status Register (PSR). The N, Z, V, C, and Q bits give information about the result of a previous ALU operation. In general, the N bit is set after an arithmetical or logical operation signifying whether or not the result is negative. Similarly, the Z bit is set if the result is zero. The C bit means carry and is set on an unsigned overflow, and the V bit signifies signed overflow. The Q bit indicates that “saturation” has occurred.



The T bit will always be 1, indicating the ARM® Cortex™-M processor is executing Thumb® instructions. The ISR_NUMBER indicates which interrupt if any the processor is handling. Bit 0 of the special register PRIMASK is the interrupt mask bit. If this bit is 1, most interrupts and exceptions are not allowed. If the bit is 0, then interrupts are allowed. Bit 0 of the special

register FAULTMASK is the fault mask bit. If this bit is 1, all interrupts and faults are not allowed. If the bit is 0, then interrupts and faults are allowed. The non-maskable interrupt (NMI) is not affected by these mask bits. The BASEPRI register defines the priority of the executing software. It prevents interrupts with lower or equal priority but allows higher priority interrupts. For example, if BASEPRI equals 3, then requests with level 0, 1, and 2 can interrupt, while requests at levels 3 and higher will be postponed.

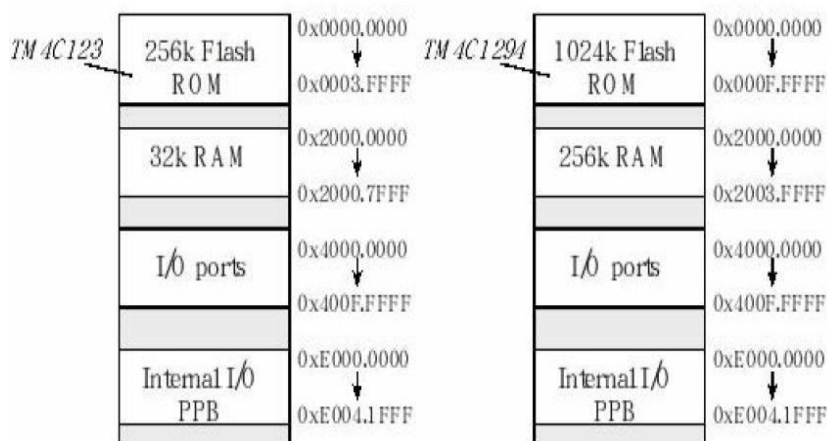
Address Space

Microcontrollers within the same family differ by the amount of memory and by the types of I/O modules. All LM3S and TM4C microcontrollers have a Cortex -M processor. There are hundreds of members in this family; The memory map of TM4C123 is illustrated in above fig. Although specific for the TM4C123, all ARM ® Cortex™-M microcontrollers have similar memory maps. In general, Flash ROM begins at address 0x0000.0000, RAM begins at 0x2000.0000, the peripheral I/O space is from 0x4000.0000 to 0x5FFFF.FFFF, and I/O modules on the private peripheral bus (PPB) exist from 0xE000.0000 to 0xE00F.FFFF. In particular, the only differences in the memory map for the various 180 members of the LM3S/TM4C family are the ending addresses of the flash and RAM. The M4 has an advanced high-performance bus (AHB). Having multiple buses means the processor can perform multiple tasks in parallel. The following is some of the tasks that can occur in parallel

ICode bus Fetch opcodes from ROM

DCode bus Read constant data from ROM

System bus Read/write data from RAM or I/O, fetch opcode from RAM



PPB Read/write data from internal peripherals like the NVIC

AHB Read/write data from high-speed I/O and parallel ports (M4 only)

When we store 16-bit data into memory it requires two bytes. Since the memory systems on most computers are byte addressable (a unique address for each byte), there are two possible ways to store in memory the two bytes that constitute the 16-bit data. Freescale microcomputers implement the big endian approach that stores the most significant byte at the lower address. Intel microcomputers implement the little endian approach that stores the least significant byte at the lower address. Cortex M microcontrollers use the little endian format. Many ARM processors are biendian, because they can be configured to efficiently handle both big and little endian data. Instruction fetches on the ARM are always little endian. Figure 3.5 shows two ways to store the 16-bit number 1000 (0x03E8) at locations 0x2000.0850 and 0x2000.0851. Computers must choose to use either the big or little endian approach when storing 32-bit numbers into memory that is byte (8-bit) addressable. Figure 3.6 shows the big and little endian formats that could be used to store the 32-bit number 0x12345678 at locations 0x2000.0850 through 0x2000.0853. Again the Cortex M uses little endian for 32-bit numbers.

Address	Data	Address	Data
0x2000.0850	0x03	0x2000.0850	0xE8
0x2000.0851	0xE8	0x2000.0851	0x03

Big Endian Little Endian

Figure. Example of big and little endian formats of a 16-bit number.

Address	Data	Address	Data
0x2000.0850	0x12	0x2000.0850	0x78
0x2000.0851	0x34	0x2000.0851	0x56
0x2000.0852	0x56	0x2000.0852	0x34
0x2000.0853	0x78	0x2000.0853	0x12

Big Endian Little Endian

Figure Example of big and little endian formats of a 32-bit number.

In the previous two examples, we normally would not pick out individual bytes (e.g., the 0x12), but rather capture the entire multiple byte data as one non divisible piece of information. On the other hand, if each byte in a multiple byte data structure is individually addressable, then both the big- and little-endian schemes store the data in first to last sequence. For example, if we wish to store the four ASCII characters 'LM3S', which is 0x4C4D3353 at locations 0x2000.0850 through 0x2000.0853, then the ASCII 'L'=0x4C comes first in both big- and little-endian schemes, as illustrated in Figure.

Address	Data
0x2000.0850	0x4C
0x2000.0851	0x4D
0x2000.0852	0x33
0x2000.0853	0x53

Big Endian and Little Endian

Figure . Character strings are stored in the same for both big and little endian formats.

The terms “big and little endian” come from Jonathan Swift’s satire Gulliver’s Travels. In Swift’s book, a Big Endian refers to a person who cracks their egg on the big end. The Lilliputians were Little Endians because they insisted that the only proper way is to break an egg on the little end. The Lilliputians considered the Big Endians as inferiors. The Big and Little Endians fought a long and senseless war over the best way to crack an egg.

Common Error: An error will occur when data is stored in Big Endian by one computer and read in Little Endian format on another.

ARM Addressing Modes

1. Register direct Addressing Mode
2. Direct/Absolute Addressing Mode
3. Immediate Addressing Mode
4. Register Indirect Addressing Mode
5. Register Indirect with pre-indexed Addressing Mode
6. Register Indirect auto-indexed Addressing Mode
7. Register Indirect post-indexed Addressing Mode
8. Register Indirect Register Indexed Addressing Mode
9. PC Relative Addressing mode

1. Register Addressing Mode

- The operand is specified with in one of the processor register.
- Instruction specifies the register in which the operand is stored.

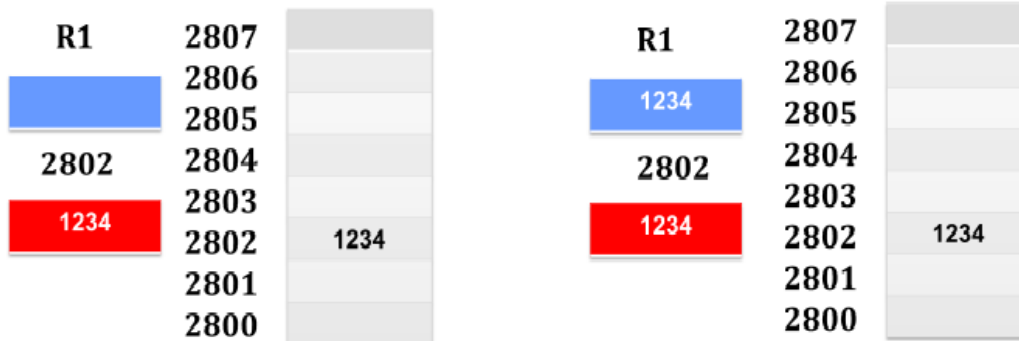
MOV R1 , R2 $R1 \leftarrow R2$ Here R1 is the operand specified in register`



2. Direct/Absolute Addressing Mode

- The operand is specified with in one of the memory location.
- Instruction specifies the register in which the operand is stored.

LDR R1 , 2082 $R1 \leftarrow 2082$ Here R1 is the operand specified in register`



3. Immediate Addressing Mode

- The operand is specified with in the instruction.
- Operand itself is provided in the instruction rather than its address.

Move Immediate
MOV R1, #15h R1 ← 15h Here 15h is the immediate operand

Add Immediate
ADD R1, #3Eh R1 ← R1 + 3Eh Here 3Eh is the immediate operand



MOV

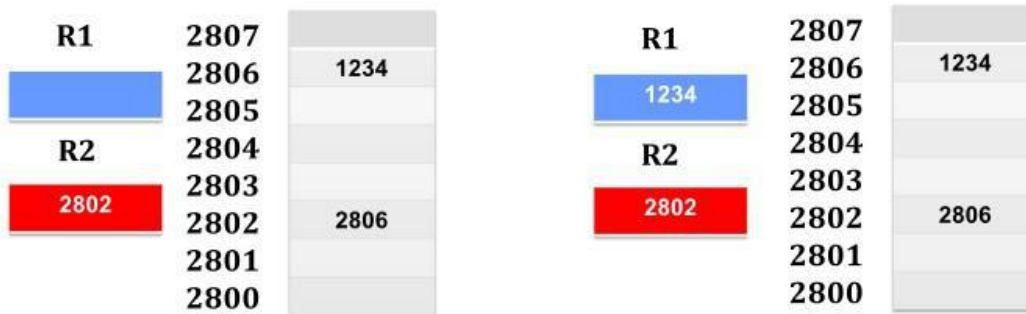


ADD

4. Register Indirect Addressing Mode

- Register indirect addressing means that the location of an operand is held in a register. It is also called indexed addressing or base addressing.
- It moves the data from memory location specified by the location 2802 to **R1**.

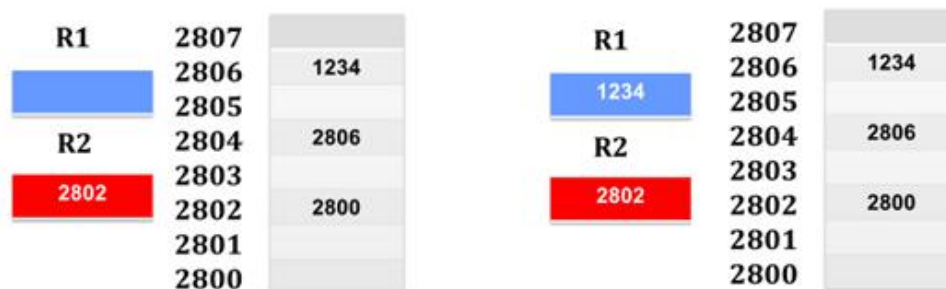
LDR R1 , [R2] R1 ← [2082] Here R1 is the operand specified in register`



5. Register Indirect with pre-index Addressing Mode

- ARM supports a memory-addressing mode where the effective address of an operand is computed by adding the content of a register and a literal offset coded into load/store instruction.

LDR R1 , [R2,#02] R1 ← [2082+2] Here R1 is the operand specified in register`



6. Register Indirect Auto-indexing Addressing Mode

- This is used to facilitate the reading of sequential data in structures such as arrays, tables, and vectors. A pointer register is used to hold the base address. An offset can be added to achieve the effective address.

LDR R1 , [R2,#2]! R1 ← [2082+2] Here R1 is the operand specified in register`

R1	2807	
	2806	5678
	2805	
R2	2804	2806
	2803	
	2802	2800
	2801	
	2800	

R1	2807	
	2806	5678
	2805	
R2	2804	2806
	2803	
	2802	2806
	2801	
	2800	

7. Register Indirect post-indexing Addressing Mode

- This is similar to the above, but it first accesses the operand at the location pointed by the base register, then increments the base register.

LDR R1 , [R2], [#2] R1 ← [2082] Here R1 is the operand specified in register`

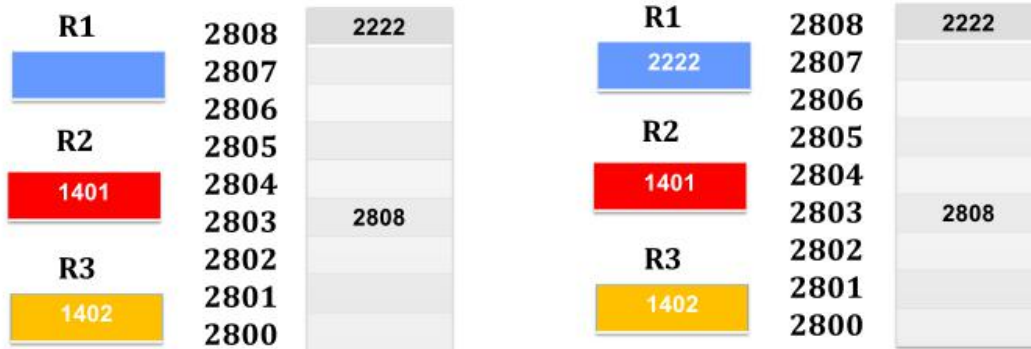
	2808	2222
R1	2807	
	2806	1111
	2805	
R2	2804	2808
	2803	
	2802	2806
	2801	
	2800	

	2808	2222
R1	2807	
	2806	1111
	2805	
R2	2804	2808
	2803	
	2802	2806
	2801	
	2800	

8. Register Indirect Register indexed Addressing Mode

- This is similar to the above, but it first accesses the operand at the location pointed by the base register, then increments the base register.

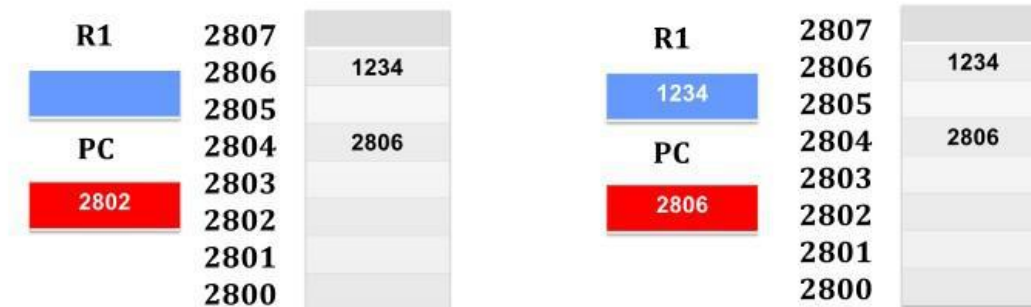
LDR R1 , [R2,R3] $R1 \leftarrow [R2+R3]$ Here R1 is the operand specified in register`



9. PC Relative Addressing Mode

- Register R15 is the program counter. If you use R15 as a pointer register to access operand, the resulting addressing mode is called PC relative addressing. The operand is specified with respect to the current code location.

LDR R1 , [PC, #offset] $R1 \leftarrow [PC+2]$ Here R1 is the operand specified in register`



Instruction set Basics:

1. Data Processing Instructions & Data Movement Instructions

- o Arithmetic/logic Instructions
- o Move instructions
- o Barrel shifting instructions
- o Comparison Instructions
- o Multiply Instructions

2. Branch Instructions

3. Load and store Instructions

1. Data Movement instructions			
Syntax: <instruction>{<condition>}{S} Rd, N			
MOV	Move a 32-bit value into a register	MOV r1, r2, LSL #4	Move (r2<<4) to r1.
MVN	Move the NOT of the 32-bit value into a register	MVN r1, r3	Move (~ r3) to r1.
2. Data Processing instructions			
i. Arithmetic Instructions; Syntax:<instruction>{<cond>}{S} Rd, Rn, N			
ADC	Add two 32-bit values and carry	ADC r1, r2, r3	r1= r2+r3+Carry
ADD	add two 32-bit values	ADD r4, r5, r3, LSR # r1	r4= r5+(r3>> by r1)
RSC	Reverse subtract with carry of two 32-bit values	RSC r3, r2, r1	r3= r1- r2 - !Carry
RSB	Reverse subtract of two 32-bit values	RSB r3, r2, r1	r3= r1- r2
SBC	Subtract with carry of two 32-bit values	SBC r2,r4, r6	r2=r4-r6- !Carry
SUB	Subtract two 32-bit values	SUB r2,r4, r6	r2=r4-r6
ii. Logical Instructions; Syntax:<instruction>{<cond>}{S} Rd, Rn, N			
AND	logical bitwise AND of two 32-bit values	AND r7, r5, r2	r7= r5 & r2
ORR	logical bitwise OR of two 32-bit values	ORR r6, r4, r1, LSR r2	r6= r4 (r1>>r2)
EOR	logical exclusive OR of two 32-bit values	EOR r5, r1, r2	r5= r1 ^ r2
BIC	logical bit clear (AND NOT)	BIC r3, r1,r4	r3= r1 & ~ r4
iii. Comparison Instructions; Syntax:<instruction>{<cond>} Rn, N			
CMN	Compare negated	CMN r1, r2	Flags set as results of r1+r2
CMP	Compare	CMP r1, # 0XFF	Flags set as results of r1-0XFF
TEQ	Test for equality of two 32-bit values	TEQ r3, r5	Flags set as results of r3 ^ r5
TST	Test bits of a 32-bit values	TST r1, r2	Flags set as results of r1& r2
IV. Multiply Instructions; Syntax:MLA{<cond>}{S} Rd, Rm, Rs, Rn; MUL{<cond>}{S} Rd, Rm, Rs			
MLA	Multiply and accumulate	MLA r1,r2,r3,r4	r1=(r2*r3)+r4
MUL	Multiply	MUL r3, r7, r6	R3= r7*r6

3. Branch instructions			
Syntax: B{<cond>} label; BL{<cond>} label; BX{<cond>} Rm; BLX{<cond>} label Rm			
B	Branch	B label	PC= label
BL	Branch with link	BL label	PC=label and Lr= Address of the next instruction after BL.
BX	Branch exchange	BX r5	PC=r5 & 0Xffffffe and T= r5 & 1
BLX	Branch exchange with link	BLX r6	PC=r6 & 0Xffffffe, T=r6 & 1 and lr= address of the next instruction after BLX.
4. Load/Store Instructions			
I. Single register transfer; Syntax:<LDR STR>{<cond>} Rd, Address			
LDR	Load register from memory	LDR r0, [r2, #0X8]	Load r0 with the content of memory address pointed to by [r2+0X8]
STR	Store register to memory	STR r1, [r4], #0X10	Store r1 into the memory address pointed to by r4 and update r4 by [r4+0X10]
II. Multiple register transfer; Syntax:<LDM STM>{<cond>}<addressing mode> Rn{!},{registers}; Addressing modes: IA-Increment after; IB-Increment before; DA-Decrement after; DB-Decrement before:- Increment or decrement the memory pointer after or before the data transfer.			
LDM	Load multiple registers from memory	LDMIA r6!, {r2-r4}	r2=[r6]; r3= [r6+4]; r4=[r6+8] and update r6 by [r6+12]
STM	Store multiple registers to memory	STMDB r1!, {r3-r5}	[r1-4]=r5 [r1-8]=r4 [r1-12]=r3 and update r1 by [r1-12]
III. Stack Operations ; Syntax:<LDM STM>{<cond>}<addressing mode> SP{!},{registers}; Addressing modes: FA-Full ascending ; FD-Full descending ;EA-Empty ascending ;ED –Empty descending;			
LDM	Load multiple registers from stack memory	LDMED Sp!, {r1, r3}	r1= [Sp+4] r2= [Sp+8] r3= [Sp+12] and Sp is updated by [Sp+12]
STM	Store multiple registers to stack memory	STMFD Sp!, {r4,r6}	[Sp-4]= r6 [Sp-8]= r5 [Sp-12]=r4 and Sp is updated by [Sp-12]
IV. Swap instruction ; Syntax: SWP{B}{<cond>} Rd,Rm,[Rn]			
SWP	swap a word between memory and a	SWP/SWPB	Load a 32 bit word

UNIT 3

UNIT- III Overview of Microcontroller and Embedded Systems

Embedded hardware and various building blocks, Processor Selection for an Embedded System , Interfacing Processor, Memories and I/O Devices, I/O Devices and I/O interfacing concepts, Timer and Counting Devices, Serial Communication and Advanced I/O, Buses between the Networked Multiple Devices.Embedded System Design and Co-design Issues in System Development Process, Design Cycle in the Development Phase for an Embedded System, Uses of Target System or its Emulator and In-Circuit Emulator (ICE), Use of Software Tools for Development of an Embedded System Design metrics of embedded systems - low power, high performance, engineering cost, time-to-market.

Embedded hardware and various building blocks:

The software embeds into hardware. Hardware consists of number of building blocks in a circuit board or in ASIC or on the SoC along with the processors. Hardware consists of following building blocks and devices in a system in general.

1. Power Source :

Various units in an embedded system operate in one of the ranges $5.0V \pm 0.25 V$, $3.3 V \pm 0.3V$, $2.0 V \pm 0.2 V$ and $1.5V \pm 0.2V$.

2. Clock Oscillator and Clocking Unit(s):

The clock controls the time for executing an instruction, The frequency depends on the processor circuit execution rate.

3. System Timer:

A timer circuit is suitably configured and functions as system clock. The system clock ticks and generates system-interrupts periodically. The system-clock interrupt enables execution of the system supervisory functions in the OS at the periodic intervals. System clock ticks can be 60 times in second.

4. Real-Time Clock (RTC):

A real-time clock is required in a system. The clock drives the timers for various timing and counting needs in a system. The clock also updates time and date in the system. A microcontroller provides the timer circuits for the counting and timing devices.

5. Reset Circuit, Power-up Reset and Watchdog-Timer Reset

A circuit for **reset** enables restart of the system from the beginning using a switch or signal. The reset can also be performed by using an instruction to the processor. A **power-up reset** circuit enables restart of the system from beginning whenever power is switched on in the system. A **watchdog-timer reset** enables the restart of system when it is stuck up in certain set of instructions for a period more than preset time-interval. Reset based on reset-switch, reset-instruction or power-up or power-up reset and watchdog reset can be from same starting instruction or the different starting instructions.

6. Memory

Various forms of memories are used in a system. Figure shows a chart for the various forms of memories that are present in systems. These are as follows:

- (a) Internal RAM
- (b) Internal ROM/PROM/EPROM
- (c) External RAM for the temporary data and stack (in most systems)
- (d) Internal caches (in pipelined and superscalar microprocessors)
- (e) Internal EEPROM or flash
- (f) Memory Stick (card): video, images, songs, or speeches and large storage in digital camera, mobile systems
- (g) External ROM or PROM for embedding software (in almost all other than microcontroller-based systems)
- (h) RAM Memory buffers at the ports

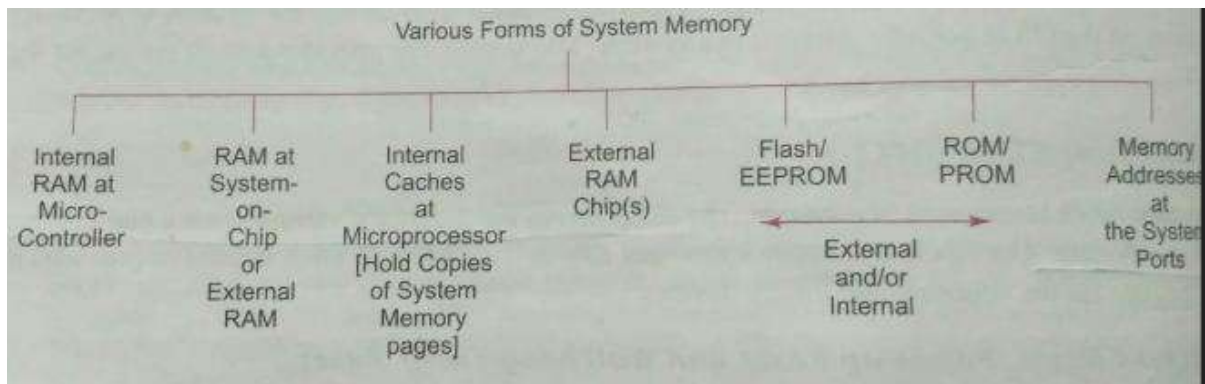


Fig. The various forms of memory in the system

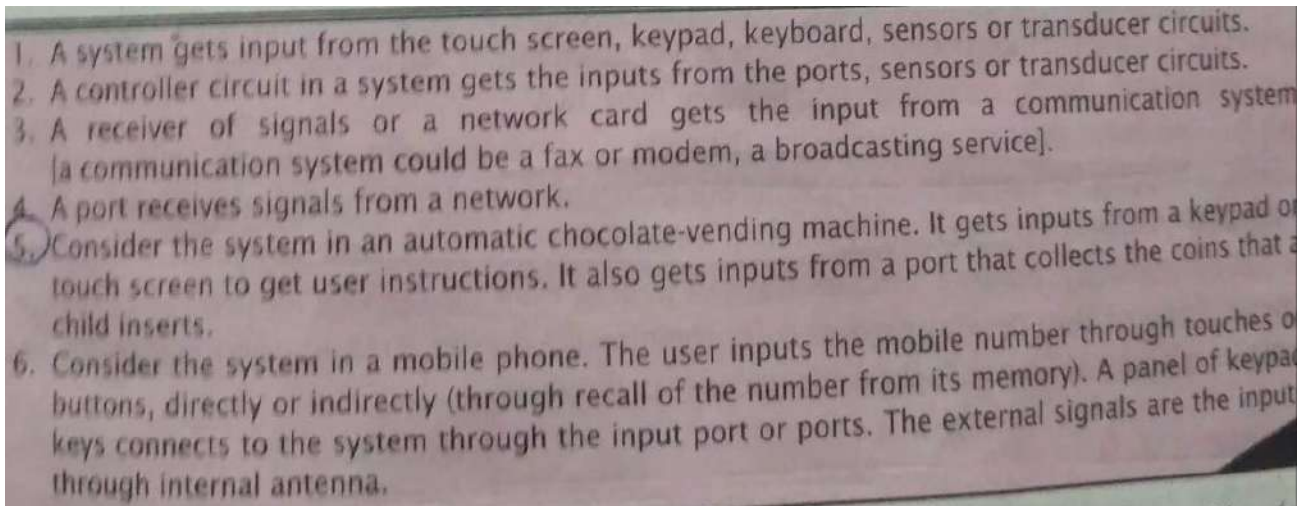
Table gives the functions assigned in the embedded system to the memories, ROM or PROM or EPROM embeds the embedded software specific to the system.

Memory Needed	Functions
ROM or EPROM or Flash	Storing application programs from where the processor fetches the instruction codes. Storing codes for system booting, initialising, initial input data and strings. Codes for RTOS. Pointers (addresses) of various Interrupt Service Routines (ISRs).
RAM (Internal and External) and RAM for Buffer	Storing the variables during program run and storing the stack. Storing input or output buffers, for example, for speech or image.
Memory Stick	A flash memory stick is inserted in mobile computing systems and digital camera. It stores high-definition video, images, songs or speeches after a suitable format compression, and stores large persistent data.
EEPROM or Flash	Storing nonvolatile results of processing.
Caches	Storing copies of instructions and data from external memories in advance and storing the results temporarily during fast processing.

A system embeds(locates) the following either in the internal ROM, PROM or in an external ROM or PROM of microcontroller; boot-up programs, initialization data, strings for an initial screen-display or initial state of the system, the programs for various tasks, ISRs and operating system kernel. The system has RAMs for saving temporary data, stack and buffers that are needed during a program run. The system also has flash for storing non-volatile results.

7 Input, Output and I/O Ports, I/O Buses and I/O interfaces

The system gets inputs from physical devices through the input ports. Following are the examples

- 
1. A system gets input from the touch screen, keypad, keyboard, sensors or transducer circuits.
 2. A controller circuit in a system gets the inputs from the ports, sensors or transducer circuits.
 3. A receiver of signals or a network card gets the input from a communication system [a communication system could be a fax or modem, a broadcasting service].
 4. A port receives signals from a network.
 5. Consider the system in an automatic chocolate-vending machine. It gets inputs from a keypad or touch screen to get user instructions. It also gets inputs from a port that collects the coins that a child inserts.
 6. Consider the system in a mobile phone. The user inputs the mobile number through touches of buttons, directly or indirectly (through recall of the number from its memory). A panel of keypad keys connects to the system through the input port or ports. The external signals are the input through internal antenna.

A processor identifies each input port by its memory buffer address, called port address. Just as a memory location holding a byte or word is identified by an address, each input port is identified by the address. The system gets the inputs by the read operations at the port addresses. The system has output ports through which it sends output bytes to the real world.

Each output port is identified by its memory-buffer address(es) called port address. The system sends the output by a write operation to the port address. There are also general- purpose ports for both the input and output (I/O) operations.

8. Bus

Processor of a system might have to be connected to a number of other devices and systems. A bus consists of a common set of lines to interconnect the multiple devices, hardware units and systems. It enables the communication between two units at any given instance. The remaining units remain in an in connected state during communication between these two. A bus communication protocol specifies the ways of communication of signals on the bus. At any instance, a bus may be a serial bus or a parallel bus transferring one bit or multiple data bits respectively. Protocol also specifies ways of arbitration when several devices need to communicate through the bus. Alternatively, protocol specifies ways of polling from each device for need of the bus at an instance. Protocol also specifies ways of daisy chaining the devices so that, at an instance, the bus is -anted to a device according to the device priority in the chain.

9. Digital to Analog conversion (ADC):

DAC is a circuit that converts digital 8, 10 or 12 bits to analog output. A DAC operation is done with the help of a combination of the Pulse Width Modulation (PWM) unit in a microcontroller and an external integrator chip.

10. Analog to Digital Conversion (ADC):

ADC is a circuit that converts the analog input to digital. The output is of 4, 8, 10 or 12 bits from an ADC. Analog input is applied between + and – pins. ADC circuit converts them into bits. The converted bits value depends on the reference voltage. When input +ve and -ve pins are at voltage equal to reference +ve and -ve voltage pins, then all output bits = 1. When the difference in voltage at inputs +ve and -ve pins = 0v then all output bits = 0.

An ADC unit in the embedded system microcontroller may have multichannels. It can then take the inputs in succession from the various interconnected to different analog sources.

11. LED, LCD and Touch-Screen Displays:

A system requires an interfacing circuit and software to display. LED is used for indicating ON status of the system. A flashing LED may indicate that, a specific task is under completion or is running. It may indicate a wait status for a message. The display may show the status or message. Display may be a line display, a multiline display or a flashing display. An LSI (Lower scale integrated circuit) is used as display controller in case of the LCD matrix display.

Touch screen is an input as well as output device, which is used by the user of a system to enter a command, choose a menu or to give user reply as input. The input is on physical touch as a screen position. The touch at a position is mostly by the finger or some times by stylus. Stylus is a thin pencil shaped long object. It is held between the fingers and used just as a pen to mark a dot. An LSI (Lower scale integrated circuit) functions as touch-screen controller. The display-screen display is similar to a computer display unit screen.

12. Keypad, Keyboard or Virtual Keypad at touch screen:

The keypad or keyboard, is an important device for getting user inputs. A touch screen provides virtual keypad in a mobile computing system. Virtual keypad is a keypad displayed on the LCD display screen on touch plate. A user can enter the inputs using touches. A tile is displayed for a command on the LCD display screen. User can enter the command using touch at the tile.

A keypad or keyboard may interface to a system. The system may provide necessary interfacing circuit and software to receive inputs directly from the keys or touch screen controller

13. Interrupt Handler:

A system may process a number of devices. The system processor controls and handles the requirements of each device by running an appropriate ISR for each interrupting event, An interrupts-handling mechanism must exist in each system. It handles interrupts from various events or processors in the system. The system handled multiple interrupts, which may be simultaneously pending for service.

Processor Selection for an Embedded System:

Processor Selection: Different systems require different features. A hardware designer takes these into view and selects an optimum performance-giving processor. A system designer uses the instruction cycle time as indicator to match the processor speed with the application.

1. A processor, which can operate at higher clock speed, processes more instructions per second.
2. A processor gives high computing performance when there exist (a) Pipelines and superscalar architectures, (b) prefetch cache unit, caches, register files and MMU, and (c) RISC core architecture and most instructions of single clock cycle.
3. A processor with register windows provides fast context switching in a multitasking system.
4. Code-efficient instruction set and when needed dual 16/32 bit instructions set required for smaller memory needs and higher energy efficiency due to less number of memory fetches.
5. A power-efficient embedded system requires a processor that has auto shut-down feature for its units and programmability for disabling these when the processing need for a function or instruction set does not have constraint of execution time. It is also required to have Stop, Sleep and wait instructions. It may also require special cache design.
6. A processor with burst-mode access external memories fast, reads fast and writes fast.
7. It takes into account when considering a processor that an embedded system has to be energy efficient. A processor must have auto shutdown features in its various structural units when these are not employed during a particular time interval. Processor thus has high computing power at lower power dissipation

Processor or Microcontroller Version Selection

The processor and microcontroller selection process needs the following parameters:

1. Processor instruction cycle in ns (typical)
2. Internal bus width in Bits
3. CISC or RISC architecture
4. Pipeline and superscalar architecture
5. On-Chip RAM and/or register file bytes
6. instruction cache
7. Data cache
8. Program memory EPROM/EEPROM/Flash
9. Program memory capacity in bytes
10. Data/Stack memory capacity in bytes
11. Main memory Harvard or Princeton architecture

12. External interrupts
13. Bit manipulation instructions
14. Floating-point processor
15. Interrupt controller
16. DMA controller channels
17. On-chip MMU

Microcontroller Version Selection:

There are numerous versions of 8051. Additional devices and units are provided in these versions. A version is selected for embedded system design as per the application as well as its cost.

1. An embedded system in automobile for example requires CAN bus. Then a version with CAN bus controller is selected.
2. An 8051 enhancement, 8052, has an additional timer.
3. Philips P83C528 has I²C serial bus.
4. The 8051-family member 83C152JA (and its sister JB., JC and JD) microcontrollers) have two direct memory access (DMA) channels on-chip. The 805196K has a PTS (Peripheral Transactions Server) that supports DMA functions. [Only single and bulk transfer modes are supported, not the burst transfer mode] When a system requires direct transfer to memory from external systems, the DMAC is used so that the system performance improves by a separate processing unit for the data transfers to and to the peripherals.

Interfacing Processor, Memories and I/O Devices:

Computer-System Buses:

Bus is set of parallel lines (wires) which carry signals from one unit to another unit. Bus lines interconnect several units, but at a given instance, only two of them communicate. Bus enables interconnections among many units in a simple way. The signals are in specific sequences according to a method or protocol. Computing system buses are as follows:

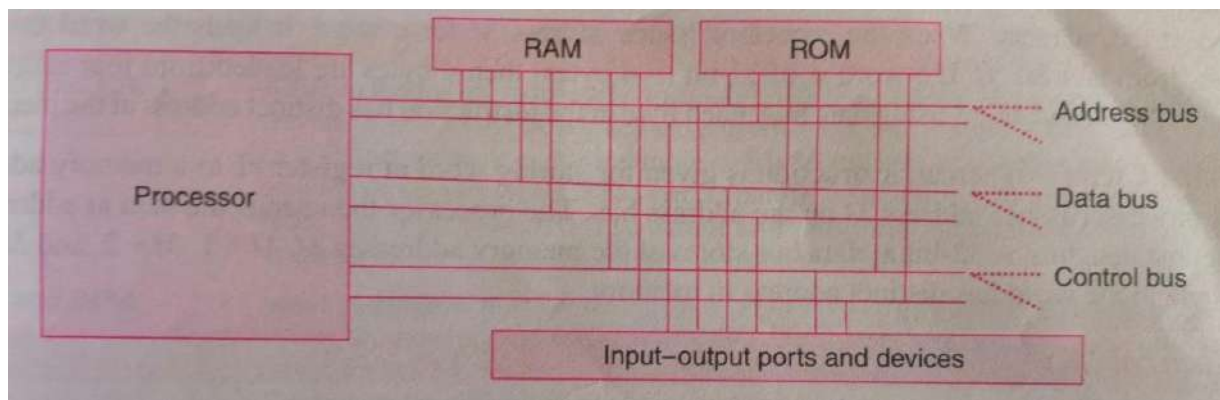
- System Bus also called memory bus, which interconnects the subsystems. It interconnects the processor to memory system and other hardware units. This bus has high speed and bandwidth. It is according to the processor, memory system bandwidth, and for read-write cycles of instructions and data. (Bandwidth means number of bits transferred per second.)
- I/O Bus also called peripheral bus. It interconnects the memory bus to a variable number of I/O devices functioning at variable speeds. Devices or peripherals are designed to interface to the I/O bus. The devices can be attached or withdrawn from the I/O bus at any time.

CPU/Microprocessor System buses Computing system hardware consists of processor, memory and I/O units (ports, devices and peripherals). System bus enables the

interconnections among multiple subsystems in the system CPU/microprocessor/processor in a computing system interconnects to memory, I/O units, devices and peripherals through a bus, called system bus or system memory bus. Three sets of signals—classified as address bus, data bus and control bus define the system bus. A system-bus interfacing-design is according to the timing diagram of processor signals, bus bandwidth and word length. A simple structure of system bus is that same bus, which connects the memory also connects the other subsystems (I/O units, ports, devices and peripherals). Processor interfaces memory as well as I/O devices using system memory bus. Figure (a) shows the interfacing of processor, memory and I/O devices using memory system bus in a simple bus structure.

1. Address Bus

Address bus signals are from processor to memory or other interfaced units. Address bus is unidirectional. When it has 'n' signals A0 to An-1, the Processor issues (send) addresses between 0 and 2^n-1 using the bus. The processor issues the address of the instruction byte or word to the memory system. The address bus communicates the address to the memory. The address bus of 32 bits fetches the instruction or data when an address specified by a 32-bit number, between 0 and $2^{32}-1$.



Fig(a) Interfacing of processor, memory and I/O devices using memory system bus

2. Data Bus

A data bus is bidirectional. If it has 'm' bits then signals are D0-Dm-1 and processor reads a word or instruction or writes a word. Data signals are from processor to memory during read cycle and memory to processor during write cycle. A data bus of 8, 16, 32 or 64 bits fetches, loads, or stores the instruction or data.

- **Read Cycle:** Read cycle means a sequence of signals during which using the data-bus processor (i) fetches instruction from program-memory section of memory, or (ii) loads data word from data-memory section of memory. When the processor issues the address of the instruction, it gets back the instruction from memory through the data bus. When it issues the address of the data it processor loads the data through the data bus into a register.
- **Write Cycle:** Write cycle means a sequence of signals during which, using data bus, the processor sends data word to data-memory address in memory. When it issues the address of the data it stores the data in the memory through the data bus.

3. Control Bus

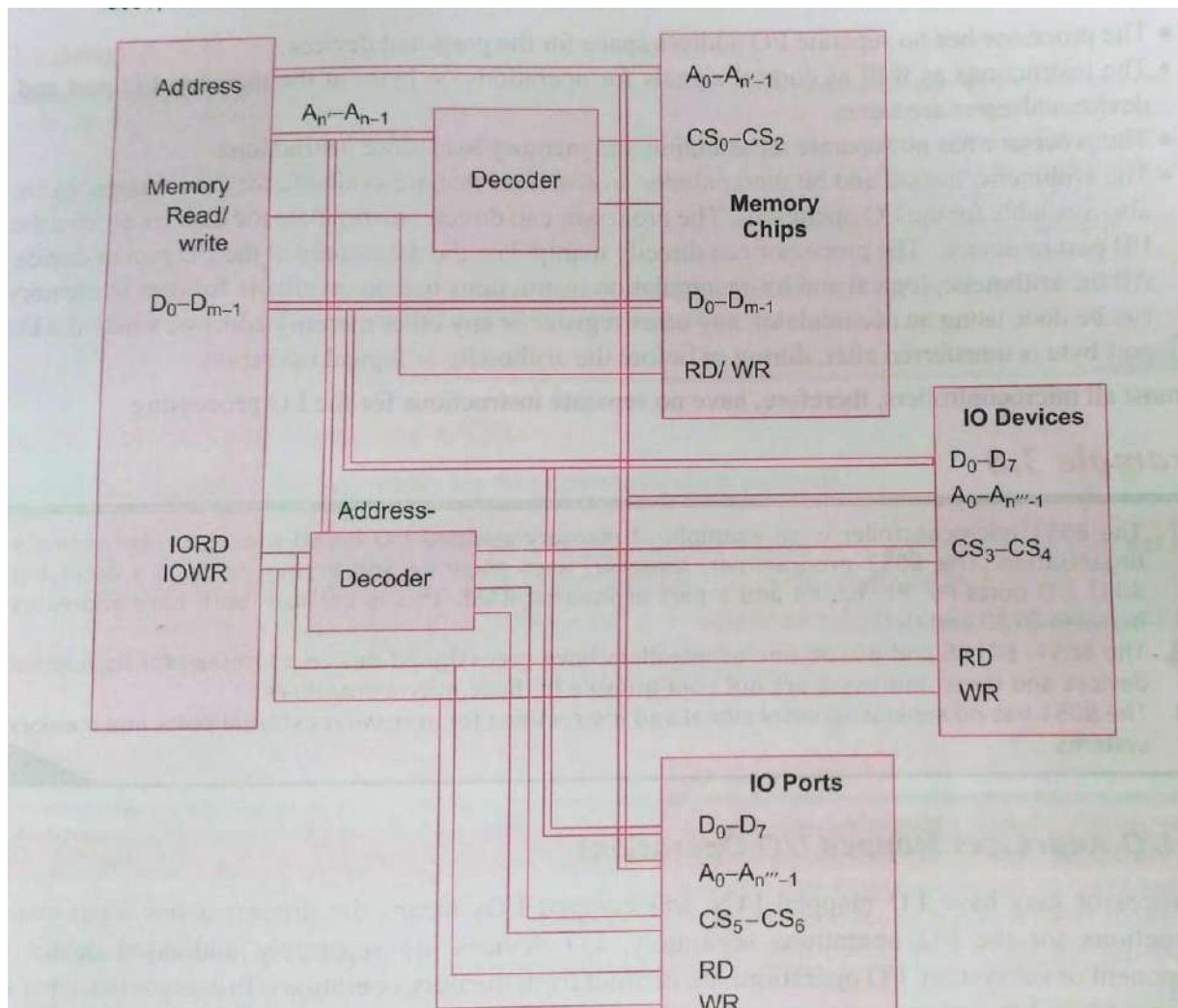
A control bus issues signals to control the timing of various actions during communication of signals. These signals synchronize the subsystems.

(a) When the processor issues the address, after allowing sufficient time for the set-up of all address bits, it also issues a memory-read control signal and waits for the data or instruction after a time interval. A memory unit must place the instruction or data during the interval in which memory-read signal is active (not inactivated by the processor).

(b) When the processor issues the address on the address bus, and (after allowing sufficient time for the set-up of all address bits) it places the data on the data bus, it also then issues memory-write control signal (after allowing sufficient time for the set-up of all data bits) for store signal to memory. The memory unit must write (store) the data during the interval in which memory-write is active (not inactivated by the processor).

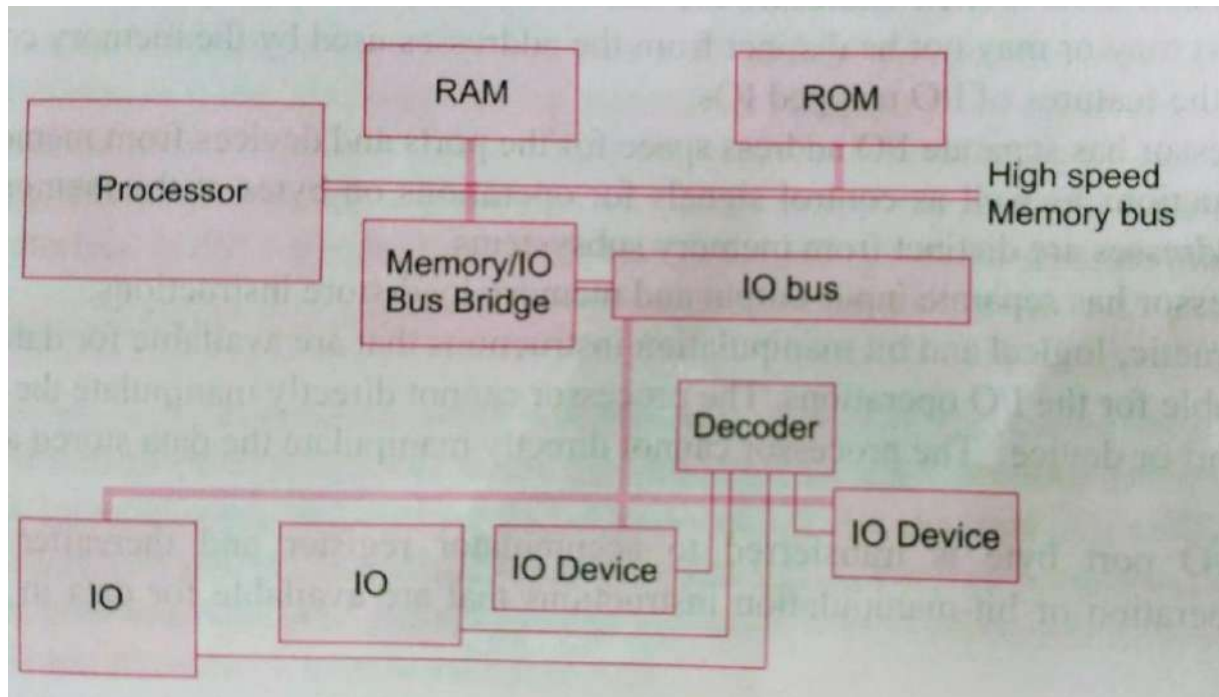
I/O Devices and I/O interfacing concepts:

Interfacing methods for I/O device:



Fig(a) I/O devices and components interfacing circuit with ports

- **Method 1:** I/O devices or components interface using ports, interfacing circuit consists of decoder. The decoder circuit connects the processor address bus and control signals. A port select output of the decoder is active when the address input corresponds to the port address. Interfacing circuit is designed as per available control signals and timing diagrams of the system bus signals. Fig(a) shows interface using ports.



Fig(b) I/O devices and components interfacing circuit using I/O bus

- **Method 2:** A method is interfacing through an I/O bus. Interfacing circuit consists of I/O controller (called bridge or switch also). The switch circuit connects the processor and memory on system memory bus with the I/O bus. Interfacing-circuit is designed as per available control signals and timing diagrams of the system bus signals and I/O bus. Fig(b) shows interface using I/O bus and switch circuit between system memory bus and I/o bus.

I/O managing data

Memory mapped I/O and I/O mapped I/O operations are two types of operations based on processor and memory organisation.

Memory mapped I/O is mapped into the same address space as program memory and/or user memory, and is accessed in the same way.

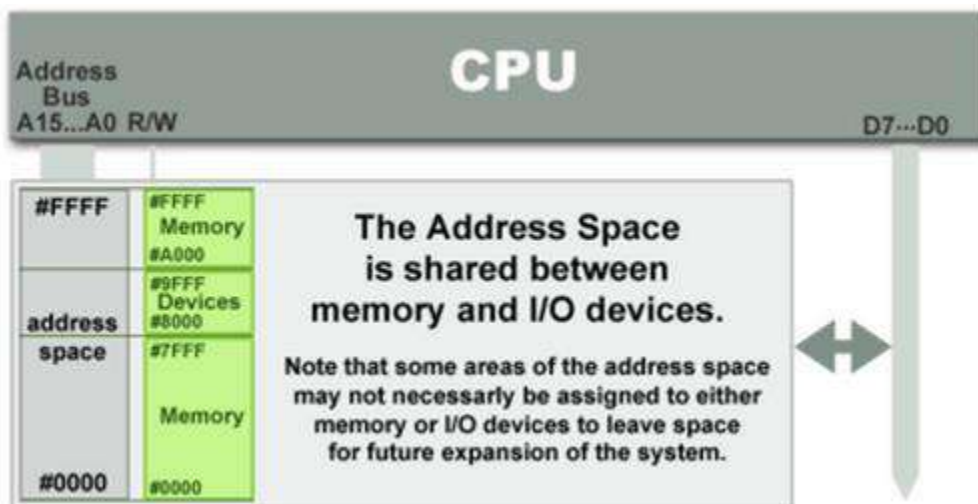
Port mapped I/O uses a separate, dedicated address space and is accessed via a dedicated set of microprocessor instructions.

The difference between the two schemes occurs within the microprocessor. Intel has, for the most part, used the port mapped scheme for their microprocessors and Motorola has used the memory mapped scheme.

As 16-bit processors have become obsolete and replaced with 32-bit and 64-bit in general use, reserving ranges of memory address space for I/O is less of a problem, as the memory address space of the processor is usually much larger than the required space for all memory and I/O devices in a system.

Therefore, it has become more frequently practical to take advantage of the benefits of memory-mapped I/O. However, even with address space being no longer a major concern, neither I/O mapping method is universally superior to the other, and there will be cases where using port-mapped I/O is still preferable.

Memory-mapped IO (MMIO):



I/O devices are mapped into the system memory map along with RAM and ROM. To access a hardware device, simply read or write to those 'special' addresses using the normal memory access instructions.

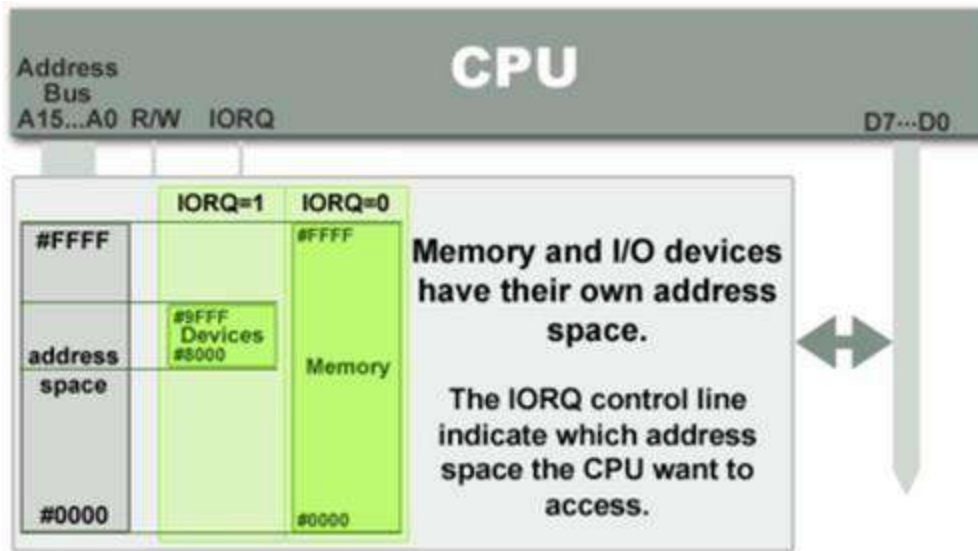
The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device.

The disadvantage to this method is that the entire address bus must be fully decoded for every device. For example, a machine with a 32-bit address bus would require logic gates to resolve the state of all 32 address lines to properly decode the specific address of any device. This increases the cost of adding hardware to the machine.

Port-mapped IO (PMIO or Isolated IO):

I/O devices are mapped into a separate address space. This is usually accomplished by having a different set of signal lines to indicate a memory access versus a port access. The address lines are usually shared between the two address spaces, but less of them are used for accessing ports. An example of this is the standard PC which uses 16 bits of port address space, but 32 bits of memory address space.

The advantage to this system is that less logic is needed to decode a discrete address and therefore less cost to add hardware devices to a machine. On the older PC compatible machines, only 10 bits of address space were decoded for I/O ports and so there were only 1024 unique port locations; modern PC's decode all 16 address lines. To read or write from a hardware device, special port I/O instructions are used.



From a software perspective, this is a slight disadvantage because more instructions are required to accomplish the same task. For instance, if we wanted to test one bit on a memory mapped port, there is a single instruction to test a bit in memory, but for ports we must read the data into a register, then test the bit.

Memory-mapped IO	Port-mapped IO
Same address bus to address memory and I/O devices	Different address spaces for memory and I/O devices
IO is treated as memory	IO is treated as IO
16-bit addressing is used	8-bit addressing is used
More decoder hardware is used	Less decoder hardware is used
Can access $2^{16} = 64k$ locations theoretically	Can address $2^{16} = 256$ locations
Access to the I/O devices using regular instructions	Uses a special class of CPU instructions to access I/O devices
Memory instructions are used	Special IN and OUT instructions
Arithmetic and logic operations can be performed directly on data	Arithmetic and logic operations cannot be performed directly on data
Data transfer b/w register and I/O	Data transfer b/w accumulator and I/O

I/O TYPES AND EXAMPLES:

A port at a device can transmit (send) or receive through wire or wireless. Input port means a circuit to where bit or bits can be input (received) from an external device, peripheral or system. Output port means a circuit from where bit or bits can be output (sent) to an external device, peripheral or system. Input-Output (I/O) port means a circuit where bit(s) can be input or output. There are two types of I/Os, serial and parallel. Serial means in series of successive instants. Parallel means at the same instance.

1. Serial Input Serial input port means a circuit to where bits can be input (received) in successive time intervals. The time interval is known to the receiver port. The port assembles the bits on receiving at successive instances.

2. Serial Output Serial output port means a circuit from where bits can be output (sent) in successive time intervals. A time interval is known to the external device or system. The external device assembles the bits on receiving at successive instances from the output port.

3. Serial I/O Serial Input-Output (I/O) port means a circuit where serially received or sent bits can be input or output.

4. Parallel Input Parallel input port means a circuit where bits can be input (received) at an instant. The processor at the input device, circuit or system reads the bits from the port at next instant.

5. Parallel Output Parallel output port means a circuit from where bits can be output (sent) at an instant. The external device can read the bits at the output port.

6. Parallel I/O Parallel Input-Output (I/O) port means a circuit where received or sent bits in parallel can be input or output.

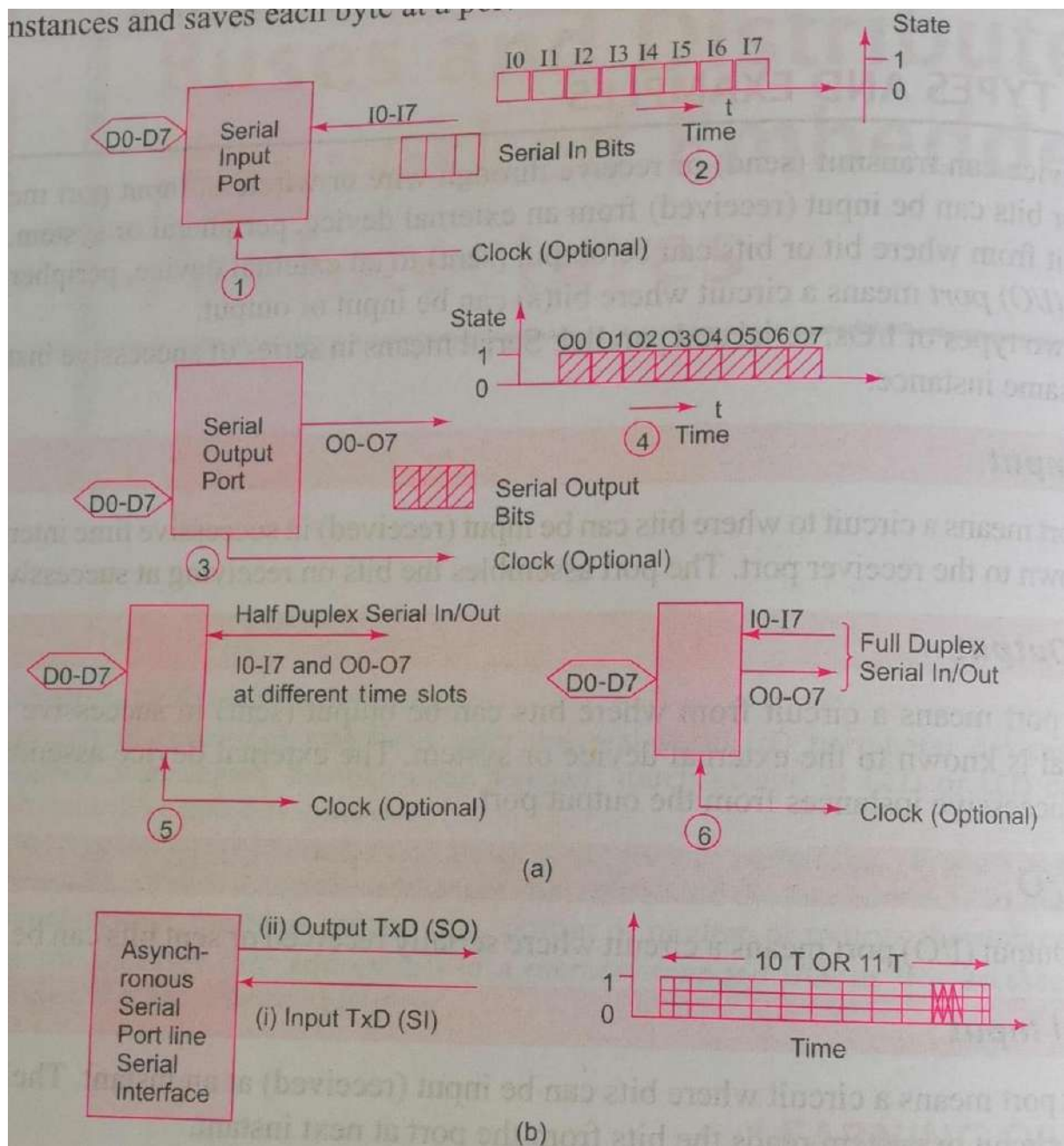
7. I/O Types (i) Serial Input, (ii) Synchronous Serial Output, (iii) Asynchronous Serial Input, (iv) Asynchronous Serial Input, (v) Parallel Port One-bit Input, (vi) Parallel Port One-bit Output, (vii) Parallel Port Input, and (viii) Parallel Port Output.

Synchronous serial input:

The upper part of figure shows a synchronous input serial port at a device or system. Each bit in each byte is in synchronisation and each received byte is in synchronisation. Synchronisation means separation by a constant time interval or phase difference. A port (device) processing element reads a successive instances and saves each byte at a port buffer (register).

Synchronous serial output:

The middle part of Figure shows a synchronous output serial port. The serial port device optionally also sends the clock pulses at clock pin SCLK. Each bit in each byte is in synchronisation with a clock. The serial port at device optionally sends the clock pulses at SCLK pin.



Fig(1.a)

(a) Input serial port (1) and input bits (2), output serial port (3) and output bits (4), bi-directional half-duplex serial I/O port (5), and bi-directional full-duplex serial port (6). (b) (i) Asynchronous input serial port line SI (receive data) and format of received bits (ii) Asynchronous output serial port line SO (transmit data) and format of transmitted bits.

Synchronous Serial Input/Output:

Figure at the left lower part shows a synchronous serial I/O port. Each bit in each byte is in synchronisation at input with a clock input (optional) and each bit in each byte is in synchronisation at output with the clock output (output). The bytes are sent or received at constant rates

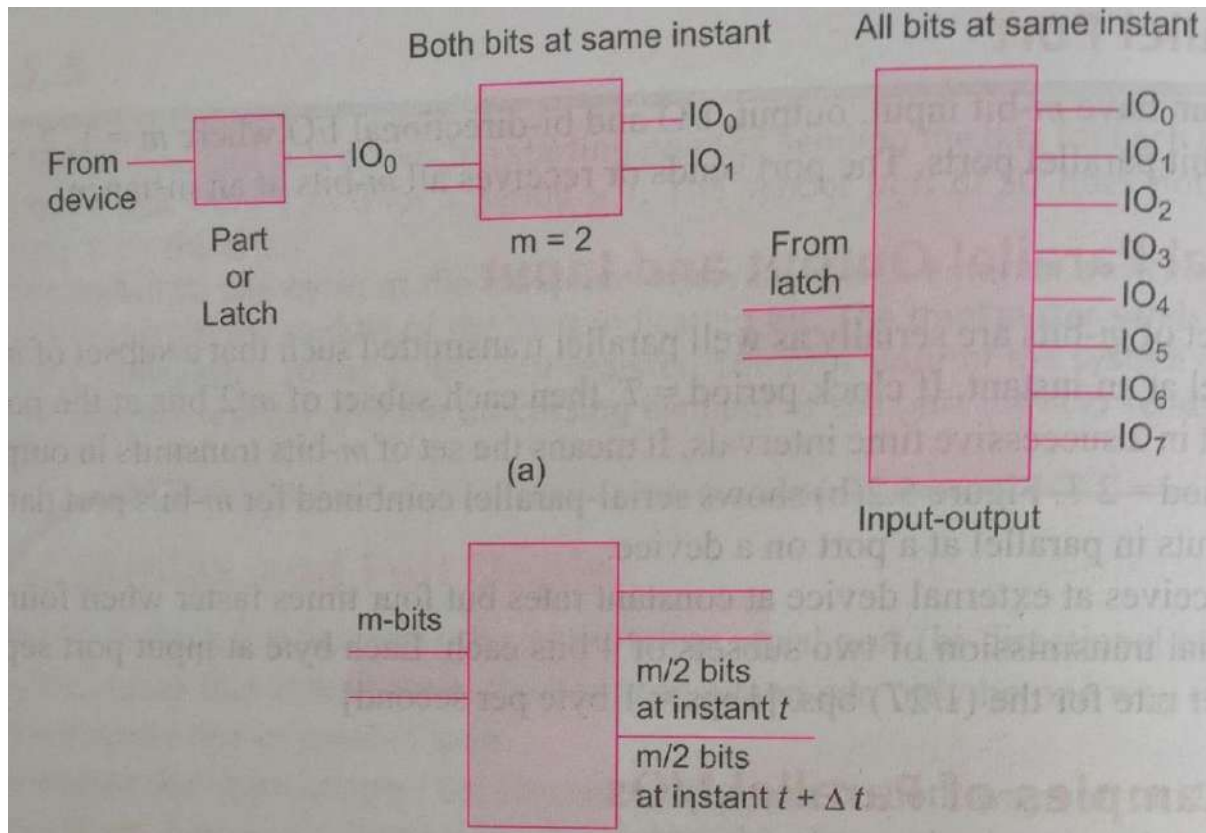
Asynchronous Serial Input:

Figure (b) on the left side shows asynchronous input serial port line SI (receive data). Each bit is received in each byte at fixed intervals but each received byte need not be in synchronisation. The bytes can separate by the variable intervals or phase differences.

Each bit at input port separates by T and bit transfer rate (for the serial line bits) is $(1/T)$ baud per second (bps), but different bytes are received at varying intervals. Baud is taken from a German word for raindrops. Bytes pour from the sender like raindrops at irregular intervals. The sender does not send the clock pulses along with the hits.

Asynchronous Serial Output:

Figure (1.b) on the left side shows as output serial port line SO (transmit data). Each bit in each byte is at fixed intervals but each output byte need not in synchronisation (separates by a variable interval or phase difference).

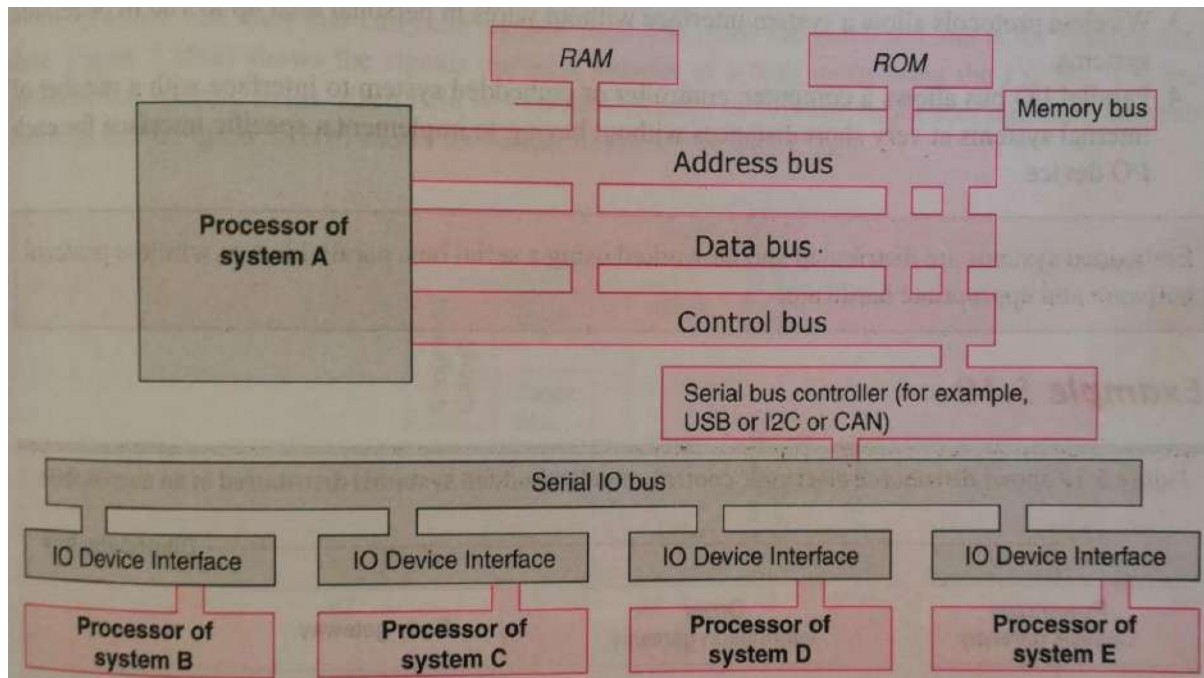


Fig(2) (a)m-bit parallel ports with $m = 1,2,\dots,7,8$ (b) Two $m/2$ bit subsets for output bits in parallel.

Serial Communication and Advanced I/O:

Refer 5th unit “Synchronous/Asynchronous Interfaces (like UART, SPI, I2C, and USB)” topic

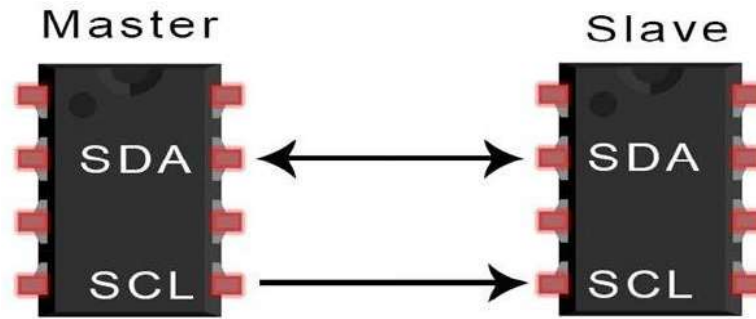
Bus architectures between the Networked Multiple Devices:



A distributed networked system means a number of systems on a common bus or a set of buses, where each system interfaces to a bus. Each bus communicates as per a protocol. Bus communication simplifies the number of connections and provides a common way (protocol) of connecting different or same type of I/O devices. A communication system may use protocols such as UART, I2C, CAN, USB, Wi-Fi or Bluetooth for synchronous or asynchronous transmission from interface at device, or from the system to another interface. Figure 5.16 shows a computer-system bus connected to serial-I/O bus using a bus controller, and I/O bus networking the number of embedded systems distributed on a serial bus.

I2C : Inter integrated circuit (I2C) is important serial communication protocol in modern electronic systems. Philips (now NXP semiconductors) invented this protocol in 1986. The objective of reducing the cost of production of television remote control motivated Philips to invent this protocol. I2C is a serial bus interface, can be implemented in software, but most of the microcontrollers support I2C by incorporating it as hard IP (Intellectual Property). I2C can be used to interface microcontroller with RTC, EEPROM and different variety of sensors. I2C is used to interface chips on motherboard, generally between a processor chip and any peripheral which supports I2C. I2C is very reliable wireline communication protocol for an on board or short distances. I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems.

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.



I2C protocol uses two wires for data transfer between devices: Serial Data Line (SDA) and Serial Clock Line (SCL). The reduction in number of pins in comparison with parallel data transfer is evident. This reduces the cost of production, package size and power consumption. I2C is also best suited protocol for battery operated devices. I2C is also referred as two wire serial interface (TWI).

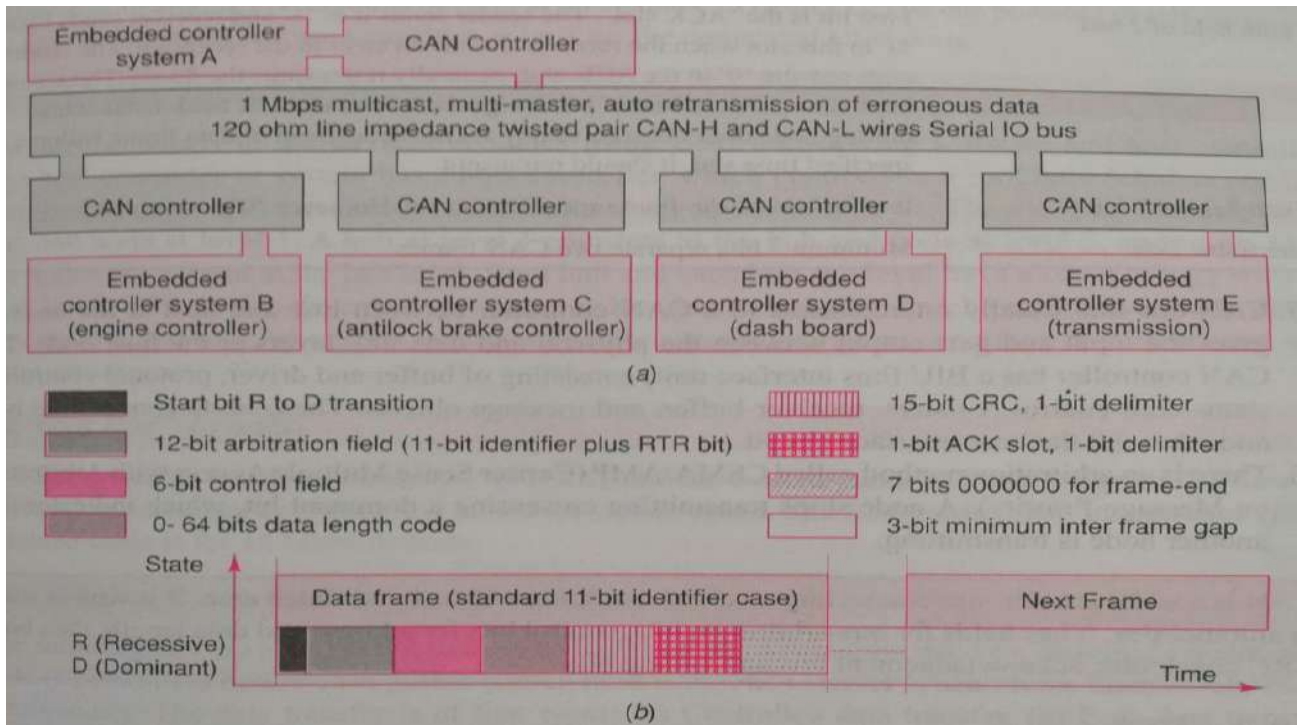
SDA (Serial Data) – The line for the master and slave to send and receive data. SCL (Serial Clock) – The line that carries the clock signal.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

Wires Used	2
Maximum Speed	Standard mode= 100 kbps Fast mode= 400 kbps High speed mode= 3.4 Mbps Ultra fast mode= 5 Mbps
Synchronous or Asynchronous?	Synchronous
Serial or Parallel?	Serial
Max # of Masters	Unlimited
Max # of Slaves	1008

Like SPI, I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

CAN: A number of devices located and are distributed in a Vehicular Control Network automobile uses a number of distributed embedded controllers. The controllers provide the controls for brakes, engines, electric power, lamps, temperature, air conditioning, car gate, front display panels, and cruising.



The embedded controllers are networked and are controlled through a controller network bit. Figure (a) shows a network of number of CAN controllers and CAN devices on a CAN bus. Figure (b) shows six fields and interframe bits during a transfer of data bits on CAN bus, and timing formats and sequences of frame bits.

- The CAN has a serial line, which is bidirectional.
- A CAN device receives or sends a bit at an instance by operating at the maximum rate of 1Mbps.
- It employs a twisted pair connection to each node.
- The pair runs up to a maximum length of 40 m.
- A CAN version also functions up to 2Mbps. CAN (control area network) bus is a standard bus in a distributed network.
- CAN is mainly used in automotive electronics. It is also used in medical electronics and industrial plants.

USB: Universal Serial Bus (USB) is a set of interface specifications for high speed wired communication between electronics systems peripherals and devices with or without PC/computer. The USB was originally developed in 1995 by many of the industry leading companies like Intel, Compaq, Microsoft, Digital, IBM, and Northern Telecom. The major goal of USB was to define an external expansion bus to add peripherals to a PC in an easy and simple manner.

USB offers users simple connectivity. It eliminates the mix of different connectors for different devices like printers, keyboards, mice, and other peripherals. That means USB-bus allows many peripherals to be connected using a single standardized interface socket. It supports all kinds of data, from slow mouse inputs to digitized audio and compressed video.

Various versions USB:

USB1.0: USB 1.0 is the original release of USB having the capability of transferring 12Mbps, supporting up to 127 devices. This USB 1.0 specification model was introduced in January 1996.

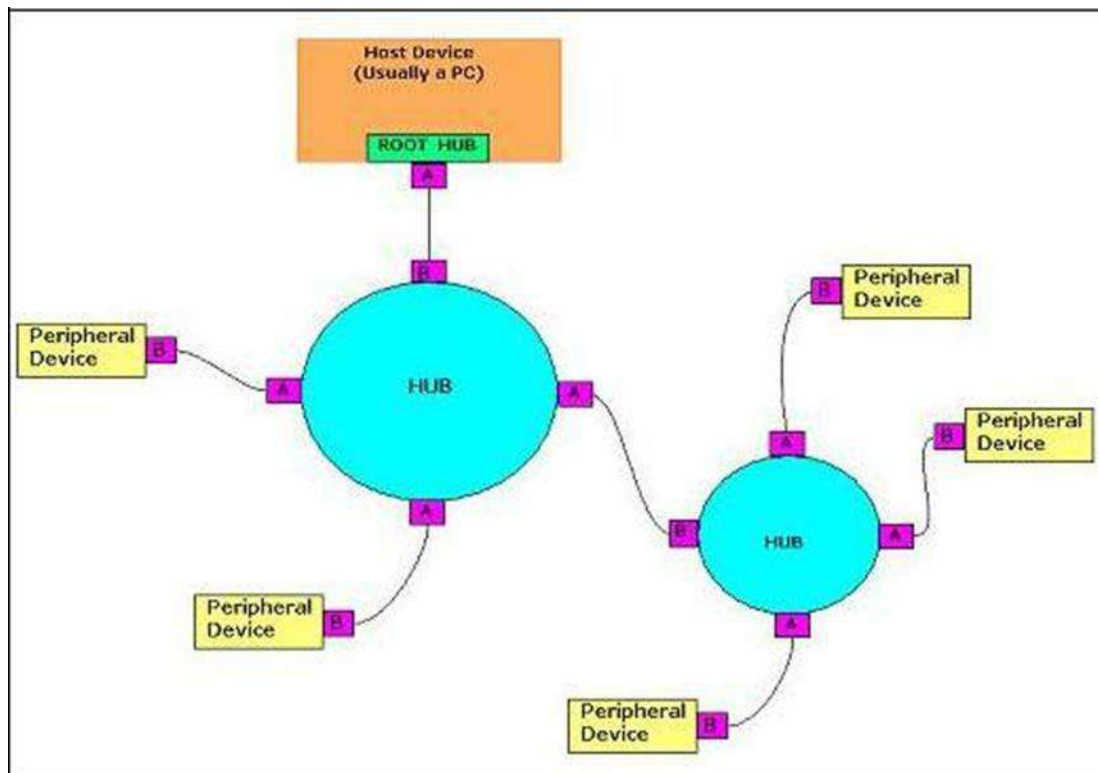
USB1.1: USB 1.1 came out in September 1998. USB 1.1 is also known as full-speed USB. This version is similar to the original release of USB; USB version 1.1 supported two speeds, a full speed mode of 12Mbps/s and a low speed mode of 1.5Mbps/s.

USB2.0: Hewlett-Packard, Intel, LSI Corporation, Microsoft, NEC, and Philips jointly led the initiative to develop a higher data transfer rate than the 1.1 specifications. USB 2.0, also known as hi-speed USB. This hi-speed USB is capable of supporting a transfer rate of up to 480 Mbps, compared to 12 Mbps of USB 1.1. That's about 40 times as fast!

USB3.0: It is also called as Super-Speed USB having a data transfer rate of 4.8Gbps(~5Gbps) That means it can deliver over 10x the speed of today's Hi-Speed USB connections

USB 3.1 : is the latest version of USB also known as Super-Speed USB+, which having data transfer rate of 10Gbps.

The USB system is made up of a host, multiple numbers of USB ports, and multiple peripheral devices connected in a tiered-star topology.



USB can support 4 data transfer types or transfer modes.

1. Control
2. Isochronous
3. Bulk
4. Interrupt

Embedded System Design and Co-design Issues in System Development Process:

There are two approaches for the embedded-system design. (1) The software development life cycle ends and the life cycle for the process of integrating the software into the hardware begin at the time when a system is designed. (2) Both cycles concurrently proceed when co-designing a time-critical sophisticated system.

The final design, when implemented, gives the targeted embedded system, and thus the final product. Therefore, an understanding of the (a) software and hardware designs and integrating both into a system, and (b) hardware-software co-designing are important aspects of designing embedded systems. There is a hardware-software trade-off.

The selection of the hardware during hardware design and an understanding of the possibilities and capabilities of hardware during software design are critical especially for a sophisticated embedded-system development.

Choosing the Right Platform

1. Hardware-software trade-off -There is a trade-off between the hardware and software. It is possible that certain subsystems in hardware, I/O memory access, real-time clock, system clock, pulse-width modulation, timer, and serial communication are also implemented by the software.

Hardware implementation provides the following advantages (i) Reduced memory for the program (ii) Reduced Number of chips but an increased cost (iii) Simple coding for the device drivers (iv) Internally embedded codes, which are more secure than at the external ROM.

Software implementation provides the following advantages: (i) Easier to change when new hardware versions become available (ii) Programmability for complex operations (iii) faster development time (iv) Modularity and portability (v) Use of a standard software-engineering model (vi) RTOS (vii) Faster speed of operation of complex functions with high-speed microprocessors (viii) Less cost for simple systems.

2. Choosing a Right Platform- System design of an embedded system also involves choosing a right platform. A platform consists of a number of following units.

Processor, ASIP or ASSP, Multiple Processors, System-on-Chip, Memory Other Hardware Units of System, Buses, Software language, RTOS, Code generation tools, tools for finally embedding the software into binary image.

3. Embedded System Processors' Choice

■ Processor-Less System -We have an alternative to a microprocessor microcontroller or DSP. Figure (a) shows the use of a PLC in place of a processor. We can use a PLC for the clothes-in clothes-out type system. A PLC fabricates by the programmable gates, PALs GALs, PLDs and CPLDs.

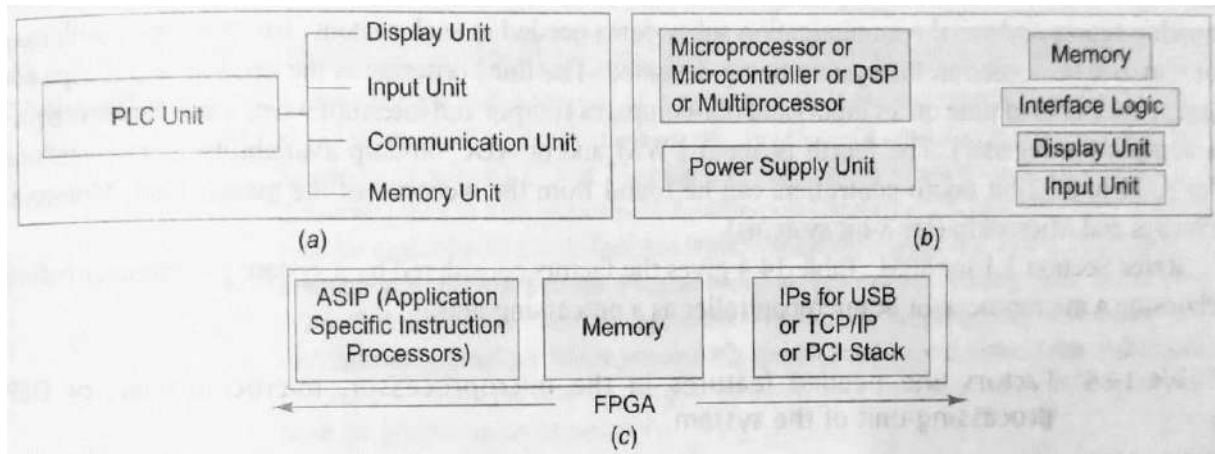


Fig. (a) Use of a PLC in place of a processor (b) Use of a microprocessor, microcontroller or a DSP (c) Processing of functions by using IP embedded into the FPGA instead of processing by the ALU

A PLC has very low operation speed. It also has a very low computational ability. It has very strong interfacing capability with its multiple inputs and outputs. It has system-specific programmability. It is simple in application. Its design implementation is also fast. Automatic chocolate-vending machine another exemplary application of PLC.

■ Fig (b) -System with Microprocessor, Microcontroller or DSP

■ System with Single-purpose Processor or ASSP in VLSI or FPGA Figure © shows the processing of functions in using IP embedded into VLSI or FPGA instead of processing by the ALU

A line of action in designing can be use of the IP, synthesising using V HDL like tool and embedding the synthesis into the FPGA. This FPGA implements the functions, which if implemented with the ALU and programmer coding will take a long time to develop.

■ Factors and Needed Features Taken into Consideration- We consider a general. purpose processor choice or choose an ASIP (microcontroller, DSP or network processor). the 32-bit system, 16 KB- on-chip memory. and need of cache, memory management unit, SIMD, M1MD or DSP instructions arise, we use a microprocessor or DSP.

S. No.	Factors for On-Chip Feature	Needed or which one Needed	Available in Chosen Chip
1	8-bit or 16-bit or 32-bit ALU	8/16/32	8/16/32
2	Cache, Memory Management Unit or DSP Calculations	Yes or No	Yes or No
3	Intensive Computations at Fast Rate	Yes or No	Yes or No
4	Total External and Internal Memory up to or more than 64 kB	Yes or No	Yes or No
5	Internal RAM	256/512 B	256/512 B
6	Internal ROM/EPROM/EEPROM	4 kB/8 kB/16 kB	4 kB/8 kB/16 kB
7	Flash	16 kB/64 kB/1 MB/8 MB	16 kB/64 kB/1 MB/8 MB
8	Timer 1, 2 or 3	1/2/3	1/2/3
9	Watchdog Timer	Yes or No	Yes or No
10	Serial Peripheral Interface Full duplex or Serial Synchronous Communication Interface (SI) Half Duplex	Full/Half	Full/Half

S. No.	Factors for On-Chip Feature	Needed or which one Needed	Available in Chosen Chip
11	Serial UART	Yes or No	Yes or No
12	Input Captures and Out-compares	Yes or No	Yes or No
13	PWM	Yes or No	Yes or No
14	Single or Multichannel ADC with or without programmable Voltage reference (single or dual reference)	S/M W/WO V_{ref} S/D	Yes or No S/M W/WO V_{ref} S/D
15	DMA Controller	Yes or No	Yes or No
16	Power Dissipation	Very low/Low or normal	Very low/Low or normal

Memory and Processor sensitive software

Processor sensitive - A processor has different types of structural units. It can have memory- mapped I/Os or I/O-mapped I/Os. The I/O instructions are processor sensitive. A processor may be having fixed point ALU only. Floating-point operations, when needed, are handled differently than in a processor with floating-point operations.

Memory sensitive - (i) An example of a memory-sensitive program is video processing and real-time video processing. The picture resolution actually used for processing and number of frames processed per second will depend upon the memory available as well as processor performance (ii) Memory address of I/O device registers, buffers, control-registers and vector addresses for the interrupt sources or source groups are prefixed in a microcontroller Memory-sensitive programs need to be optimized for the memory use by skilful programming. (iii) When using certain instruction sets like Thumb in ARM processor helps in 16-bit instructions, which save in less memory space than use of 32-bit ARM instruction set.

Allocation of Addresses to memory, program segments and devices.

1. Functions, Processes, Data and stacks at the various segments of memory

Program routines and processes can have different segments. Each segment has a pointer address and an offset address. Using offset, a code or data word is retrieved from a segment. A stack is a special data structure at the memory. It has a pointer address that always points to the top of a stack. This pointer address is called a stack pointer. The other data sets, which are also allotted memory are as following: String, Circular queue, A one-dimensional array, A table, A hash table, Look-up tables, A list.

2. Device, Internal devices and I/O devices addresses and device drivers

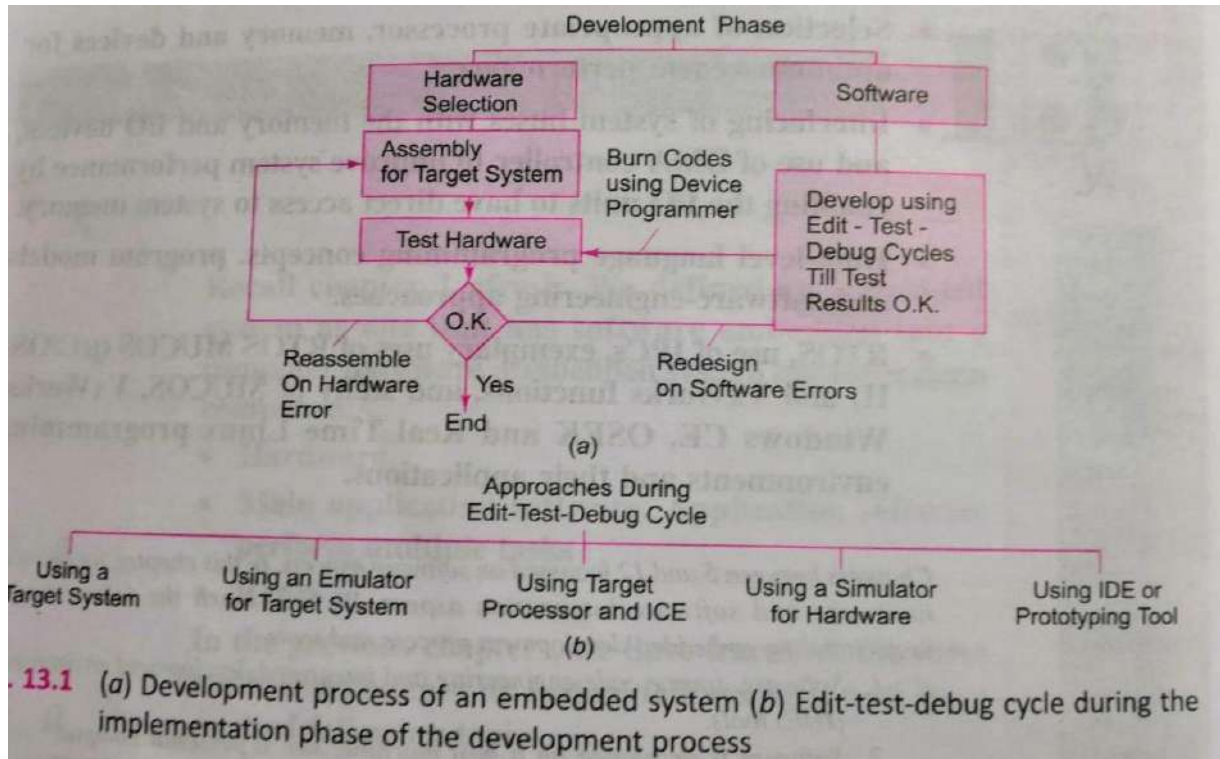
All I/O ports and devices have addresses. These are allocated to the devices according to the system processor and the system hardware configuration. Device addresses are used for processing by the driver. A device has an address, which is usually according to the system hardware or may also be the processor assigned ones. These addresses allocated to the following:

1. Device data Registers or RAM buffers
2. Device Control registers – it saves control bits and may save configuration bits also.
3. Device status registers – it saves flag bits as device status. A flag may include the need for servicing and show occurrence of a device-interrupt.

Porting issues of OS in an embedded platform

The following porting issues may arise when the OS is used in an embedded platform: I/O instructions, Interrupt servicing routines, data types, interface specific data types, byte order, data alignment, linked lists, memory page size, time intervals.

Design Cycle in the Development Phase for an Embedded System



Unlike the design of a software application on a standard platform, the design of an embedded system implies that both software and hardware are being designed in parallel. Although this isn't always the case, it is a reality for many designs today. The profound implications of this simultaneous design process heavily influence how systems are designed.

Figure shows the development process or an embedded system and Figure edit-test-debug cycle during implementation phase of the development process. There are cycles of editing- testing-debugging during the development phase, Whereas the processor part once chosen remains fixed, the application software codes have to be perfected by a number of runs and tests. Whereas the cost of the processor is quite small. The cost of developing a final targeted system is quite high and needs a larger time frame Man the hardware circuit design.

The developer uses four main approaches to the edit-test-debug cycles.

- I. An IDE or prototype tool
2. A simulator without any hardware
3. Processor only at the target system and uses an in-between ICE (in-circuit-emulator).
4. Target system at the lest stage.

UNIT 4

UNIT- IV Introduction to IoT

Introduction to Internet of Things: Characteristics of IoT, Design principles of IoT, IoT Architecture and Protocols, Enabling Technologies for IoT, IoT levels and IoT vs M2M. IoT Design Methodology: Design methodology, Challenges in IoT Design, IoT System Management, IoT Servers – Sensors.

Introduction to Internet of Things:

IoT comprises things that have unique identities and are connected to internet. By 2020 there will be a total of 50 billion devices /things connected to internet. IoT is not limited to just connecting things to the internet but also allow things to communicate and exchange data.

Definition:

A dynamic global n/w infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual —things have identities, physical attributes and virtual personalities and use intelligent interfaces, and are seamlessly integrated into information n/w, often communicate data associated with users and their environments.

Characteristics of IoT

1) Dynamic & Self Adapting: IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, users context or sensed environment.

Eg: the surveillance system is adapting itself based on context and changing conditions.

2) Self Configuring: allowing a large number of devices to work together to provide certain functionality.

3) Inter Operable Communication Protocols: support a number of interoperable communication protocols and can communicate with other devices and also with infrastructure.

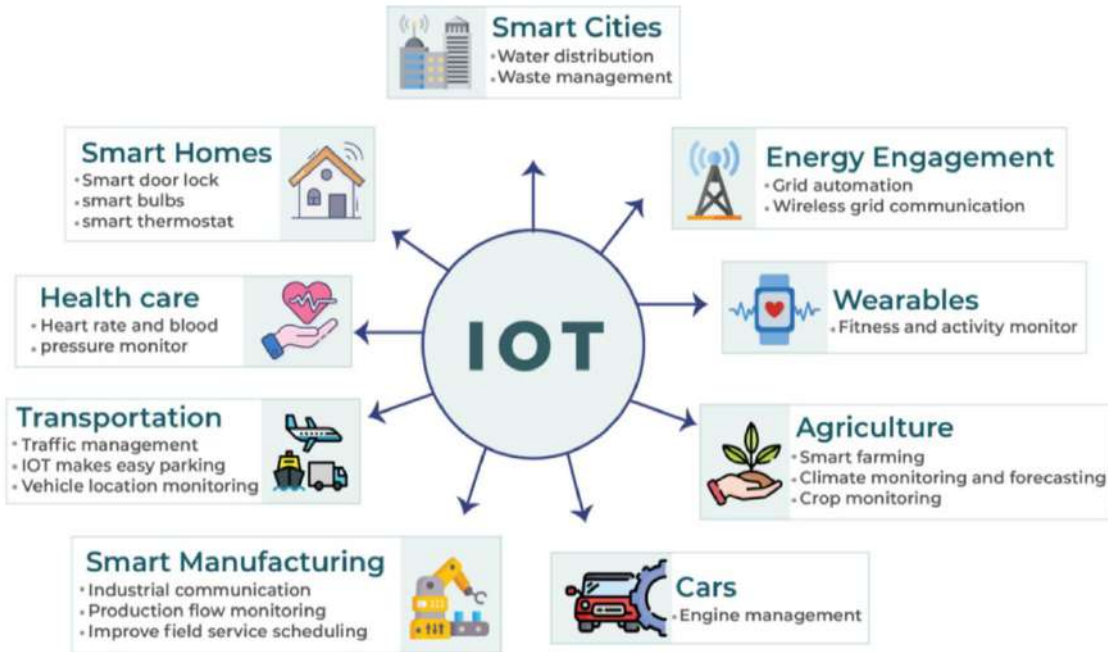
4) Unique Identity: Each IoT device has a unique identity and a unique identifier (IP address).

5) Integrated into Information Network: that allow them to communicate and exchange data with other devices and systems.

Applications of IoT:

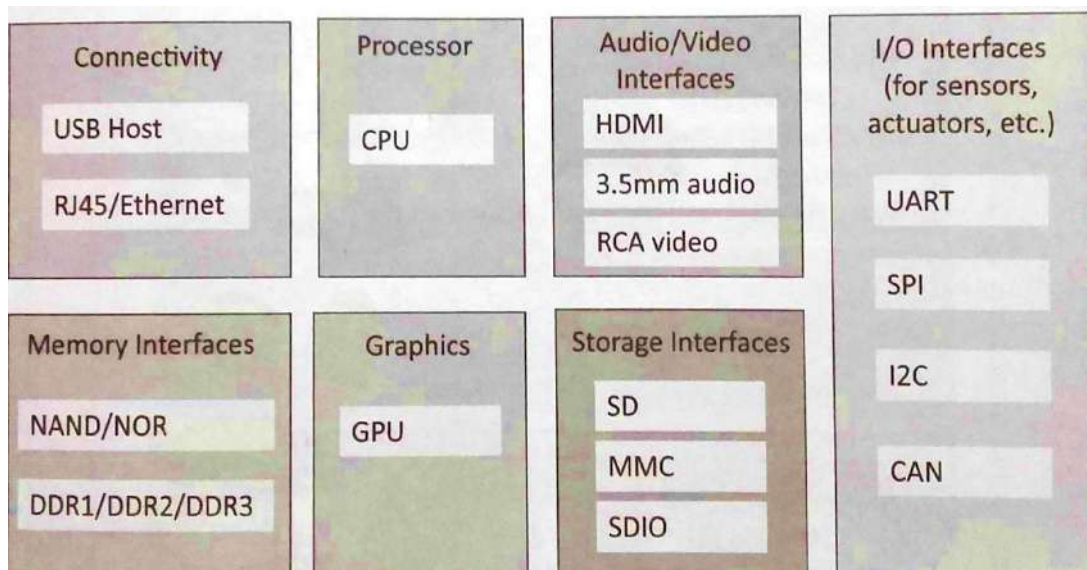
- 1) Home
- 2) Cities
- 3) Environment
- 4) Energy

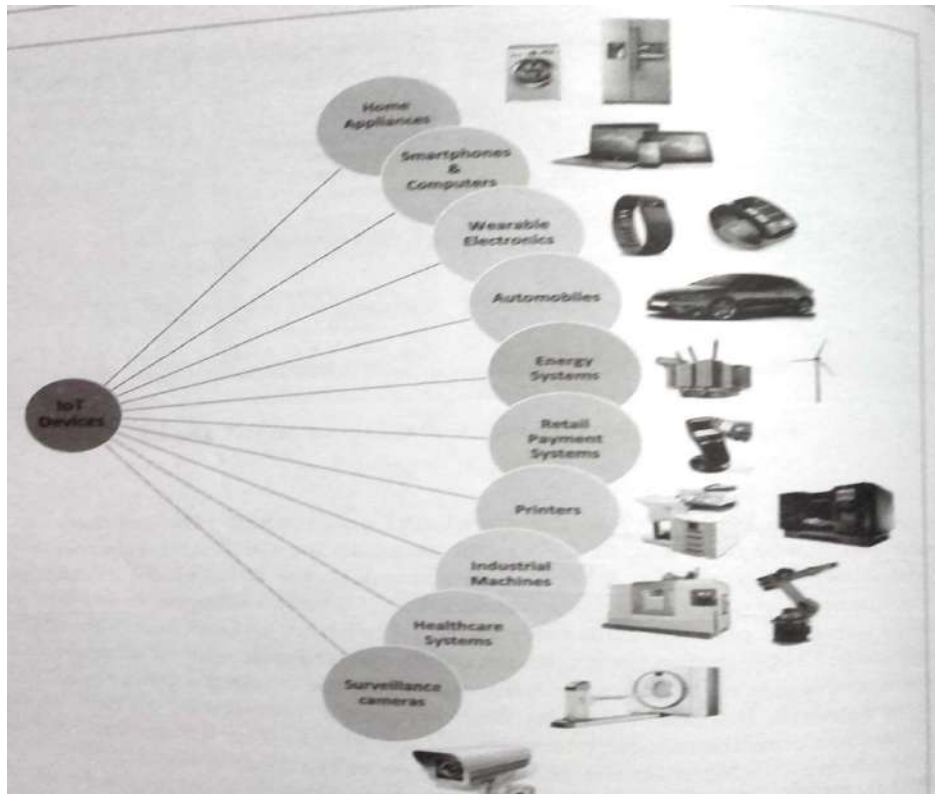
- 5) Retail
- 6) Logistics
- 7) Agriculture
- 8) Industry
- 9) Health & Life Style



Design principles of IoT:

Physical Design of IoT





The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.

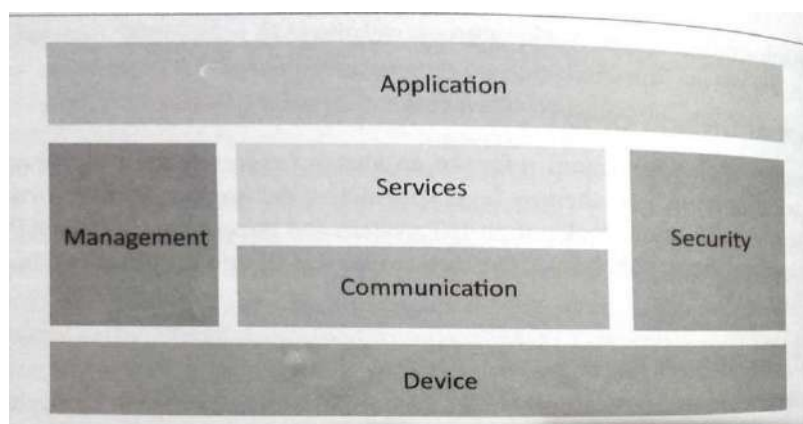
An IoT device may consist of several interfaces for communication to other devices both wired and wireless. These includes (i) I/O interfaces for sensors, (ii) Interfaces for internet connectivity (iii) memory and storage interfaces and (iv) audio/video interfaces.

LOGICAL DESIGN of IoT

Refers to an abstract represent of entities and processes without going into the low level specifics of implementation.

1) IoT Functional Blocks 2) IoT Communication Models 3) IoT Comm. APIs

1)IoT Functional Blocks:



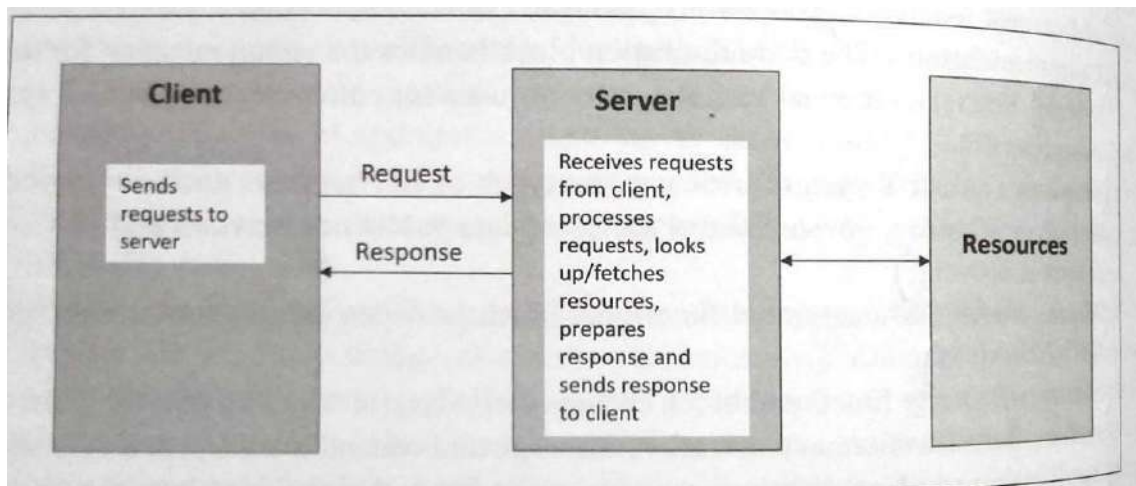
Provide the system the capabilities for identification, sensing, actuation, communication and management.

- Device:** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.
- Communication:** handles the communication for IoTsystem.
- Services:** for device monitoring, device control services, data publishing services and services for device discovery.
- Management:** Provides various functions to govern the IoT system.
- Security:** Secures IoT system and priority functions such as authentication ,authorization, message and context integrity and data security.
- Application:** IoT application provide an interface that the users can use to control and monitor various aspects of IoT system

2) IoT Communication Models:

1) Request-Response 2) Publish-Subscibe 3)Push-Pull4) ExclusivePair

1) Request-Response Model:



In which the client sends request to the server and the server replies to requests. Is a stateless communication model and each request-response pair is independent of others.

2) Publish-Subscibe Model:

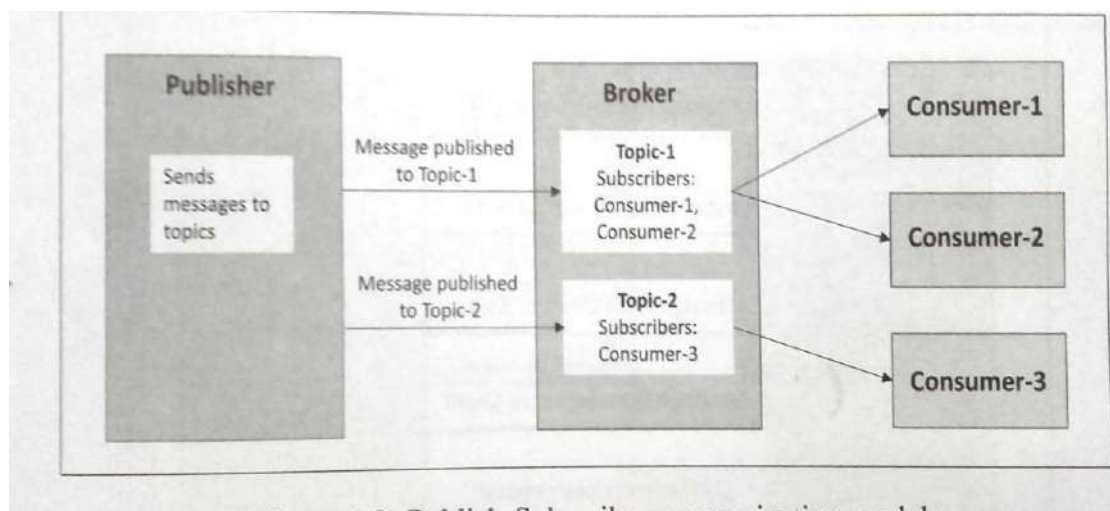
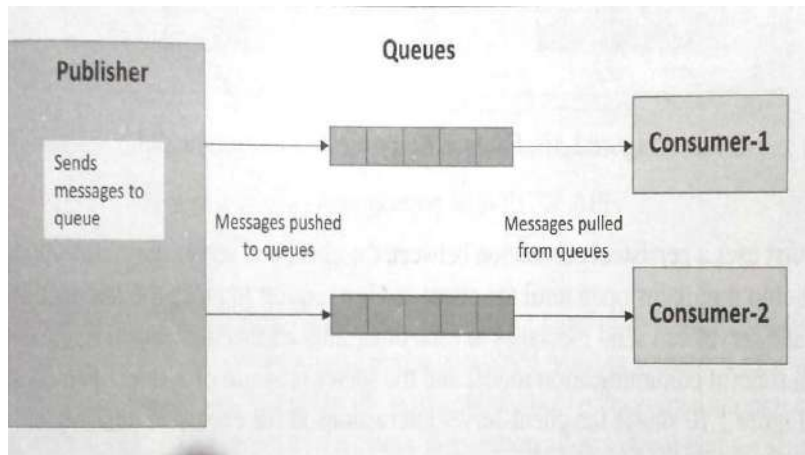


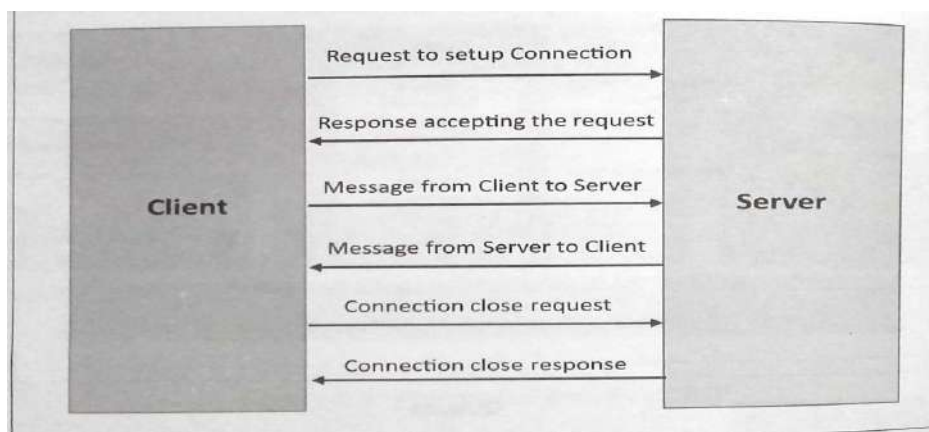
Figure 1.9: Publish-Subscribe communication model

Involves publishers, brokers and consumers. Publishers are source of data. Publishers send data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.

3) Push-Pull Model: in which data producers push data to queues and consumers pull data from the queues. Producers do not need to aware of the consumers. Queues help in decoupling the message between the producers and consumers.



4) Exclusive Pair: is bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once connection is set up it remains open until the client send a request to close the connection. Is a stateful communication model and server is aware of all the open connections.



3)IoT Communication APIs:

- a) REST based communication APIs(Request-Response Based Model)
- b) WebSocket based Communication APIs(Exclusive PairBased Model)

a) REST based communication APIs: Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a systems resources and have resource states are addressed and transferred. The REST architectural constraints.

b) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.

6 Principles of IoT design

1. Do your research

When designing IoT-enabled products, designers might make the mistake of forgetting why customers value these products in the first place. That's why it's a good idea to think about the value an IoT offering should deliver at the initial phase of your design.

When getting into IoT design, you're not building products anymore. You're building services and experiences that improve people's lives. That's why in-depth qualitative research is the key to figuring out how you can do that.

Assume the perspective of your customers to understand what they need and how your IoT implementation can solve their pain points. Research your target audience deeply to see what their existing experiences are and what they wish was different about them.

2. Concentrate on value

Early adopters are eager to try out new technologies. But the rest of your customer base might be reluctant to put a new solution to use. They may not feel confident with it and are likely to be cautious about using it.

If you want your IoT solution to become widely adopted, you need to focus on the actual tangible value it's going to deliver to your target audience.

What is the real end-user value of your solution? What might be the barriers to adopting new technology? How can your solution address them specifically?

Note that the features the early tech adopters might find valuable might turn out to be completely uninteresting for the majority of users. That's why you need to carefully plan which features to include and in what order, always concentrating on the actual value they provide.

3. Don't forget about the bigger picture

One characteristic trait of IoT solutions is that they typically include multiple devices that come with different capabilities and consist of both digital and physical touchpoints. Your solution might also be delivered to users in cooperation with service providers.

That's why it's not enough to design a single touchpoint well. Instead, you need to take the bigger picture into account and treat your IoT system holistically.

Delineate the role of every device and service. Develop a conceptual model of how users will perceive and understand the system. All the parts of your system need to work seamlessly together. Only then you'll be able to create a meaningful experience for your end-users.

4. Remember about the security

Don't forget that IoT solutions aren't purely digital. They're located in the real-world context, and the consequences of their actions might be serious if something goes wrong. At the same time, building trust in IoT solutions should be one of your main design drivers.

Make sure that every interaction with your product builds consumer trust rather than breaking it. In practice, it means that you should understand all the possible error situations that may be related to the context of its use. Then try to design your product in a way to prevent them. If error situations occur, make sure that the user is informed appropriately and provided with help.

Also, consider data security and privacy as a key aspect of your implementation. Users need to feel that their data is safe, and objects located in their workspaces or home can't be hacked.

That's why quality assurance and testing the system in the real-world context are so important.

5. Build with the context in mind

And speaking of context, it pays to remember that IoT solutions are located at the intersection of the physical and digital world. The commands you give through digital interfaces produce real-world effects. Unlike digital commands, these actions may not be easily undone.

In a real-world context, many unexpected things may happen. That's why you need to make sure that the design of your solution enables users to feel safe and in control at all times.

The context itself is a crucial consideration during IoT design. Depending on the physical context of your solution, you might have different goals in mind. For example, you might want to minimize user distraction or design devices that will be resistant to the changing weather conditions.

The social context is an important factor, as well. Don't forget that the devices you design for workspaces or homes will be used by multiple users.

6. Make good use of prototypes

IoT solutions are often difficult to upgrade. Once the user places the connected object somewhere, it might be hard to replace it with a new version – especially if the user would have to pay for the upgrade.

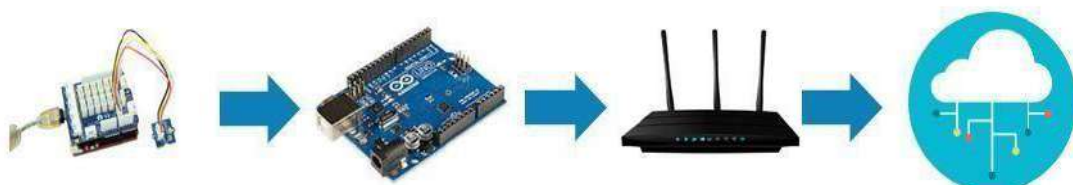
Even the software within the object might be hard to update because of security and privacy reasons. Make sure that your design practices help to avoid costly hardware iterations. Get your solution right from the start. From the design perspective, it means that prototyping and rapid iteration will become critical in the early stages of the project.

IoT Architecture and Protocols

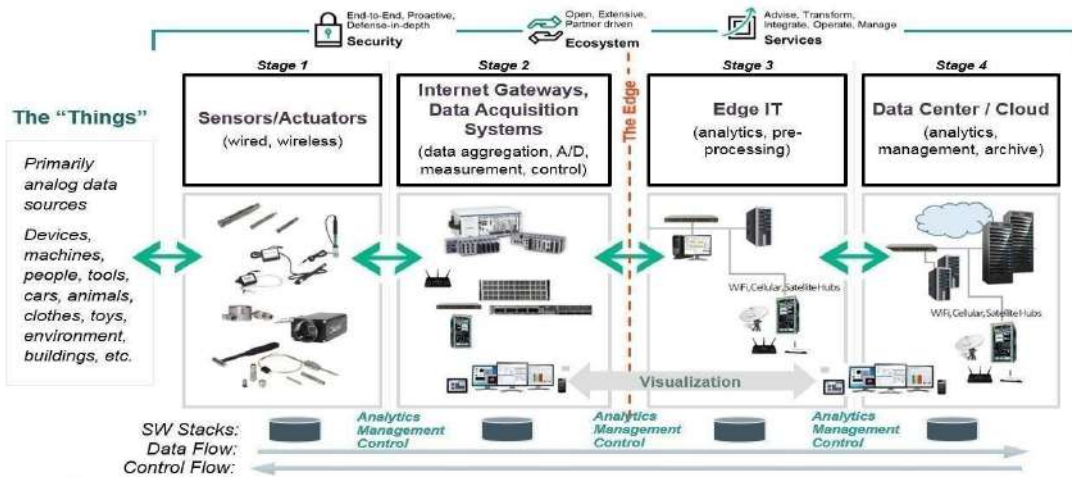
An IoT architecture is the system of numerous elements that range from sensors, protocols, actuators, to cloud services, and layers. Besides, devices and sensors the Internet of Things (IoT) architecture layers are distinguished to track the consistency of a system through protocols and gateways. Different architectures have been proposed by researchers and we can all agree that there is no single consensus on architecture for IoT. The most basic architecture is a three-layer architecture.

IoT architecture varies from solution to solution, based on the type of solution which we intend to build. IoT as a technology majorly consists of four main components, over which an architecture is framed.

- 1) Sensors
- 2) Devices
- 3) Gateway
- 4) Cloud



The 4 Stage IoT Solutions Architecture



Stage 1:- Sensors/actuators

- Sensors collect data from the environment or object under measurement and turn it into useful data. Think of the specialized structures in your cell phone that detect the directional pull of gravity and the phone's relative position to the —thing‖ we call the earth and convert it into data that your phone can use to orient the device.
- Actuators can also intervene to change the physical conditions that generate the data. An actuator might, for example, shut off a power supply, adjust an air flow valve, or move a robotic gripper in an assembly process.
- The sensing/actuating stage covers everything from legacy industrial devices to robotic camera systems, water level detectors, air quality sensors, accelerometers, and heart rate monitors. And the scope of the IoT is expanding rapidly, thanks in part to low-power wireless sensor network technologies and Power over Ethernet, which enable devices on a wired LAN to operate without the need for an A/C power source.

Stage 2:- The Internet gateway

- The data from the sensors starts in analog form. That data needs to be aggregated and converted into digital streams for further processing downstream. Data acquisition systems (DAS) perform these data aggregation and conversion functions. The DAS connects to the sensor network, aggregates outputs, and performs the analog-to-digital conversion. The Internet gateway receives the aggregated and digitized data and routes it over Wi-Fi, wired LANs, or the Internet, to Stage 3 systems for further processing. Stage 2 systems often sit in close proximity to the sensors and actuators.
- For example, a pump might contain a half-dozen sensors and actuators that feed data into a data aggregation device that also digitizes the data. This device might be physically attached to the pump. An adjacent gateway device or server would then process the data and forward it to the Stage 3 or Stage 4 systems. Intelligent gateways can build on additional, basic gateway functionality by adding such capabilities as analytics, malware protection, and data management services. These systems enable the analysis of data streams in real time.

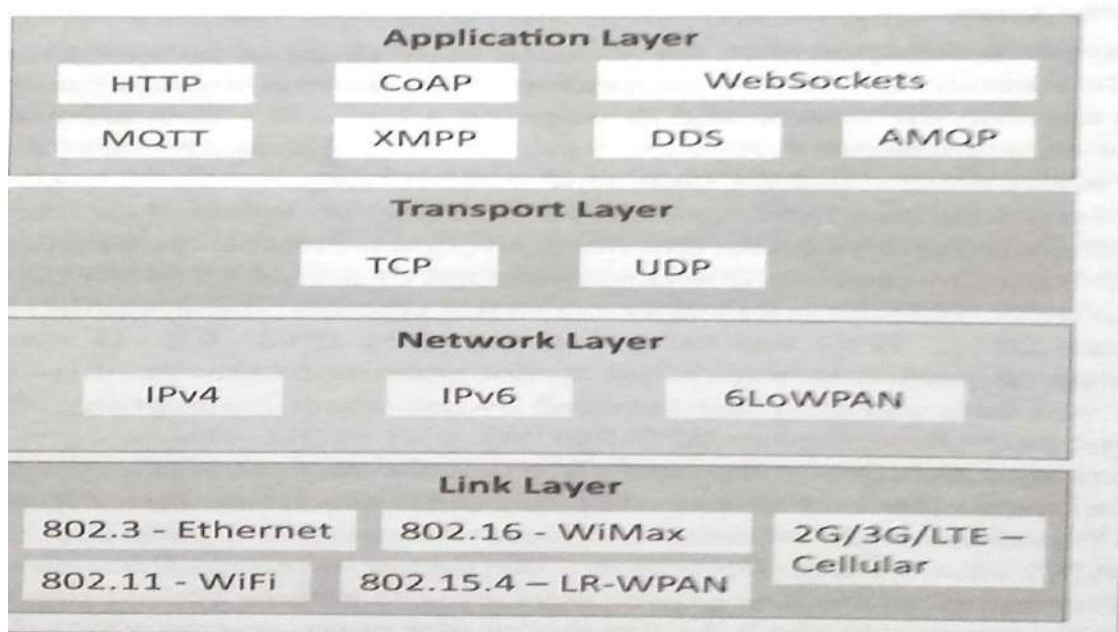
Stage 3:- Edge IT

- Once IoT data has been digitized and aggregated, it's ready to cross into the realm of IT. However, the data may require further processing before it enters the data center. This is where edge IT systems, which perform more analysis, come into play.
- Edge IT processing systems may be located in remote offices or other edge locations, but generally these sit in the facility or location where the sensors reside closer to the sensors, such as in a wiring closet.
- Because IoT data can easily eat up network bandwidth and swamp your data center resources, it's best to have systems at the edge capable of performing analytics as a way to lessen the burden on core IT infrastructure. You'd also face security concerns, storage issues, and delays processing the data. With a staged approach, you can preprocess the data, generate meaningful results, and pass only those on. For example, rather than passing on raw vibration data for the pumps, you could aggregate and convert the data, analyze it, and send only projections as to when each device will fail or need service.

Stage 4:- The data center and cloud

- Data that needs more in-depth processing, and where feedback doesn't have to be immediate, gets forwarded to physical data center or cloud-based systems.
- where more powerful IT systems can analyze, manage, and securely store the data. It takes longer to get results when you wait until data reaches Stage 4, but you can execute a more in-depth analysis, as well as combine your sensor data with data from other sources for deeper insights.
- Stage 4 processing may take place on-premises, in the cloud, or in a hybrid cloud system, but the type of processing executed in this stage remains the same, regardless of the platform.

IoT Protocols:



a) Link Layer :

Protocols determine how data is physically sent over the network's physical layer or medium. Local network connect to which host is attached. Hosts on the same link exchange data packets over the link layer using link layer protocols. Link layer determines how packets are coded and signaled by the h/w device over the medium to which the host is attached

Protocols:

- 802.3-Ethernet: IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 802.3 uses co-axial cable; 802.3i uses copper twisted pair connection; 802.3j uses fiber optic connection; 802.3ae uses Ethernet over fiber.
- 802.11-WiFi: IEEE802.11 is a collection of wireless LAN(WLAN) communication standards including extensive description of link layer. Eg: 802.11a operates in 5GHz band, 802.11b and 802.11g operates in 2.4GHz band, 802.11n operates in 2.4/5GHz band, 802.11ac operates in 5GHz band, 802.11ad operates in 60Ghzband.
- 802.16 - WiMax: IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMax provide data rates from 1.5 Mb/s to 1Gb/s.
- 802.15.4-LR-WPAN: IEEE802.15.4 is a collection of standards for low rate wireless personal area network(LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s.
- 2G/3G/4G-Mobile Communication: Data rates from 9.6kb/s(2G) to up to100Mb/s(4G).

B)Network/Internet Layer:

Responsible for sending IP datagrams from source n/w to destination n/w. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- IPv4: Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32 bit address. Allows total of 2^{32} addresses.
- IPv6: Internet Protocol version6 uses 128 bit address scheme and allows 2^{128} addresses.
- 6LOWPAN:(IPv6overLowpowerWirelessPersonalAreaNetwork)operates in 2.4 GHz frequency range and data transfer 250 kb/s.

C)Transport Layer:

Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control. Protocols:

- TCP: Transmission Control Protocol used by web browsers(along with HTTP and HTTPS), email(along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. Avoids n/w congestion and congestion collapse.

•UDP: User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery.

D)Application Layer:

Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.

Protocols:

•HTTP: Hyper Text Transfer Protocol that forms foundation of WWW. Follow request-response model Stateless protocol.

•CoAP: Constrained Application Protocol for machine-to-machine (M2M) applications with constrained devices, constrained environment and constrained n/w. Uses client- server architecture.

•WebSocket: allows full duplex communication over a single socket connection.

•MQTT: Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment.

•XMPP: Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.

•DDS: Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish-subscribe model.

•AMQP: Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model.

IoT Enabling Technologies

IoT is enabled by several technologies including Wireless Sensor Networks, Cloud Computing, Big Data Analytics, Embedded Systems, Security Protocols and architectures, Communication Protocols, Web Services, Mobile internet and semantic search engines.

1)Wireless Sensor Network(WSN):

Comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. Zig Bee is one of the most popular wireless technologies used by WSNs.

WSNs used in IoT systems are described as follows:

•Weather Monitoring System: in which nodes collect temp, humidity and other data, which is aggregated and analyzed.

•Indoor air quality monitoring systems: to collect data on the indoor air quality and concentration of various gases.

•Soil Moisture Monitoring Systems: to monitor soil moisture at various locations.

•Surveillance Systems: use WSNs for collecting surveillance data(motion data detection).

- Smart Grids : use WSNs for monitoring grids at various points.

- Structural Health Monitoring Systems: Use WSNs to monitor the health of structures(building, bridges) by collecting vibrations from sensor nodes deployed at various points in the structure.

2)Cloud Computing:

Services are offered to users in different forms.

- Infrastructure-as-a-service(IaaS):provides users the ability to provision computing and storage resources. These resources are provided to the users as a virtual machine instances and virtual storage.

- Platform-as-a-Service(PaaS): provides users the ability to develop and deploy application in cloud using the development tools, APIs, software libraries and services provided by the cloud service provider.

- Software-as-a-Service(SaaS): provides the user a complete software application or the user interface to the application itself.

3)Big Data Analytics:

Some examples of big data generated by IoT are

- Sensor data generated by IoT systems.

- Machine sensor data collected from sensors established in industrial and energy systems.

- Health and fitness data generated IoT devices.

- Data generated by IoT systems for location and tracking vehicles.

- Data generated by retail inventory monitoring systems.

4)Communication Protocols:

form the back-bone of IoT systems and enable network connectivity and coupling to applications.

- Allow devices to exchange data over network.

- Define the exchange formats, data encoding addressing schemes for device and routing of packets from source to destination.

- It includes sequence control, flow control and retransmission of lost packets.

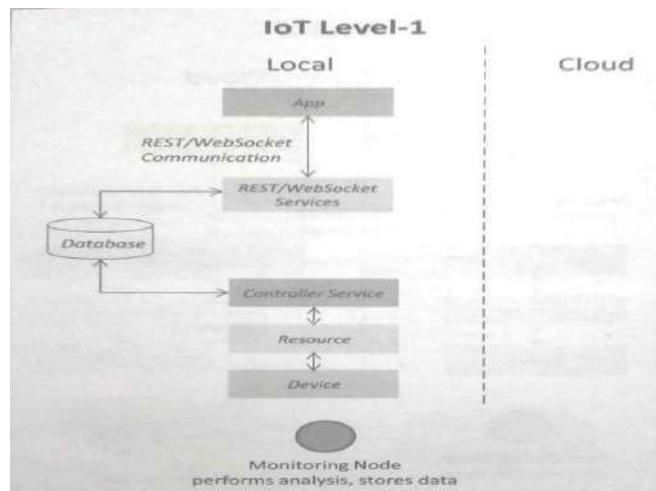
5)Embedded Systems:

is a computer system that has computer hardware and software embedded to perform specific tasks. Embedded System range from low cost miniaturized devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

IoT Levels and Deployment Templates

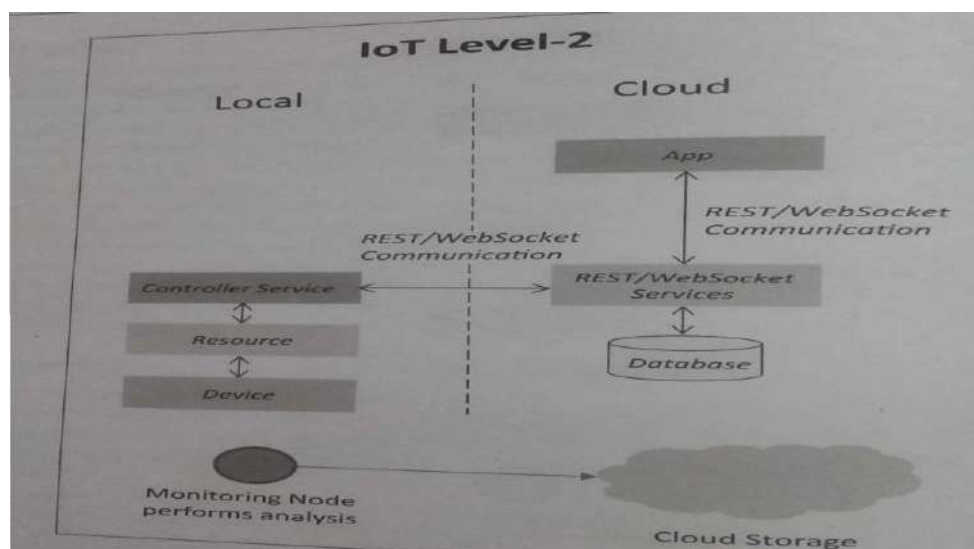
1)IoT Level1:

System has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application as shown in fig. Suitable for modeling low cost and low complexity solutions where the data involved is not big and analysis requirement are not computationally intensive. An e.g., of IoT Level1 is Home automation.



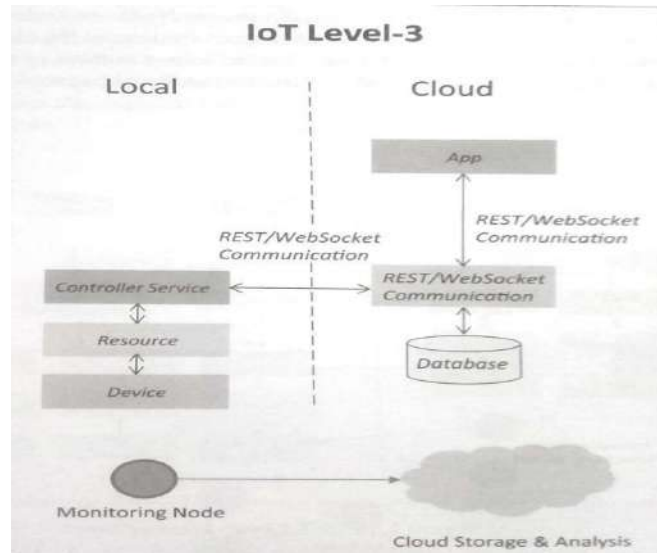
2)IoT Level2:

has a single node that performs sensing and/or actuating and local analysis as shown in fig. Data is stored in cloud and application is usually cloud based. Level2 IoT systems are suitable for solutions where data are involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself. An e.g., of Level2 IoT system for Smart Irrigation.



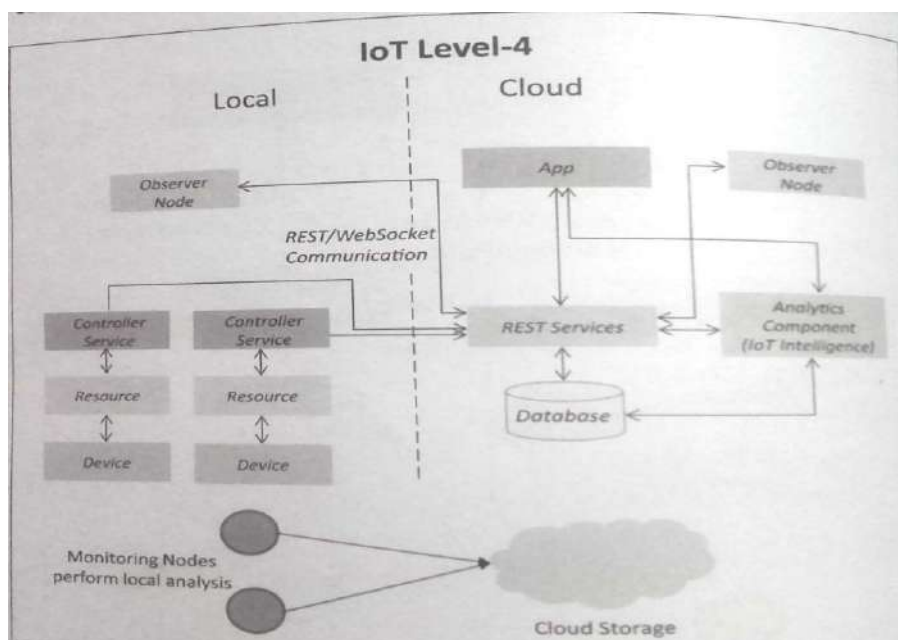
3)IoT Level3:

system has a single node. Data is stored and analyzed in the cloud application is cloud based as shown in fig. Level3 IoT systems are suitable for solutions where the data involved is big and analysis requirements are computationally intensive. An example of IoT level3 system for tracking package handling.



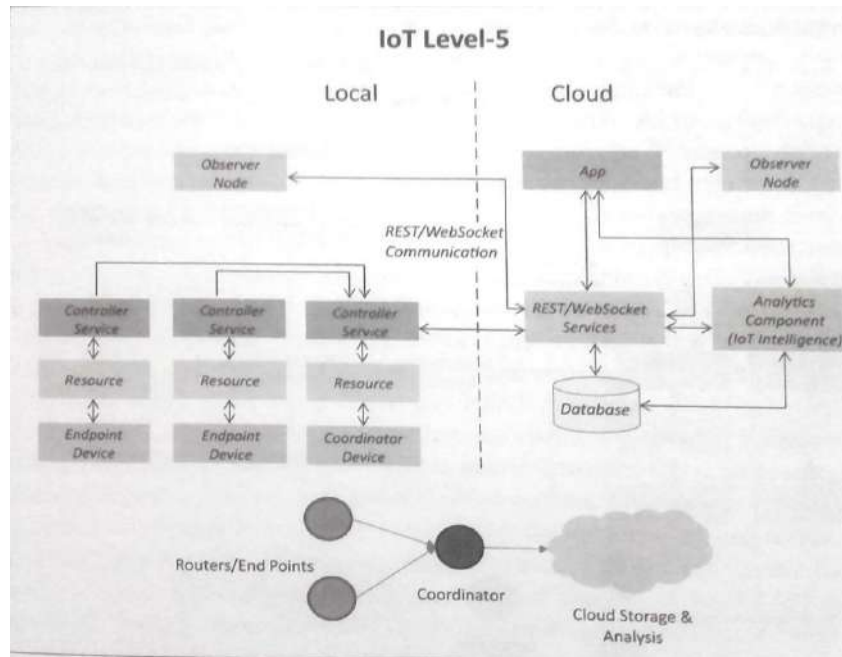
4)IoT Level4:

System has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud based as shown in fig. Level4 contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. An example of a Level4 IoT system for Noise Monitoring.



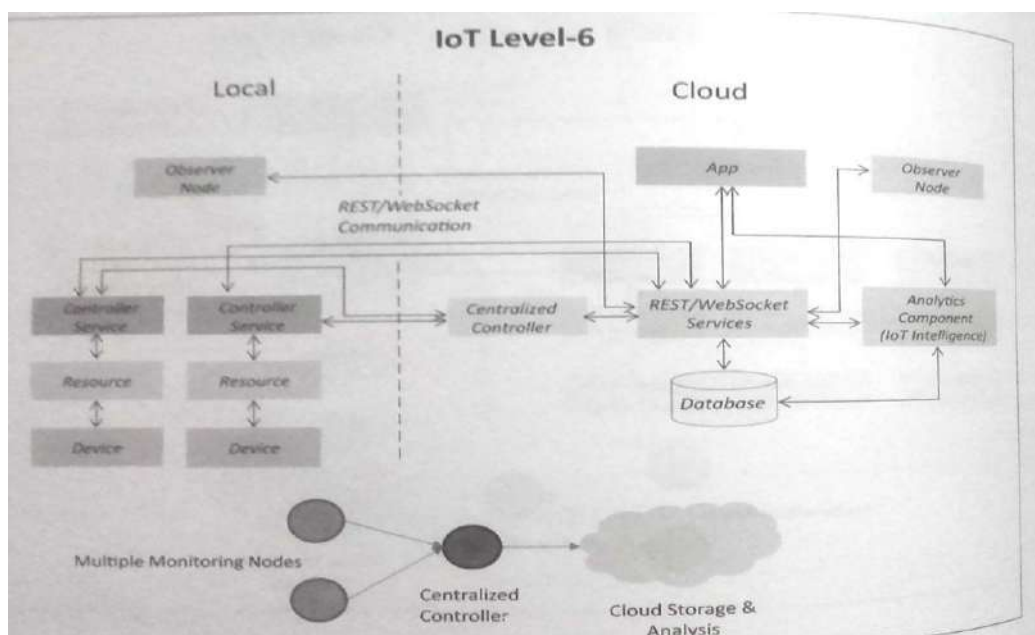
5)IoT Level5:

System has multiple end nodes and one coordinator node as shown in fig. The end nodes that perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud. Data is stored and analyzed in the cloud and application is cloud based. Level5 IoT systems are suitable for solution based on wireless sensor network, in which data involved is big and analysis requirements are computationally intensive. An example of Level5 system for Forest Fire Detection.



6)IoT Level6:

System has multiple independent end nodes that perform sensing and/or actuation and sensed data to the cloud. Data is stored in the cloud and application is cloud based as shown in fig. The analytics component analyses the data and stores the result in the cloud data base. The results are visualized with cloud based application. The centralized controller is aware of the status of all the end nodes and sends control commands to nodes. An example of a Level6 IoT system for Weather Monitoring System.



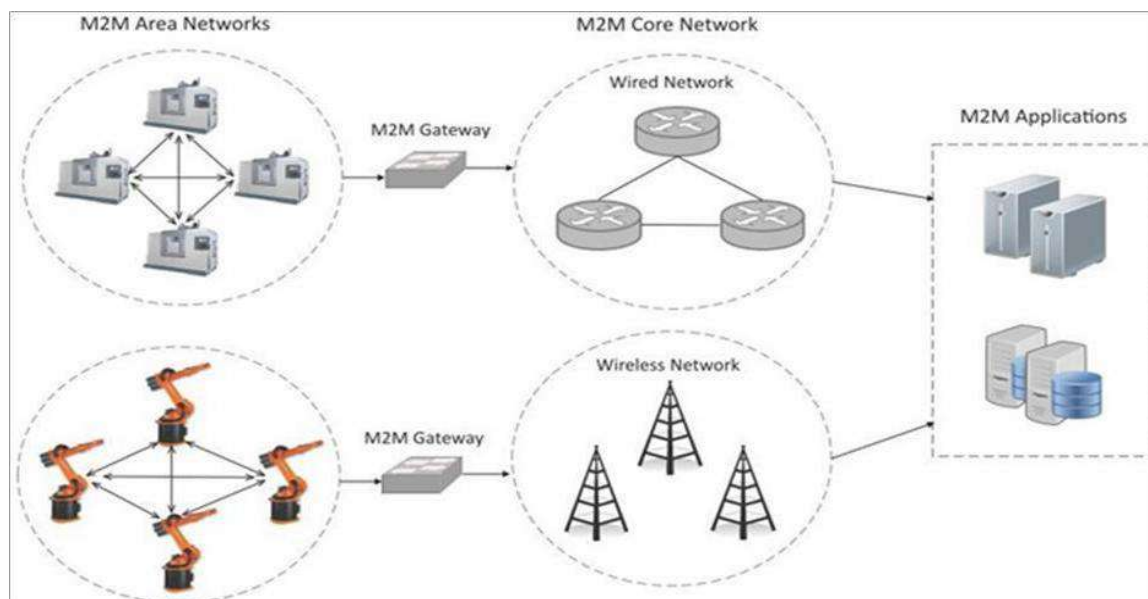
IoT and M2M

M2M:

Machine-to-Machine (M2M) refers to networking of machines(or devices) for the purpose of remote monitoring and control and data exchange.

- Term which is often synonymous with IoT is Machine-to-Machine (M2M).
- IoT and M2M are often used interchangeably.

Fig. Shows the end-to-end architecture of M2M systems comprises of M2M area networks, communication networks and application domain.



- An M2M area network comprises of machines(or M2M nodes) which have embedded network modules for sensing, actuation and communicating various communication protocols can be used for M2M LAN such as ZigBee, Bluetooth, M-bus, Wireless M-Bus etc., These protocols provide connectivity between M2M nodes within an M2M area network.

- The communication network provides connectivity to remote M2M area networks. The communication network provides connectivity to remote M2M area network. The communication network can use either wired or wireless network(IP based). While the M2M are networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based network. Since non-IP based protocols are used within M2M area network, the M2M nodes within one network cannot communicate with nodes in an external network.

- To enable the communication between remote M2M are network, M2M gateways are used

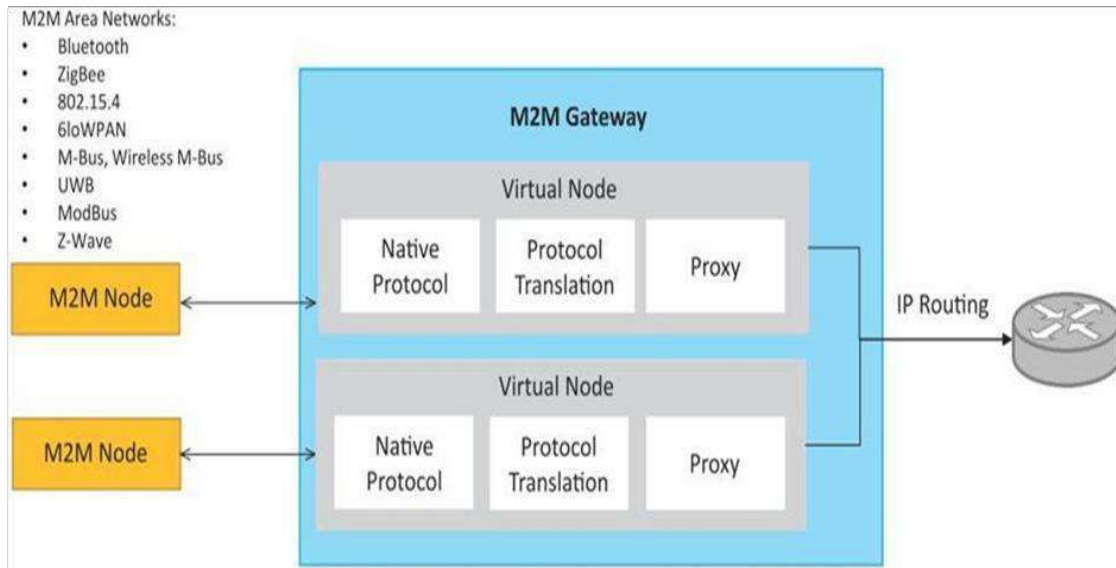


Fig. Shows a block diagram of an M2M gateway. The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable Ip-connectivity for M2M are networks. M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol(IP). With an M2M gateway, each mode in an M2M area network appears as a virtualized node for external M2M area networks.

Differences between IoT and M2M

1)Communication Protocols:

- Commonly uses M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus tec.,
- In IoT uses HTTP, CoAP, WebSocket , MQTT ,XMPP ,DDS ,AMQP etc.,

2)Machines in M2M Vs Things in IoT:

- Machines in M2M will be homogenous whereas Things in IoT will be heterogeneous.

3)Hardware Vs Software Emphasis:

- the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.

4)Data Collection &Analysis

- M2M data is collected in point solutions and often in on-premises storage infrastructure.
- The data in IoT is collected in the cloud (can be public, private or hybrid cloud).

5)Applications

- M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premises enterprise applications.
- IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

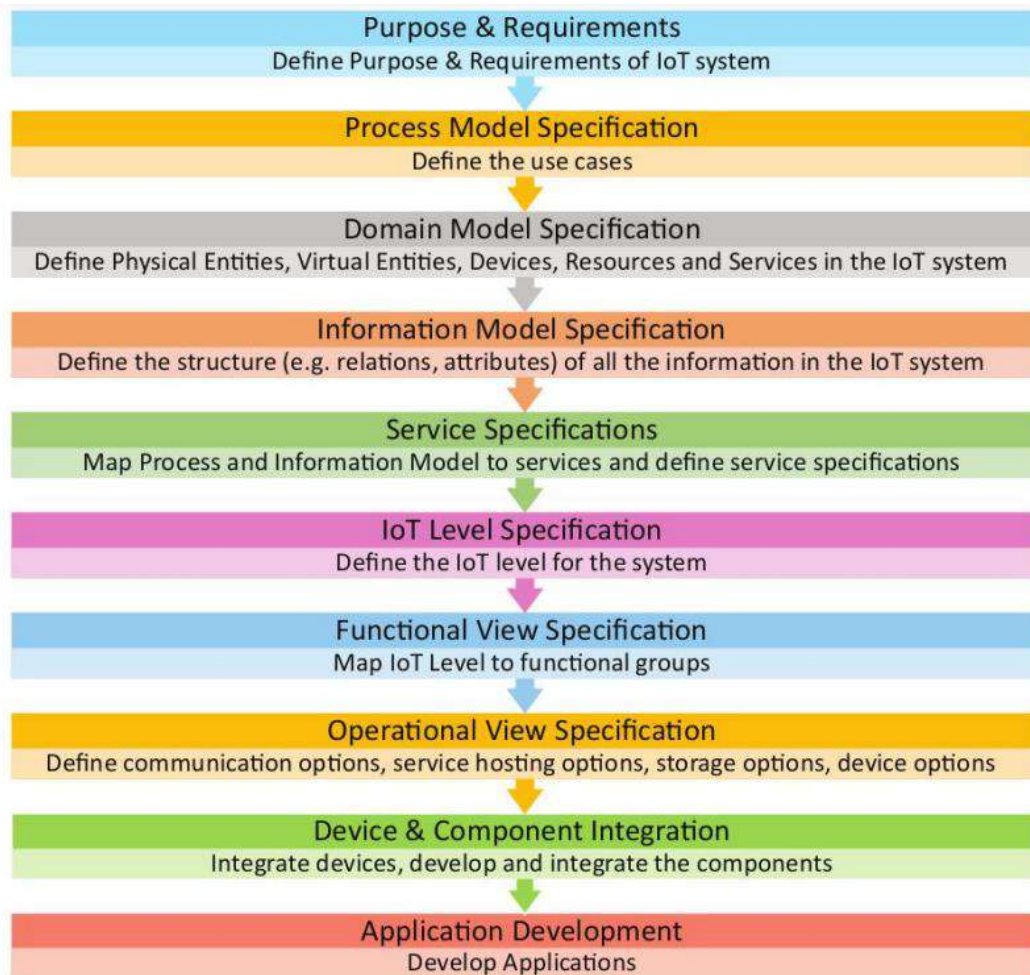
PARAMETERS	M2M	IOT
Abbreviation for	Machine to Machine	Internet of Things
Philosophy	M2M is Concept where two or more machines can communicate with each other and carry out certain functions without human intervention. Some degree of intelligence can be observed in M2M model.	IOT is an ecosystem of connected devices (via Internet) where the devices have ability to collect and transfer data over a network automatically without human intervention. IOT helps objects to interact with internal and/or external environment which in turn control the decision making.
Connection Type	Point to Point	Through IP Network using various Communication types
Communication protocols	Old proprietary protocols and communication techniques	Internet protocols used commonly
Value Chain	Linear	Multi-sided
Focus Area	For monitoring and control of 1 or few infrastructure/assets.	To address everyday needs of humans.
Sharing of collected data	Data collected is not shared with other applications	Data is shared with other applications (like weather forecasts, social media etc.) improve end user experience
Device dependency	Devices usually don't rely over Internet connection	Devices usually rely over Internet connection
Device in scope	Limited devices in scope	Large number of device sin scope
Scalability	Less scalable than IOT	More scalable due to cloud based architecture
Example	Remote monitoring, fleet control	Smart Cities, smart agriculture etc.
Business Type	B2B	B2B and B2C
Technology Integration	Vertical	Vertical and Horizontal
Open APIs	Not supported	Supported
Related terms	Sensors , Data and Information	End users, devices, wearables, Cloud and Big Data

<https://ipwithease.com>

IoT Design Methodology:

Design methodology

- Designing IoT systems can be a complex and challenging task as these systems involve interactions between various components such as IoT devices and network resources, web services, analytics components, application and database servers.
- IoT system designers often tend to design IoT systems keeping specific products/services in mind.
- So that designs are tied to specific product/service choices made. But it make updating the system design to add new features or replacing a particular product/service choice for a component becomes very complex, and in many cases may require complete redesign of the system



Step 1: Purpose & Requirements Specification

- The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements, ...) are captured.

Step 2: Process Specification

- The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.

Step 3: Domain Model Specification

- The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

Step 4: Information Model Specification

- The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the

information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations.

Step 5: Service Specifications

- The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.

Step 6: IoT Level Specification

- The sixth step in the IoT design methodology is to define the IoT level for the system.

Step 7: Functional View Specification

- The seventh step in the IoT design methodology is to define the Functional View. The Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.

Step 8: Operational View Specification

- The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc

Step 9: Device & Component Integration • The ninth step in the IoT design methodology is the integration of the devices and components.

Step 10: Application Development • The final step in the IoT design methodology is to develop the IoT application.

Design Challenges

In IoT, several automated devices that are connected; communicate with each other through the Internet, and in a small network and a couple of devices, connectivity is seamless. But when IoT is deployed globally, and the number of devices and sensors connect and communicate, connectivity issues arise. And also, the Internet is not just a network; it includes a heterogeneous network having cell towers, slow connectivity, fast connectivity, proxy servers, firewalls, and different companies with different standards and technologies, all things that can disrupt connectivity. So, here the design of the entire IoT system holds utmost importance and is treated as an essential component of IoT as the overall success of the procedure depends on great design. So, here, we will analyze factors which challenge the design of IoT technology.

The various design challenges in IoT are as follows.

1. Power and Battery Life
2. Security and Compliances
3. Tests and Certifications
4. Emerging Standards
5. Designing for Everyone

1.Power and Battery Life:

At the trim level of IoT setup may be energy efficient. Still, when in a high level of IoT setup, things become more complex in high-performance devices where processor, displays, and communication interfaces require varying amounts of power there, power usage management becomes difficult. So minimal battery drain and long battery life are needed, and it must achieve low power consumption and energy efficiency.

2.Security and Compliances:

It is essential in the IoT Connectivity—monitoring and Maintaining security across connectivity change. Many IoT/M2M devices also come with several internet connectivity options. Many security pitfalls allow hackers to access or take control of the device and create connectivity issues.

3.Tests and Certifications:

Testing and certification are significant parts of designing any product related to IoT. Boards with wireless components—or any radiating component—must undergo strict certification processes before they can be permitted to be sold in different parts of the world. At the same time, these certification rules may vary from continent to continent or even from country to country. Aside from being a tedious process, this requires high costs, which the system designer should include when planning budgets for their circuit designs. For any Bluetooth module or chip present onboard, several certifications are required—from the FCC (for the US), CE (for European countries), IC (for Canada) and the list goes on.

4.Emerging Standards:

Despite IoT being depicted as a connected ecosystem where devices work in harmony, the reality is different. As with any untested frontier, plenty of companies are racing to become the dominant players in the emerging space; even when some product lines are completely walled off and are often designed to work exclusively with trusted providers, while, other systems are completely open where the biggest challenges for developers are usually coping with the potential interference between various equipment.

To help overcome these types of challenges, the "Open Connectivity Foundation" is currently developing an open standard to overcome previously mentioned issues of devices being produced independently of one another.

The biggest takeaway from the draft specs is that the full operability of the system needs to be engineered at all layers of the development stack i.e. vertical services, platform, and connectivity – to ensure a seamless user experience. The bulk of the OCF set of standards

leverages abstraction to streamline development workflows while guaranteeing that the data protocols are all dynamic and have layer agnostic capabilities. Therefore, here the five methods of the above-discussed standards include:

creating,
retrieving,
updating,
deleting, and
notifying.

There's also the IEEE standard which has an extensive and exclusive line of standards for the Internet of things

5.Designing for Everyone:

Perhaps one of the biggest challenges for any IoT system development or requirement is accommodating all the users' needs and being genuinely successful; it can't only target connected devices to a tech-savvy audience. Smart homes, involve leveraging an entire ecosystem of devices. Locks, thermostats, lighting, alarms, and more – are the foundations of living at home with more comfort.

There is also a lot of machine-to-machine (M2M) projects and systems such as intelligent power grids, general building automation, vehicle-to-vehicle communication, and wearable communication devices. It seems overwhelming, right? It doesn't have to be.

In the past, visuals were mostly the cornerstone of good successful user experience platforms; however, the future is now all about conversational UIs (UI that interact, know and serve the needs of the user). This opens an entirely brand new can of worms as user experience professionals now need to handle both linguistics and general visual design

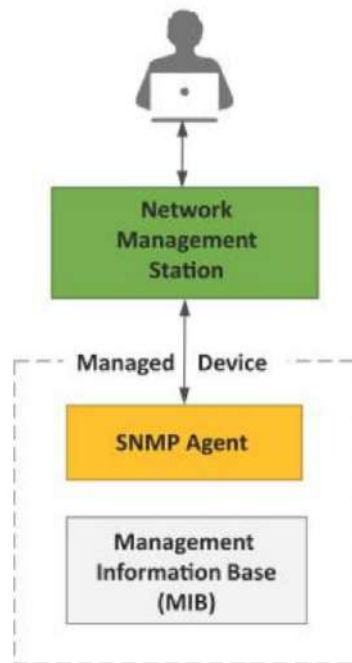
IoT System Management

Need for IoT Systems Management

- Automating Configuration
- Monitoring Operational & Statistical Data
- Improved Reliability
- System Wide Configurations
- Multiple System Configurations
- Retrieving & Reusing Configurations

Simple Network Management Protocol (SNMP)

- SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc. •
- SNMP component include
- Network Management Station (NMS)
- Managed Device
- Management Information Base (MIB)
- SNMP Agent that runs on the device



Limitations of SNMP

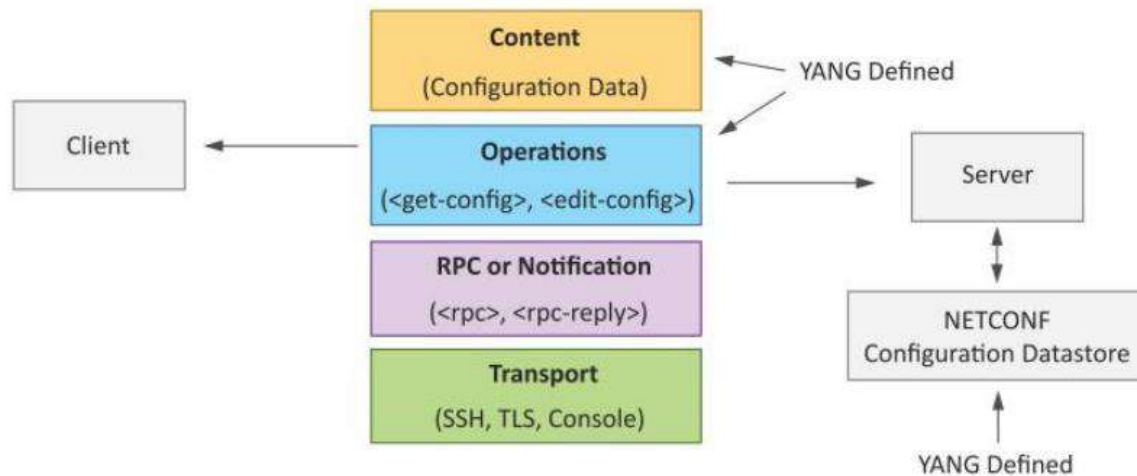
- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features.

Network Operator Requirements

- Ease of use
- Distinction between configuration and state data
- Fetch configuration and state data separately
- Configuration of the network as a whole
- Configuration transactions across devices
- Configuration deltas
- Dump and restore configurations
- Configuration database schemas
- Comparing configurations
- Role-based access control
- Consistency of access control lists:
- Multiple configuration sets

NETCONF

- Network Configuration Protocol (NETCONF) is a session-based network management protocol. NETCONF allows retrieving state or configuration data and manipulating configuration data on network devices



- NETCONF works on SSH transport protocol.
- Transport layer provides end-to-end connectivity and ensure reliable delivery of messages.
- NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages.
- The RPC layer provides mechanism for encoding of RPC calls and notifications.
- NETCONF provides various operations to retrieve and edit configuration data from network devices.
- The Content Layer consists of configuration and state data which is XML-encoded.
- The schema of the configuration and state data is defined in a data modeling language called YANG.
- NETCONF provides a clear separation of the configuration and state data.
- The configuration data resides within a NETCONF configuration datastore on the server

YANG

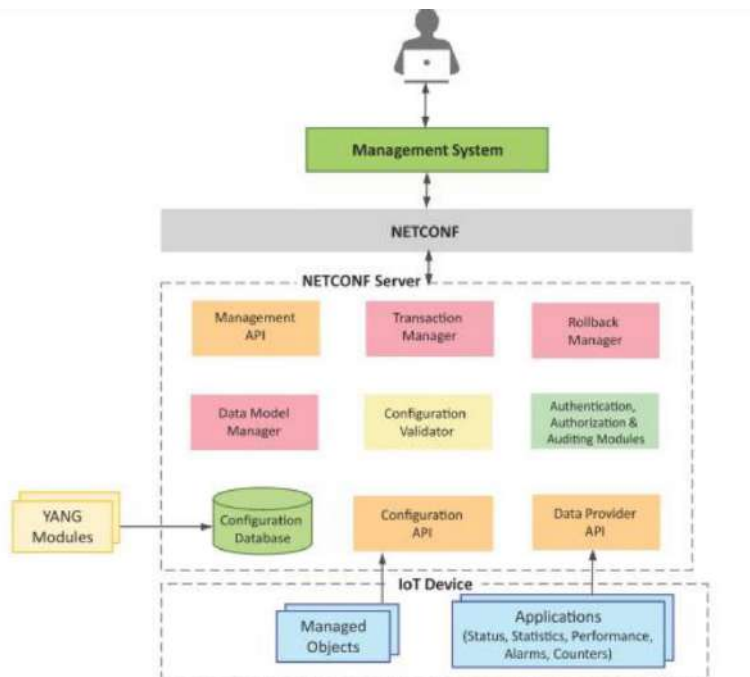
- YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol
- YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.
- YANG modules defines the data exchanged between the NETCONF client and server.
- A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure.
- The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs.
- Leaf nodes are organized using 'container' or 'list' constructs.
- A YANG module can import definitions from other modules.
- Constraints can be defined on the data nodes, e.g. allowed values.
- YANG can model both configuration data and state data using the 'config' statement.

IoT Systems Management with NETCONF-YANG

Management System

- Management API
- Transaction Manager
- Rollback Manager
- Data Model Manager

- Configuration Validator
- Configuration Database
- Configuration API
- Data Provider API



IoT Servers – Sensors.

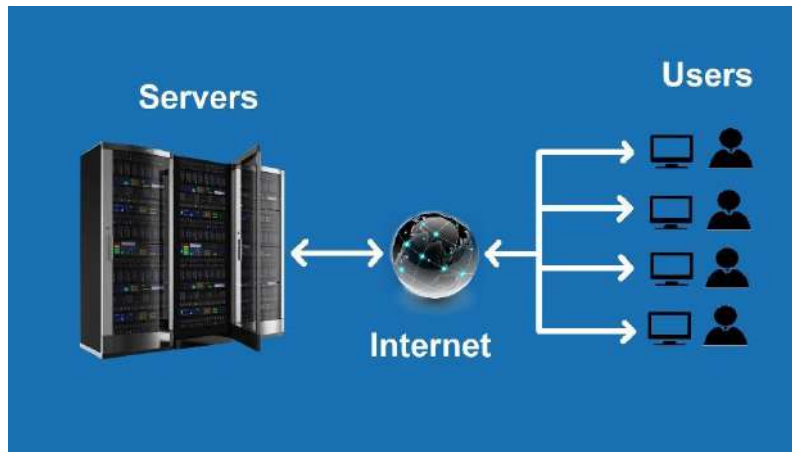
Server

A server is a computer, a device or a program that is dedicated to managing network resources. They are called that because they “serve” another computer, device, or program called “client” to which they provide functionality.

There are a number of categories of servers, including print servers, file servers, network servers and database servers. In theory, whenever computers share resources with client machines they are considered servers.

However, servers are often referred to as dedicated because they carry out hardly any other tasks apart from their server tasks.

Most frequently client–server systems are implemented by the request–response model., i.e., a client sends a request to the server. In this model server performs some action and sends a response back to the client, typically with a result or acknowledgement. Designating a computer as server-class hardware means that it is specialized for running servers on it. This implies that it is more powerful and reliable than standard personal computers. But large computing clusters may be composed of many relatively simple, replaceable server components.



Types of Servers and their applications:

1.Application server –

These servers hosts web apps (computer programs that run inside a web browser) allowing users in the network to run and use them preventing the installation a copy on their own computers. These servers need not be part of the World Wide Web. Their clients are computers with a web browser.

2.Catalog server –

These servers maintains an index or table of contents of information that can be found across a large distributed network. Distributed network may include computers, users, files shared on file servers, and web apps. Examples of catalog servers are Directory servers and name servers. Their clients are any computer program that needs to find something on the network. Example can be a Domain member attempting to log in, an email client looking for an email address, or a user looking for a file

3.Communications server –

These servers maintains an environment needed for one communication endpoint to find other endpoints and then communicates with them. These servers may or may not include a directory of communication endpoints and a presence detection service, depending on the openness and security parameters of the network. Their clients are communication endpoints.

4.Computing server –

These servers share vast amounts of computing resources which include CPU and random-access memory over a network. Any computer program that needs more CPU power and RAM than a personal computer can probably afford can use these types of servers. The client must be a networked computer to implement the client–server model which is necessity.

5.Database server –

These servers maintains and shares any form of database over a network. A database is a organized collections of data with predefined properties that may be displayed in a table. Clients of these servers are spreadsheets, accounting software, asset management software or virtually any computer program that consumes well-organized data, especially in large volumes.

6.Fax server –

These servers share one or more fax machines over a network which eliminates the hassle of physical access. Any fax sender or recipient are the clients of these servers.

7.File server –

Shares files and folders, storage space to hold files and folders, or both, over a network. Networked computers are the intended clients, even though local programs can be clients.

8.Game server –

These servers enable several computers or gaming devices to play multiplayer games. Personal computers or gaming consoles are their clients.

9.Mail server –

These servers make email communication possible in the same way as a post office makes snail mail communication possible. Clients of these servers are senders and recipients of email.

10.Print server –

These servers share one or more printers over a network which eliminates the hassle of physical access. Their clients are computers in need of printing something.

11.Proxy server –

This server acts as an intermediary between a client and a server accepting incoming traffic from the client and sending it to the server. Reasons to use a proxy server include content control and filtering, improving traffic performance, preventing unauthorized network access or simply routing the traffic over a large and complex network. Their clients are any networked computer.

12.Web server –

These servers host web pages. A web server is responsible for making the World Wide Web possible. Each website has one or more web servers. Their clients are computers with a web browser.

SENSORS

Sensors are devices that provide an output signal based on measuring an environmental phenomenon such as measuring temperature, humidity, pressure, altitude, ambient light, distance etc.

These devices are used to give quantitative and qualitative measurements of an environmental factor for the purposes of monitoring data to either record or take action.

For example, a temperature sensor can be used to monitor the ambient temperature. Based on the temperature sensor's measurement and output, heating or cooling can be enabled to bring the ambient temperature of the room to the optimal temperature

SENSORS IN IOT

Sensors have been around for decades being used in many different applications. But recently, with the advent of IoT sensors, today must encompass the ability to process its data, communicate with other sensors and platforms, forming a crucial part in the entire IoT ecosystem

CLASSIFICATION OF SENSORS:

Due to the sheer number of sensors available, for simplification sensors are divided into 5 core classifications depending on how they work

1. Active and Passive Sensors
2. Contact and Non-contact Sensors
3. Absolute and relative sensors
4. Analog and Digital Sensors
5. Miscellaneous Sensors

1. ACTIVE AND PASSIVE SENSORS

Active Sensors are sensors that require a dedicated external power supply in order to function. Examples include GPS and ultrasonic sensors.

Passive sensors on the other hand do not require any external supply and can receive enough electrical signal from the environment to function. Examples include thermal sensors, NFC tags, etc

2. CONTACT AND NON-CONTACT SENSORS

Contact sensors are sensors that require physical contact with the environmental stimulus the sensor is measuring. Examples include touch sensors, temperature sensors, strain gauges, etc

Non-contact sensors are sensors that do not require direct contact with the environmental stimulus it measures. Examples include optical sensors, magnetic sensors, infrared thermometers, etc

3. ABSOLUTE AND RELATIVE SENSORS

Absolute sensors, as its name suggests, provide an absolute reading of the stimulus. For example, thermistors always give out the absolute temperature readings

Relative sensors provide measurements relative to something that is either fixed or variable. Thermocouple is an example of a relative sensor, where the temperature difference is measured as opposed to direct measurement

4. ANALOG AND DIGITAL SENSORS

Analog sensors produce a continuous output signal proportional to the measurement. Examples include thermometers, LDR, pressure sensors etc

Digital sensors are sensors that convert the measurement into a digital signal. Examples include Inertial Measurement Units, ultrasonic sensors, etc

5. MISCELLANEOUS SENSORS

There are many more types of sensors that may not necessarily fit into the above categories. Those sensors will be classified as miscellaneous sensors and these include biological, chemical, radioactive sensors, etc

TOP 8 IOT SENSORS

1. Temperature Sensors

Temperature sensors measure the amount of heat energy in a source, allowing them to detect temperature changes and convert these changes to data. Machinery used in manufacturing often requires environmental and device temperatures to be at specific levels. Similarly, within agriculture, soil temperature is a key factor for crop growth.

2. Humidity Sensors

These types of sensors measure the amount of water vapor in the atmosphere of air or other gases. Humidity sensors are commonly found in heating, vents and air conditioning (HVAC) systems in both industrial and residential domains. They can be found in many other areas including hospitals, and meteorology stations to report and predict weather.

3. Pressure Sensors

A pressure sensor senses changes in gases and liquids. When the pressure changes, the sensor detects these changes, and communicates them to connected systems. Common use cases include leak testing which can be a result of decay. Pressure sensors are also useful in the manufacturing of water systems as it is easy to detect fluctuations or drops in pressure.

4. Proximity Sensors

Proximity sensors are used for non-contact detection of objects near the sensor. These types of sensors often emit electromagnetic fields or beams of radiation such as infrared. Proximity sensors have some interesting use cases. In retail, a proximity sensor can detect the motion between a customer and a product in which he or she is interested. The user can be notified of any discounts or special offers of products located near the sensor. Proximity sensors are also used in the parking lots of malls, stadiums and airports to indicate parking availability. They can also be used on the assembly lines of chemical, food and many other types of industries.

5. Level Sensors

Level sensors are used to detect the level of substances including liquids, powders and granular materials. Many industries including oil manufacturing, water treatment and beverage and food manufacturing factories use level sensors. Waste management systems provide a common use case as level sensors can detect the level of waste in a garbage can or dumpster.

6. Accelerometers

Accelerometers detect an object's acceleration i.e. the rate of change of the object's velocity with respect to time. Accelerometers can also detect changes to gravity. Use cases for accelerometers include smart pedometers and monitoring driving fleets. They can also be used as anti-theft protection alerting the system if an object that should be stationary is moved.

7. Gyroscope

Gyroscope sensors measure the angular rate or velocity, often defined as a measurement of speed and rotation around an axis. Use cases include automotive, such as car navigation and electronic stability control (anti-skid) systems. Additional use cases include motion sensing for video games, and camera-shake detection systems.

8. Gas Sensors

These types of sensors monitor and detect changes in air quality, including the presence of toxic, combustible or hazardous gasses. Industries using gas sensors include mining, oil and gas, chemical research and manufacturing. A common consumer use case is the familiar carbon dioxide detectors used in many homes.

9. Infrared Sensors

These types of sensors sense characteristics in their surroundings by either emitting or detecting infrared radiation. They can also measure the heat emitted by objects. Infrared sensors are used in a variety of different IoT projects including healthcare as they simplify the monitoring of blood flow and blood pressure. Televisions use infrared sensors to interpret the signals sent from a remote control. Another interesting application is that of art historians using infrared sensors to see hidden layers in paintings to help determine whether a work of art is original or fake or has been altered by a restoration process.

10. Optical Sensors

Optical sensors convert rays of light into electrical signals. There are many applications and use cases for optical sensors. In the auto industry, vehicles use optical sensors to recognize signs, obstacles, and other things that a driver would notice when driving or parking. Optical sensors play a big role in the development of driverless cars. Optical sensors are very common in smart phones. For example, ambient light sensors can extend battery life. Optical sensors are also used in the biomedical field including breath analysis and heart-rate monitors.

UNIT 5

UNIT- V Arduino in IoT

Basics of Arduino: Introduction to Arduino – Types of Arduino – Arduino Toolchain – Arduino Programming Structure – Sketches – Pins -Input/Output From Pins Using Sketches – Introduction to Arduino Shields – Integration of Sensors and Actuators with Arduino-Connecting LEDs with Arduino, Connecting LCD with Arduino – Tinkercad Arduino simulation.

Introduction to Arduino

Arduino is a software as well as hardware platform that helps in making electronic projects. It is an open source platform and has a variety of controllers and microprocessors. There are various types of Arduino boards used for various purposes.

The Arduino is a single circuit board, which consists of different interfaces or parts. The board consists of the set of digital and analog pins that are used to connect various devices and components, which we want to use for the functioning of the electronic devices. Most of the Arduino consists of 14 digital I/O pins. The analog pins in Arduino are mostly useful for fine-grained control. The pins in the Arduino board are arranged in a specific pattern. The other devices on the Arduino board are USB port, small components (voltage regulator or oscillator), microcontroller, power connector, etc.. Arduino also simplifies the process of working with microcontrollers.

Features of Arduino:

- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

History of Arduino:

The project began in the Interaction Design Institute in Ivrea, Italy. Under the supervision of Casey Reas and Massimo Banzi, the Hernando Bar in 2003 created the Wiring (a development platform). It was considered as the master thesis project at IDII. The Wiring platform includes the PCB (Printed Circuit Board). The PCB is operated with the ATmega168 Microcontroller. The ATmega168 Microcontroller was an IDE. It was based on the library and processing functions, which are used to easily program the microcontroller.

In 2005, Massimo Banzi, David Cuartielles, David Mellis, and another IDII student supported the ATmega168 to the Wiring platform. They further named the project as Arduino.

The project of Arduino was started in 2005 for students in Ivrea, Italy. It aimed to provide an easy and low-cost method for hobbyists and professionals to interact with the environment using the actuators and the sensors. The beginner devices were simple motion detectors, robots, and thermostats.

In mid-2011, the estimated production of Arduino commercially was 300,000. In 2013, the Arduino boards in use were about 700,000.

Around April 2017, Massimo Banzi introduced the foundation of Arduino as the "new beginning for Arduino". In July 2017, Musto continued to pull many Open Source licenses and the code from the websites of the Arduino. In October 2017, Arduino introduced its collaboration with the ARM Holdings. The Arduino continues to work with architectures and technology vendors.

Hardware and Software

Arduino boards are generally based on microcontrollers from Atmel Corporation like 8, 16 or 32 bit AVR architecture based microcontrollers. The important feature of the Arduino boards is the standard connectors. Using these connectors, we can connect the Arduino board to other devices like LEDs or add-on modules called Shields.

The Arduino boards also consists of on board voltage regulator and crystal oscillator. They also consist of USB to serial adapter using which the Arduino board can be programmed using USB connection.

In order to program the Arduino board, we need to use IDE provided by Arduino. The Arduino IDE is based on Processing programming language and supports C and C++.

Types of Arduino

Arduino makes several different boards, each with different capabilities. In addition, part of being open source hardware means that others can modify and produce derivatives of Arduino boards that provide even more form factors and functionality.

The Arduino boards are provided as open source that helps the user to build their projects and instruments according to their need. This electronic platform contains microcontrollers,

connections, LEDs and many more. There are various types of Arduino boards present in the market that includes Arduino UNO, Red Board, LilyPad Arduino, Arduino Mega, Arduino Leonardo. All these Arduino boards are different in specifications, features and uses and are used in different type of electronics project.

Entry-Level Arduino Boards

Arduino's entry-level category contains the microcontroller boards that most DIYers choose to use for their projects, as they offer straightforward features and come with heaps of documentation. This also means that they can lack the niche features that come with enhanced and IoT Arduino boards.

1.Arduino UNO



The development of Arduino UNO board is considered as new compared to other Arduino boards. This board comes up with numerous features that helps the user to use this in their project. The Arduino UNO uses the Atmega16U2 microcontroller that helps to increase the transfer rate and contain large memory compared to other boards. No extra devices are needed for the Arduino UNO board like joystick, mouse, keyboard and many more. The Arduino UNO contain SCL and SDA pins and also have two additional pins fit near to RESET pin.

The board contains 14 digital input pins and output pins in which 6 pins are used as PWM, 6 pins as analog inputs, USB connection, reset button and one power jack. The Arduino UNO board can be attached to computer system buy USB port and also get power supply to board from computer system. The Arduino UNO contains flash memory of size 32 KB that is used to the data in it. The other feature of the Arduino UNO is compatibility with other shield and can be combined with other Arduino products.

Basic Specs:

- Microcontroller: ATmega328P
- Memory: 2kB SRAM, 32kB flash, and 1kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: Replaceable chip

2.Arduino Leonardo



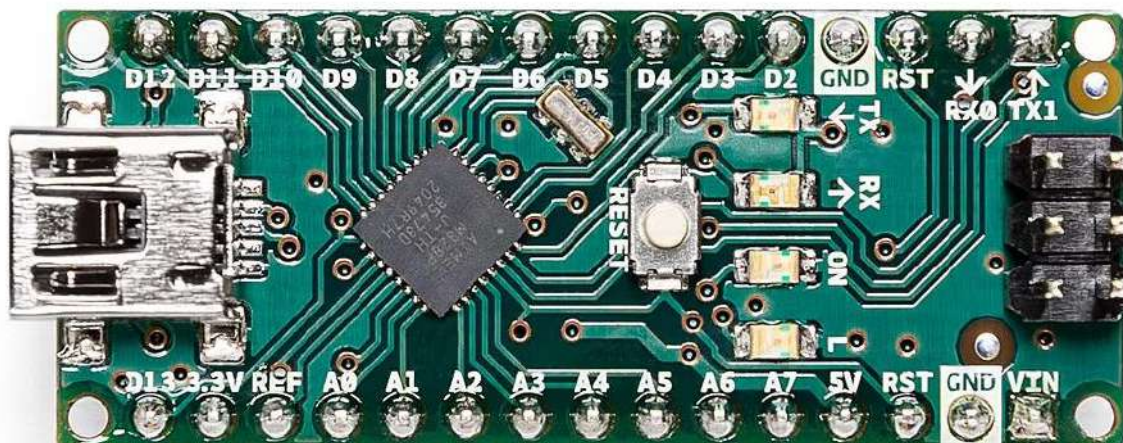
The Arduino Leonardo comes with essentially the same specifications as an Uno R3, only it features a micro-USB connector, has 20 digital and 17 analog pins, and has the ability to be used as a human interface device thanks to the ATmega32U4 chip that powers it. This means that your Leonardo can be used as a keyboard or mouse with a USB cable attached.

This type of Arduino is ideal for simple projects that need to interact with machines like computers, providing a huge range of different ideas to try for yourself.

Basic Specs:

- Microcontroller: ATmega32U4
- Memory: 2.5kB SRAM, 32kB flash, and 1kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: HID connectivity

3.Arduino Nano / Nano Every



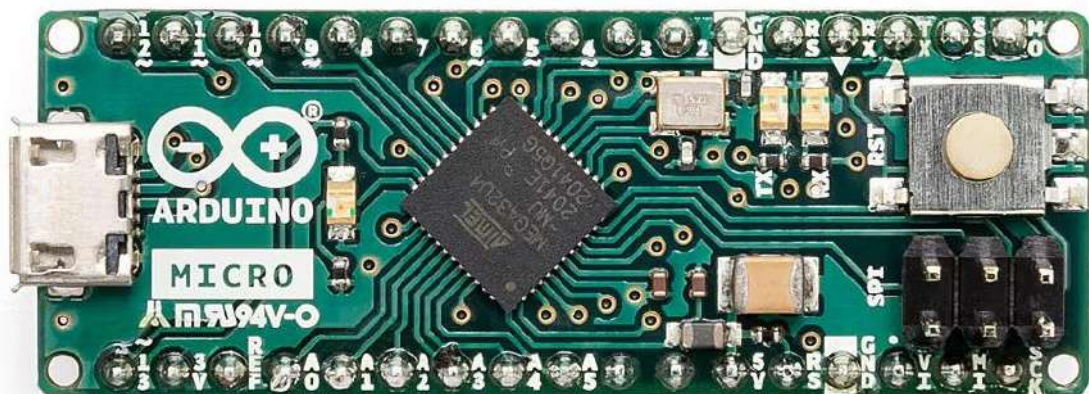
The Arduino Nano and Nano Every are the smallest microcontroller boards offered by the company. Both boards feature the same pin layout, with 14 digital pins and 8 analog pins, though the Nano Every has a beefier microcontroller chip and improved program memory. These boards both come with pre-soldered headers that make them ideal for use with breadboards, but they lack the power jack that comes on larger boards.

Their breadboard compatibility makes these small boards great for those who like to make circuits that change all the time, like school teachers and prototype makers.

Basic Specs:

- Microcontroller: ATmega32U4 (Nano); ATmega4809 (Nano Every)
- Memory: 2kB SRAM, 32kB flash, and 1kB EEPROM (Nano); 6kB SRAM, 48kB flash, and 256B EEPROM (Nano Every)
- Communication: UART, IC2, and SPI
- Special Features: Breadboard-compatible and extremely small

4.Arduino Micro



The Arduino Micro boasts very similar features to the Leonardo, only the board is much smaller and only features 12 analog pins alongside its 20 digital ones. At only 18mm wide and 48mm long, this board is one of the smallest Arduino has ever made, making it ideal for creating a keyboard, mouse, and or other HID devices that need to be tiny.

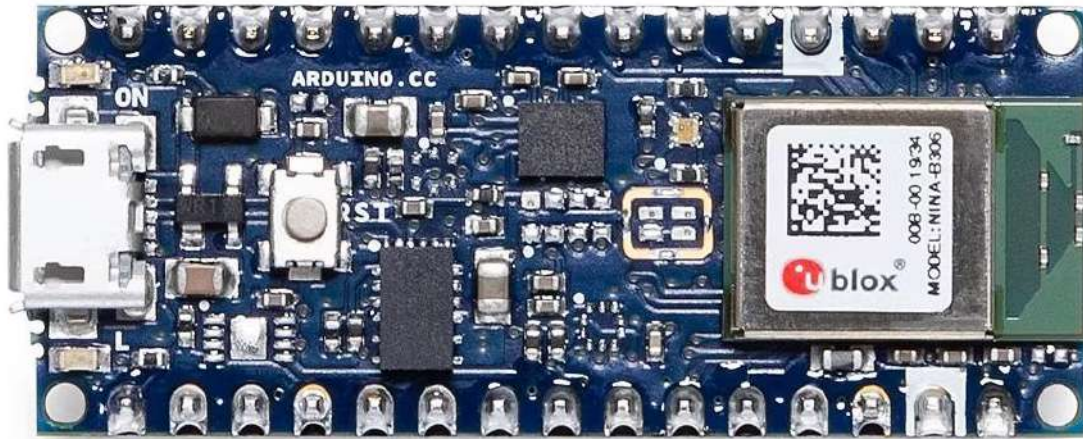
Basic Specs:

- Microcontroller: ATmega32U4
- Memory: 2.5kB SRAM, 32kB flash, and 1kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: HID connectivity and small form-factor

Enhanced Arduino Boards

Enhanced Arduino boards offer the features required to complete projects with greater complexity, while also providing improved performance for DIYers looking to push the limits.

1.Arduino Nano 33 BLE / Nano 33 BLE Sense



The Nano 33 BLE / Nano 33 BLE Sense is designed as an improved version of the Arduino Nano / Nano Every, featuring the same pin layout to make it nice and easy for DIYers. Both boards have a 32-bit Arm Cortex-M4 CPU running at 64MHz built into their nRF52840 chips, with 1MB of flash memory and 256kB of SRAM, making these boards incredibly powerful despite their small size.

They only come with 14 digital pins, but are packed with a host of sensors that don't come with regular Nanos. This sensor array includes an accelerometer, a gyroscope, and a magnetometer with 3-axis resolution, and the board comes with Bluetooth Low Energy (BLE) that makes it easy to transmit the data it collects.

Alongside all of these great features, the Nano 33 BLE Sense is also able to run edge computing applications using machine learning models from TensorFlow Lite.

Basic Specs:

- Microcontroller: nRF52840
- Memory: 256kB SRAM and 1MB flash
- Communication: UART, IC2, and SPI
- Special Features: Sensors, Bluetooth, and AI (Sense only)

2.Arduino MKR Zero

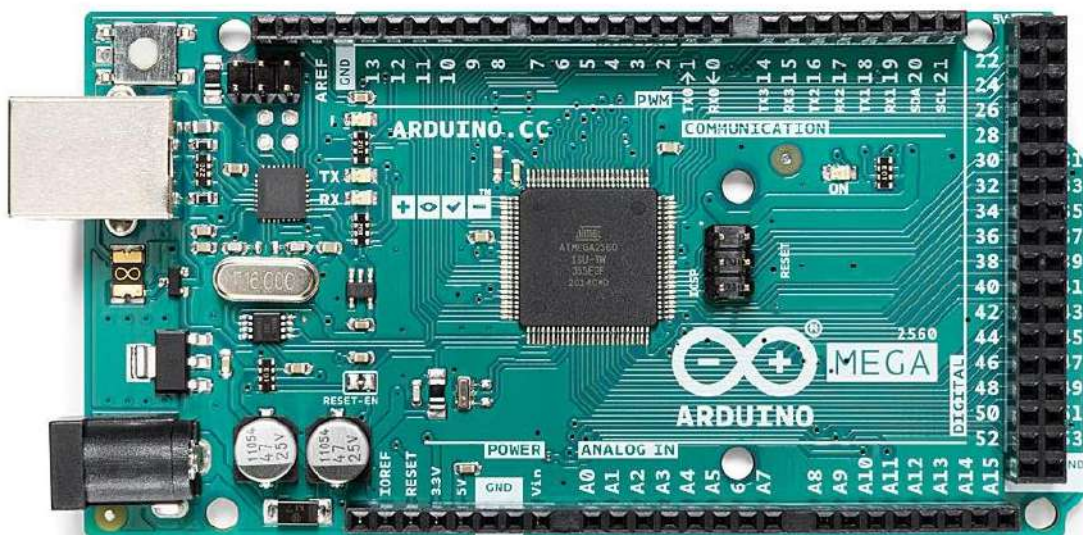


The Arduino MKR Zero is designed with music-making and other complex projects in mind, featuring a powerful Arm Cortex-M0 32-bit SAMD21 processor, native battery support, and a built-in microSD card reader. The board comes with 8 digital pins, 7 analog input pins, and 1 analog output pin. Thanks to the expandable storage that comes with this board, it is an excellent option for those working with a lot of code and a range of hardware components.

Basic Specs:

- Microcontroller: Arm Cortex-M0 32-bit SAMD21
- Memory: 32kB SRAM and 256kB flash
- Communication: UART, IC2, and SPI
- Special Features: Built-in battery connector, microSD card reader, powerful hardware

3.Arduino Mega 2560 R3



The Arduino Mega 2560 is similar to an Arduino Uno, only it features 54 digital pins, 16 analog pins, and 4 serial ports, along with being much larger and more powerful than the standard Uno. This board is great for DIYers in need of more pins, memory, or processing power without sacrificing the functionality that comes with regular Arduinos.

Basic Specs:

- Microcontroller: ATmega2560
- Memory: 8kB SRAM, 256kB flash, and 4kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: Large form-factor and serial ports

Arduino Toolchain

A toolchain is a set of software development tools used simultaneously to complete complex software development tasks or to deliver a software product. Each tool in the chain is itself a piece of software that serves a different function and is optimized to work together with other tools in the chain. Example programs in a toolchain are an assembler, linker and debugger.

Companies can customize toolchains to suit their software needs. Some software vendors also offer their own toolchains. One of the main benefits of using a toolchain instead of a disparate set of tools is the consistent programming environment throughout the development process and the smooth transition between tools as development progresses.

Example:

In programming, a toolchain is just a series of programming tools that work together in order to complete a task.

If we were cooking something, like carrot soup, the toolchain might include a carrot peeler, a produce knife, a pot, some water, and a stove.



Together, these tools help you create the desired output, carrot soup.

When we're developing programs for Arduino, we'll also be using a toolchain. This all happens behind the scenes. On our end, we literally only have to press a button for the code to get to the board, but it wouldn't happen without the toolchain.

The Arduino has a similar toolchain. When we start writing our code, and we become the author. We do this in the Arduino IDE, which is akin to a text editor. We also write the code in a programming language called C++, with a file type extension of .ino

The code that we write is called human readable code since it's meant for us to read and, hopefully, understand. However, the Arduino hardware doesn't understand C++ code.

It needs what's called machine code. In order to generate that machine code, we use a software tool called a compiler. Remember that Verify button in the Arduino IDE that looks like the checkmark?



When you press this button, it compiles the code using compiler software called AVR-GCC. This compiler software does a bunch of stuff. The main thing it does it rearrange some code and check for errors. This is like the professional editor at the publishing company. The compiler also translates the human readable code into machine code. There are some in-between file types and other fuzzy things that go on, but the final output of the compiler is a machine language saved as a .hex file.

In order to get this .hex file loaded onto our Arduino's integrated circuit, we need to press the Upload button.



This begins another piece of software called AVRDUDE, which sends the file to the integrated circuit.

Normally, we would have to use some external hardware to load that circuit. However, in Arduino's infinite wisdom, they loaded a small piece of software onto the integrated circuit that automatically comes with an Arduino board.

That piece of software is called a bootloader. It works with ARVDUDE to take the outputted .hex file and put it onto the flash memory of that Arduino's integrated circuit using only the USB cable.

Again, all we had to do was press the Upload button. This whole process happens behind the scenes.

Now the process is complete

What is the purpose of a toolchain?

The purpose of a toolchain is to have a group of linked software tools that are optimized for a specific programming process. The output generated by one tool in the chain is used by the next tool as the input.

By integrating tools together, the toolchain considers the various tool dependencies and, as a result, streamlines the software development process. For example, a programming language such as C++ may require language-specific tools, like a debugger or compiler. A development team that knows C++ can integrate one of these tools into their chain to avoid bottlenecks in the development process.

What should a toolchain include?

Toolchains may vary depending on the team using them and the product being delivered. However, basic toolchains often include the following components:

Assembler -- converts assembly language into code.

Linker -- links a set of program files together into a single program.

Debugger -- used to test and debug programs.

Compiler -- used to generate executable code from the source code of a program.

Runtime libraries -- used to interface with an operating system (OS). It enables the program to reference external functions and resources. An API is an example of this.

Developers will often write a script that links these tools together and automates some of the process. Complex software products will likely require more tools than this, but this basic assortment is generally involved in any software toolchain in some form, regardless of the product being delivered.

Some coding languages and platforms offer their own toolchains and enable customization within the integrated development environment platform. Apple's Xcode is one example of this.

Benefits of using toolchains

The main benefit of using a software toolchain is that it streamlines the development lifecycle. To a degree, the software development process can be automated, which is especially useful for DevOps implementations. In DevOps implementations, teams often employ a continuous delivery strategy and need all tools working simultaneously to be as efficient as possible.

Additionally, using a pre-built toolchain from a third party allows development teams to bypass the step of building a toolchain from scratch, which can be difficult and time consuming. Developers can use these preexisting toolchains and then customize them to the specific use case.

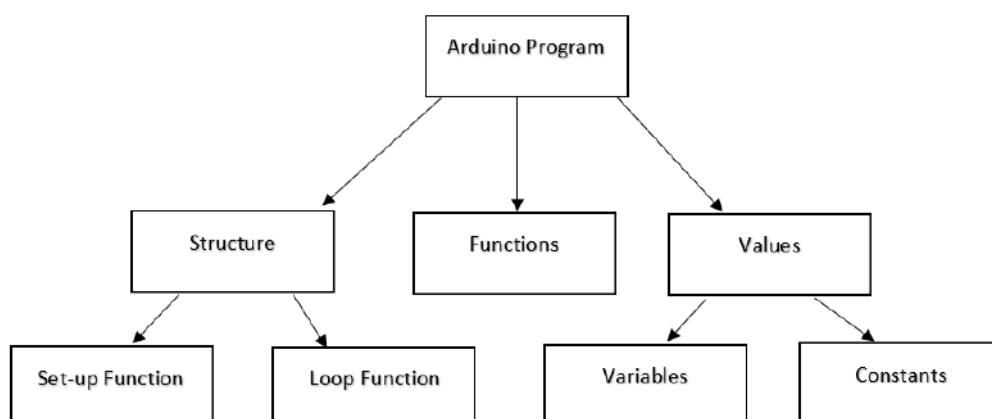
Using toolchains can also help expedite handoffs between teams that use different software tools by increasing visibility, security and productivity.

Arduino Programming Structure

Sketch – The first new terminology is the Arduino program called “sketch”, Different languages can be used to execute different functions by using electronic machines. These languages aid in giving commands to the machine. There are a lot of different programming languages, and each language has its own commands, syntax, and structure of writing a program. The language used for Arduino is C++. The Arduino program structure is briefly explained in this discourse.

Arduino Programing Overview

The Arduino program is divided into three main parts that are structure, values, and functions.



1. Structure

Structure consist of two main functions –

Setup() function:

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup() function will only run once, after each powerup or reset of the Arduino board.

Eg:

```
1 int buttonPin = 3;
2
3 void setup() {
4   // put your setup code here, to executed once:
5   Serial.begin(9600);
6   pinMode(buttonPin, INPUT);
7   Serial.println("This is setup code");
8 }
9
10 void loop() {
11   // ...
12 }
```

Output:



Loop() function:

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Eg:

```
1 void setup() {
2   // put your setup code here, to executed once:
3   Serial.begin(9600);
4   Serial.println("This is setup code");
5 }
6
7 void loop() {
8   // put your main code here, to run repeatedly:
9   Serial.println("This is loop code");
10  delay(1000);
11 }
```

Output:



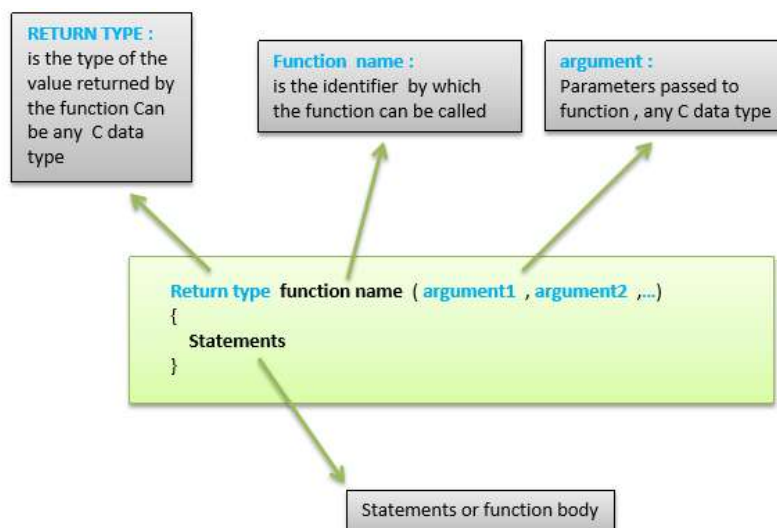
```
COM6
This is setup code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
```

2.Functions

Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Standardizing code fragments into functions has several advantages –

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought about and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it modular, and using functions often makes the code more readable.



Function Declaration

A function is declared outside any other functions, above or below the loop function.

We can declare the function in two different ways –

The first way is just writing the part of the function called **a function prototype** above the loop function, which consists of –

- Function return type
- Function name
- Function argument type, no need to write the argument name

Function prototype must be followed by a semicolon (;).

The following example shows the demonstration of the function declaration using the first method.

Example

```
int sum_func (int x, int y) // function declaration {
  int z = 0;
  z = x+y;
  return z; // return the value
}

void setup () {
  Statements // group of statements
}

Void loop () {
  int result = 0;
  result = Sum_func (5,6); // function call
}
```

3.Values

A value is a piece of data that can be manipulated by the program. This could be a number, a character, a string of text, or a logical value (such as "true" or "false"). In Arduino programs, values are often stored in variables, which are named containers that can hold a single value. These values can then be used in calculations, compared to other values, or used to control the behavior of the program. For example, a variable named "x" might be used to store the current temperature reading from a sensor, which could then be used to control a heating or cooling system.

Values Are 2 Types

1.Variable

2.Constant

1.Variable:

The variables are defined as the place to store the data and values. It consists of a name, value, and type.

The variables can belong to any data type such as int, float, char, etc.

Consider the below example:

```
int pin = 8;
```

2.Constant:

The constants in Arduino are defined as the predefined expressions. It makes the code easy to read. The constants in Arduino are defined as:

Logical level Constants:

The logical level constants are true or false.

The value of true and false are defined as 1 and 0. Any non-zero integer is determined as true in terms of Boolean language. The true and false constants are type in lowercase rather than uppercase (such as HIGH, LOW, etc.).

Pin level Constants:

The digital pins can take two value HIGH or LOW.

In Arduino, the pin is configured as INPUT or OUTPUT using the pinMode() function. The pin is further made HIGH or LOW using the digitalWrite() function.

Input/Output From Pins Using Sketches

In the Arduino platform, a sketch is a program written in the Arduino programming language. Sketches are used to control the behavior of the Arduino board, including reading input from and writing output to the board's pins.

Input and output can be performed using the digital and analog pins on the Arduino board. Digital pins can be used to read digital inputs, such as the state of a button, or to write digital outputs, such as turning on an LED. Analog pins, on the other hand, can be used to read analog inputs, such as the output of a sensor, or to write analog outputs, such as dimming an LED.

To read and write to the pins, the sketch must use the appropriate Arduino functions. For example, the *digitalRead()* function can be used to read the state of a digital input pin, while the *analogWrite()* function can be used to write an analog output to a pin. These functions are typically used within the setup() and loop() functions of the sketch, which are special functions that are called automatically when the sketch is executed.

Here is an example of a simple sketch that reads a digital input from pin 2 and writes the state of that input to an LED connected to pin 13:

```
void setup() {  
  // Set the LED pin as an output  
  pinMode(13, OUTPUT);  
}  
  
void loop() {  
  // Read the state of the input pin  
  int inputState = digitalRead(2);  
  
  // Write the state of the input to the LED  
  digitalWrite(13, inputState);  
}
```

In this sketch, the `setup()` function is used to configure pin 13 as an output, and the `loop()` function is used to read the input from pin 2 and write it to the LED on pin 13. This sketch will continue to run indefinitely, reading the input and updating the LED every time the `loop()` function is called.

Introduction to Arduino Shields

Arduino shields:

Arduino shields are the boards, which are plugged over the Arduino board to expand its functionalities. There are different varieties of shields used for various tasks, such as Arduino motor shields, Arduino communication shields, etc.

Shield is defined as the hardware device that can be mounted over the board to increase the capabilities of the projects. It also makes our work easy. For example, Ethernet shields are used to connect the Arduino board to the Internet.

The pin position of the shields is similar to the Arduino boards. We can also connect the modules and sensors to the shields with the help of the connection cable.

Arduino motor shields help us to control the motors with the Arduino board.

Why do we need Shields?

The advantages of using Arduino shields are listed below:

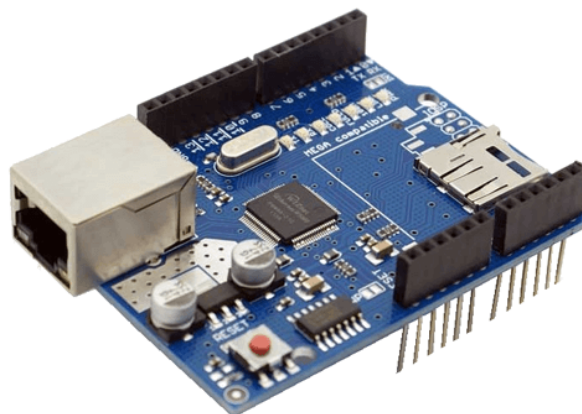
- It adds new functionalities to the Arduino projects.
- The shields can be attached and detached easily from the Arduino board. It does not require any complex wiring.
- It is easy to connect the shields by mounting them over the Arduino board.
- The hardware components on the shields can be easily implemented.

Types of Shields

The popular Arduino shields are listed below:

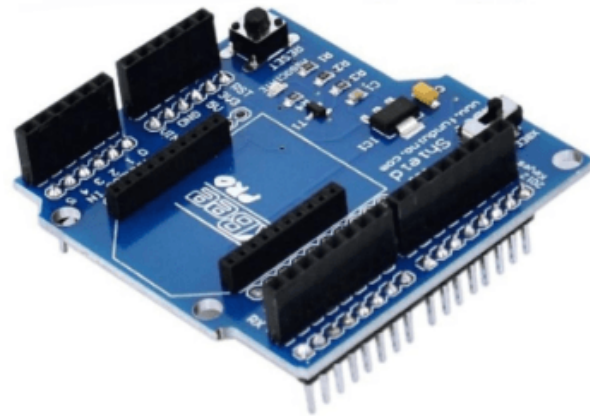
- **Ethernet shield**
- **Xbee Shield**
- **Proto shield**
- **Relay shield**
- **Motor shield**
- **LCD shield**
- **Bluetooth shield**
- **Capacitive Touchpad Shield**

1.Ethernet shield



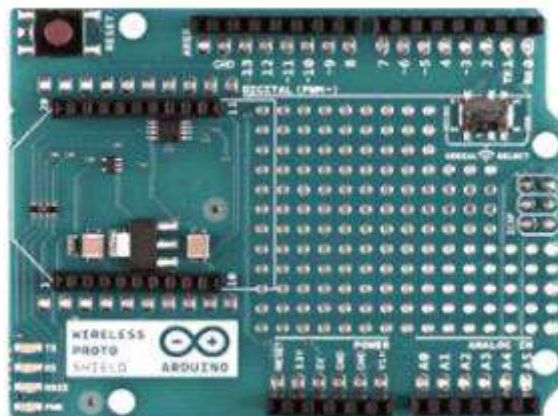
- The Ethernet shields are used to connect the Arduino board to the Internet. We need to mount the shield on the top of the specified Arduino board.
- The USB port will play the usual role to upload sketches on the board.
- The latest version of Ethernet shields consists of a micro SD card slot. The micro SD card slot can be interfaced with the help of the SD card library.
- We can also connect another shield on the top of the Ethernet shield. It means that we can also mount two shields on the top of the Arduino board.

2.Xbee Shield



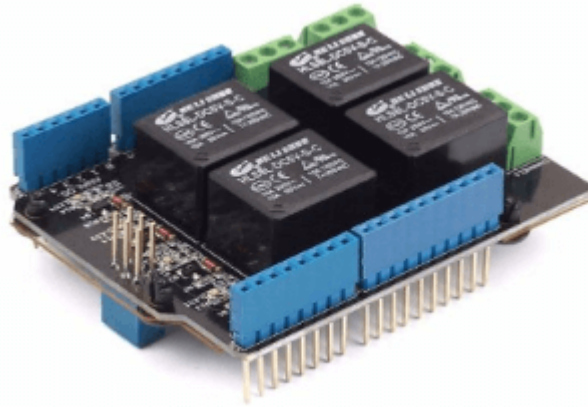
- We can communicate wirelessly with the Arduino board by using the Xbee Shield with Zigbee.
- It reduces the hassle of the cable, which makes Xbee a wireless communication model.
- The Xbee wireless module allows us to communicate outdoor upto 300 feet and indoor upto 100 feet.
- It can also be used with different models of Xbee.

3.Proto shield



- Proto shields are designed for custom circuits.
- We can solder electronic circuits directly on the shield.
- The shield consists of two LED pads, two power lines, and SPI signal pads.
- The IOREF (Input Output voltage REference) and GND (Ground) are the two power lines on the board.
- We can also solder the SMD (Surface Mount Device) ICs on the prototyping area. A maximum of 24 pins can be integrated onto the SMD area.

4. Relay shield



- The Arduino digital I/O pins cannot bear the high current due to its voltage and current limits. The relay shield is used to overcome such situation. It provides a solution for controlling the devices carrying high current and voltage.
- The shield consists of four relays and four LED indicators.
- It also provides NO/NC interfaces and a shield form factor for the simple connection to the Arduino board.
- The LED indicators depicts the ON/OFF condition of each relay.
- The relay used in the structure is of high quality.
- The NO (Normally Open), NC (Normally Closed), and COM pins are present on each relay.
- The applications of the Relay shield include remote control, etc.

5. Motor shield



- The motor shield helps us to control the motor using the Arduino board.
- It controls the direction and working speed of the motor. We can power the motor shield either by the external power supply through the input terminal or directly by the Arduino.
- We can also measure the absorption current of each motor with the help of the motor shield.
- The motor shield is based on the L298 chip that can drive a step motor or two DC motors. L298 chip is a full bridge IC. It also consists of the heat sinker, which increases the performance of the motor shield.

- It can drive inductive loads, such as solenoids, etc.
- The operating voltage is from 5V to 12V.
- The applications of the motor shield are intelligent vehicles, micro-robots, etc.

6.LCD shield



- The keypad of LCD (Liquid Crystal Display) shield includes five buttons called as up, down, left, right, and select.
- There are 6 push buttons present on the shield that can be used as a custom menu control panel.
- It consists of the 1602 white characters, which are displayed on the blue backlight LCD.
- The LED present on the board indicates the power ON.
- The five keys present on the board helps us to make the selection on menus and from board to our project.
- The LCD shield is popularly designed for the classic boards such as Duemilanove, UNO, etc.

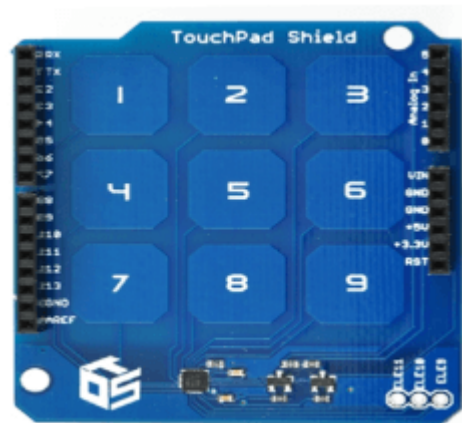
7.Bluetooth shield



- The Bluetooth shield can be used as a wireless module for transparent serial communication.
- It includes a serial Bluetooth module. D0 and D1 are the serial hardware ports in the Bluetooth shield, which can be used to communicate with the two serial ports (from D0 to D7) of the Arduino board.

- We can install Groves through the two serial ports of the Bluetooth shield called a Grove connector. One Grove connector is digital, while the other is analog.
- The communication distance of the Bluetooth shield is upto 10m at home without any obstacle in between.

8.Capacitive Touchpad shield



- It has a touchpad interface that allows to integrate the Arduino board with the touch shield.
- The Capacitive touchpad shield consists of 12 sensitive touch buttons, which includes 3 electrode connections and 9 capacitive touch pads.
- The board can work with the logic level of 3.3V or 5V.
- We can establish a connection to the Arduino project by touching the shield.

Integration of Sensors and Actuators with Arduino

Sensor

A sensor is a device that detects and responds to some type of input from the physical environment. The input can be light, heat, motion, moisture, pressure or any number of other environmental phenomena. The output is generally a signal that is converted to a human-readable display at the sensor location or transmitted electronically over a network for reading or further processing.

Sensors play a pivotal role in the internet of things (IoT). They make it possible to create an ecosystem for collecting and processing data about a specific environment so it can be monitored, managed and controlled more easily and efficiently. IoT sensors are used in homes, out in the field, in automobiles, on airplanes, in industrial settings and in other environments.

Actuators

An actuator is a device that actuates something. In simple words, it produces motion. It helps other devices move and interact with their surrounding environment. The actuator takes in energy and uses it to help the device move. Servo motors and DC motors are types of electric actuators.

Integration of IR Sensor with Arduino

WHAT IS IR SENSOR?

An IR sensor is a device that measures the Infrared radiation in its surroundings and gives an electric signal as an output. An IR sensor can measure the heat of an object as well as can detect the motion of the objects.

IR technology is used in our day-to-day life and also in industries for different purposes. For example, TVs use an IR sensor to understand the signals which are transmitted from a remote control. The main benefits of IR sensors are low power usage, their simple design & their convenient features. IR signals are not noticeable by the human eye. The IR radiation in the electromagnetic spectrum can be found in the regions of the visible & microwave. Usually, the wavelengths of these waves range from $0.7 \mu\text{m}$ to $1000\mu\text{m}$. The IR spectrum can be divided into three regions near-infrared, mid, and far-infrared. The near IR region's wavelength ranges from $0.75 - 3\mu\text{m}$, the mid-infrared region's wavelength ranges from 3 to $6\mu\text{m}$ & the far IR region's infrared radiation's wavelength is higher than $6\mu\text{m}$.



As the Infrared Radiation is invisible to our eyes, it can be detected by the Infrared Sensor. An IR sensor is a photodiode that is sensitive to IR light. When IR light falls on the photodiode, the resistances and the output voltages will change in proportion to the magnitude of the IR light received.

TYPES OF IR SENSORS

1. Active IR Sensor.
2. Passive IR Sensor.

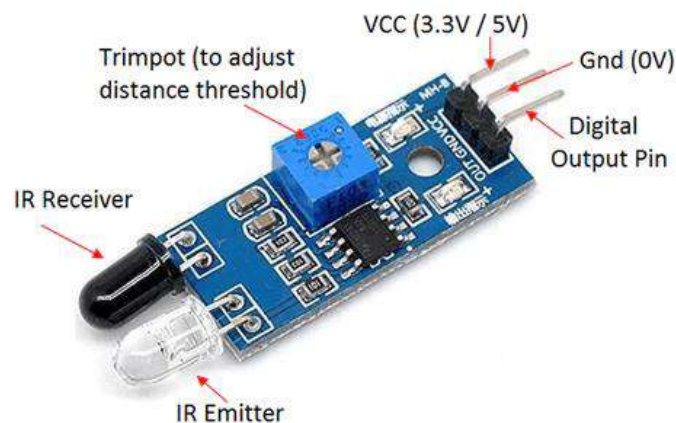
1. ACTIVE IR SENSOR

Active infrared sensors both emit and detect infrared radiation. Active IR sensors have two parts: a light-emitting diode (LED) as an Infrared Radiation transmitter and a Photodiode as an Infrared Radiation receiver. When an object comes close to the sensor, the infrared light from the LED reflects off of the object and is detected by the receiver. Active IR sensors act as proximity sensors, and they are commonly used in obstacle detection systems (such as in robots).

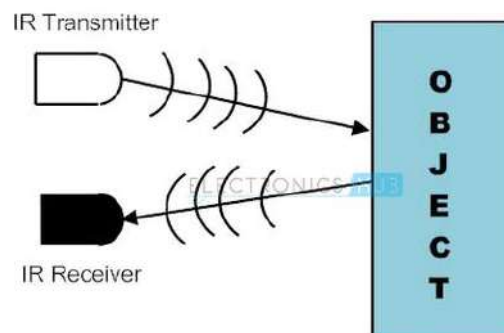
2. PASSIVE IR SENSOR

Passive infrared (PIR) sensors only detect infrared radiation and do not emit it from an LED. PIR sensors are most commonly used in motion-based detection, such as in-home security systems. When a moving object that generates infrared radiation enters the sensing range of the detector, the difference in IR levels between the two pyroelectric elements is measured. The sensor then sends an electronic signal to an embedded computer, which in turn triggers an alarm.

HOW DOES AN IR SENSOR MODULE WORK:



As we know till now that the active IR Sensors emit Infrared waves and as-well-as detect the Infrared ways. The IR sensor module exactly works on the same phenomenon. The IR Sensor module contains an Infrared LED and an infrared photodiode.

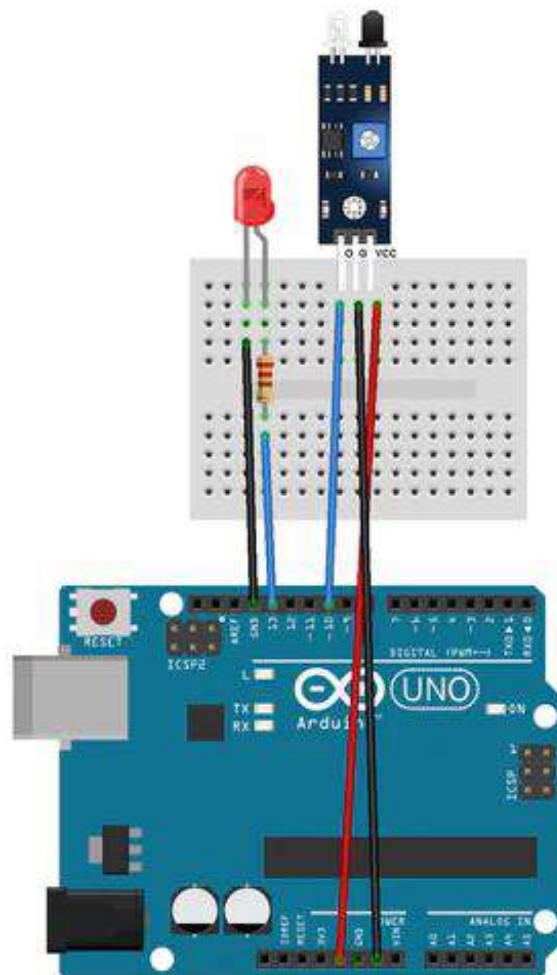


INTERFACING IR SENSOR MODULE WITH ARDUINO

HARDWARE REQUIRED

We need the following components:

1. Arduino
2. Breadboard
3. Jumper wires
4. IR Sensor module
5. LED
6. 220-ohm resistor



Code:

```
int LEDpin = 13;
int obstaclePin = 10;
int hasObstacle = LOW; // LOW MEANS NO OBSTACLE

void setup() {
  pinMode(LEDpin, OUTPUT);
  pinMode(obstaclePin, INPUT);
  Serial.begin(9600);
}

void loop() {
  hasObstacle = digitalRead(obstaclePin);
  if (hasObstacle == HIGH) {
    Serial.println("Stop something is ahead!!");
    digitalWrite(LEDpin, HIGH);
  }
  else {
    Serial.println("Path is clear");
    digitalWrite(LEDpin, LOW);
  }
  delay(200);
}
```

UPLOADING THE CODE TO THE ARDUINO BOARD:

Step 1: Connect the Arduino board with the computer using a USB cable.

Step 2: Next type the code mentioned above.

Step 3: Select the right board and port.

Step 4: Upload the code to the Arduino.

Step 5: The LED will glow when any obstacle is detected by the IR sensor.

Integration of Actuator with Arduino

Integrating servo motor with Arduino Uno. Servo Motor is an electrical linear or rotary actuator which enables precise control of linear or angular position, acceleration or velocity. Usually servo motor is a simple motor controlled by a servo mechanism, which consists of a positional sensor and a control circuit. Servo motor is commonly used in applications like robotics, testing automation, manufacturing automation, CNC machine etc. The main characteristics of servo motor are low speed, medium torque, and accurate position.

Components Required

- Arduino Uno
- Servo Motor
- Jumper Wires

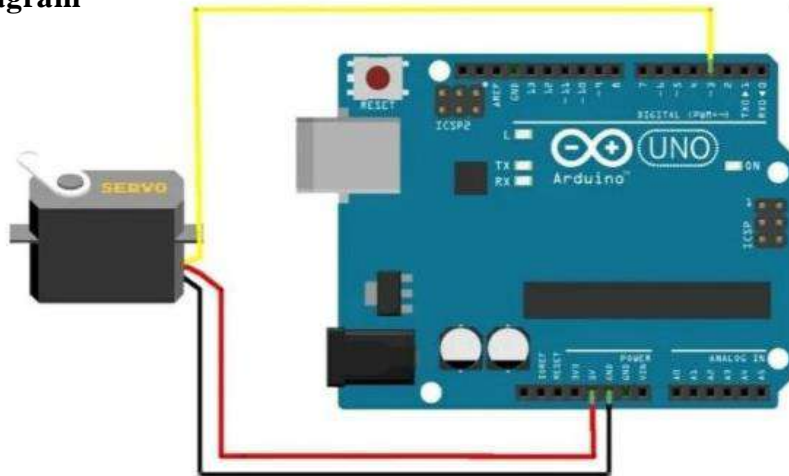
Hobby Servo Motor



Specifications

- Servo Motor consists of three pins: VCC, GROUND, and PWM signal pin.
- The VCC and GND is for providing power supply, the PWM input pin is used for controlling the position of the shaft by varying pulse width.
- Minimum Rotational Angle : 0 degrees
- Maximum Rotational Angle : 180 degrees
- Operating Voltage : +5V
- Torque : 2.5 kg/cm
- Operating Speed : 0.1 s/60 degree
- DC Supply Voltage : 4.8V to 6V

Circuit Diagram



Description

- The VCC pin (Red Color) of the Servo Motor is connected to the 5V output of the Arduino Board.
- The GND pin (Brown Color) of the Servo Motor is connected to the GND pin of the Arduino Board.
- The PWM input pin (Yellow Color) of the Servo Motor is connected to the PWM output pin of the Arduino board.

Working

The hobby servo motor which we use here consists of 4 different parts as below.

- Simple DC Motor
- Potentiometer
- Control Circuit Board
- Gear Assembly

Potentiometer is connected to output shaft of the servo motor helps to detect position. Based on the potentiometer value, the control circuit will rotate the motor to position the shaft to a certain angle. The position of the shaft can be controlled by varying the pulse width provided in the PWM input pin. Gear assembly is used to improve the torque of the motor by reducing speed.

Program

```
#include <Servo.h>

Servo servo;

int angle = 10;

void setup() {
    servo.attach(3);
    servo.write(angle);
}

void loop()
{
    for(angle = 10; angle < 180; angle++)
    {
        servo.write(angle);
        delay(15);
    }
    for(angle = 180; angle > 10; angle--)
    {
        servo.write(angle);
        delay(15);
    }
}
```

Code Explanation

- The position of the shaft is kept at 10 degrees by default and the Servo PWM input is connected to the 3rd pin of the Arduino Uno.
- In the first for loop, motor shaft is rotated from 10 degrees to 180 degrees step by step with a time delay of 15 milliseconds.
- Once it reaches 180 degree, it is programmed to rotate back to 10 degree step by step with a delay of 15 milliseconds in the second for loop.

Functioning

Wire the circuit properly as per the circuit diagram and upload the program. You can see that the servo motor is rotating as per the program.

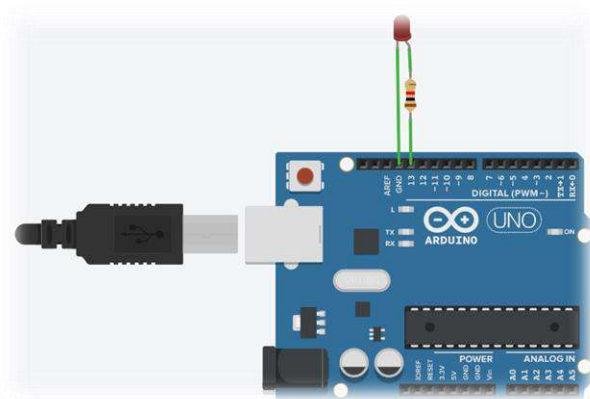
Connecting LEDs with Arduino

Light-emitting diode (LED) is a widely used standard source of light in electrical equipment. It has a wide range of applications ranging from your mobile phone to large advertising billboards. They mostly find applications in devices that show the time and display different types of data. In this Program we will Connect The LED and Blink it. Blinking means turning ON an LED for a few seconds, then OFF for seconds, repeatedly.

REQUIRED COMPONENTS

S.N.	COMPONENTS	QUANTITY
1.	Arduino	1
2.	LED	1
3.	Resistor (280 ohm)	1
4.	Connecting Wires	Few

CIRCUIT DIAGRAM



CIRCUIT EXPLANATION

Here, The anode (+) and cathode (-) terminals of LED are connected to pin 13 and ground (GND) of Arduino Uno respectively and a resistor of 280 ohm is placed between the LED anode terminal and pin number 13 which help us to limit the current and prevent LED from burning.

STEP 2: CODE

```
void setup()
{
  pinMode(13, OUTPUT);
```

```
}  
void loop() {  
digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000);  
}
```

Functioning

Wire the circuit properly as per the circuit diagram and upload the program. You can see that the the Blinking an LED as per the program.

Connecting LCD with Arduino

Display devices are used to visually display the information we are working with. LCD (Liquid Crystal Display) screens are one of many display devices that makers use. We have libraries to control specific LCD functions which make it ridiculously easy to get up and running with LCDs.

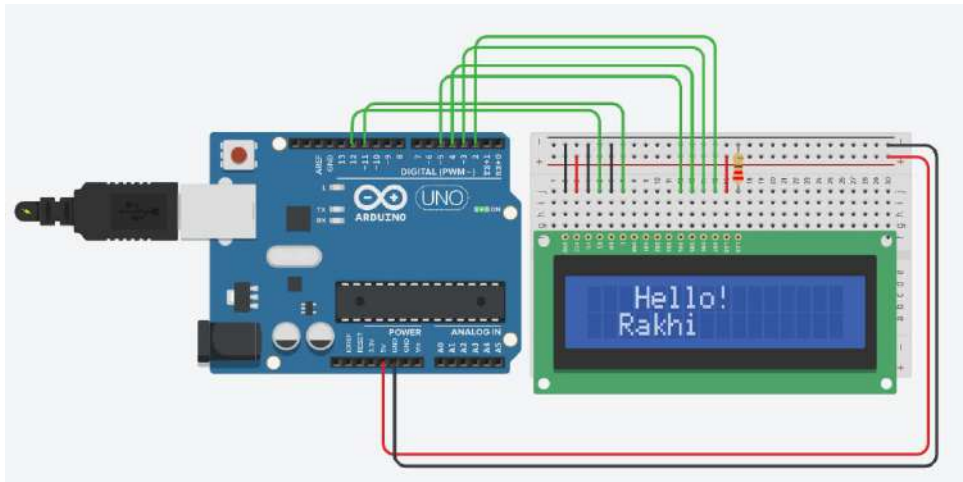
We are going to connect an LCD screen to our Arduino and make it print some stuff to the display, grab whatever you need and get started

- **Solderless Breadboard**
- **Arduino Board**
- **USB Cable**
- **Potentiometer (10k ohm)**
- **LCD Screen (16x2)**
- **Jumper Cables**

What is the LCD and how do it work?

A liquid crystal display is essentially a liquid crystal sandwiched between 2 pieces of glass, the liquid crystal reacts depending on current applied. Our LCD is a white on black, 16 by 2 character LCD that we will use to display symbols. Graphical LCDs also exist, but today we are just focusing on the character variety.

Each character is off by default and is a matrix of small dots of liquid crystal. These dots make up the numbers and letters that we display on screens. The actual coding that goes into making these characters appear is quite complicated, luckily for you, the people over at Arduino.cc have made a library of functions for the LCD that we can import into our sketch, The LCD also has a backlight and contrast control options. The backlight will shine through the pieces of glass the screen is made of to display the characters on the screen. The contrast will control how dark (or light) the characters appear. We will use a variable resistor to control contrast, and we will set the backlight to being on.



Circuit Explanation:

This circuit is the basic circuit for Arduino projects with LCD Display. If you know how to wire up a LCD Display with an Arduino then you can do any Arduino Projects including the LCD Display and the Arduino. You can use any Arduino flavor for this circuit, but just remember to put a current limiting resistor between the the pin 16 of the LCD Display to GND. Also if you want you can connect a 10k ohm Potentiometer to pin 3 of the LCD Display, remember you need to connect pin 1 of the POT to VCC - 5V and pin 2 of the POT to GND.

Code:

```
#include<LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  lcd.begin(16, 2);
}

void loop()
{
  lcd.setCursor(0,0);
  lcd.print(" Hello!");
  lcd.setCursor(2,1);
  lcd.print("Rakhi");
}
```

Functioning

Wire the circuit properly as per the circuit diagram and upload the program. You can see that the LCD Display With Some text as per the program.

Tinkercad Arduino simulation.

Arduino Simulator

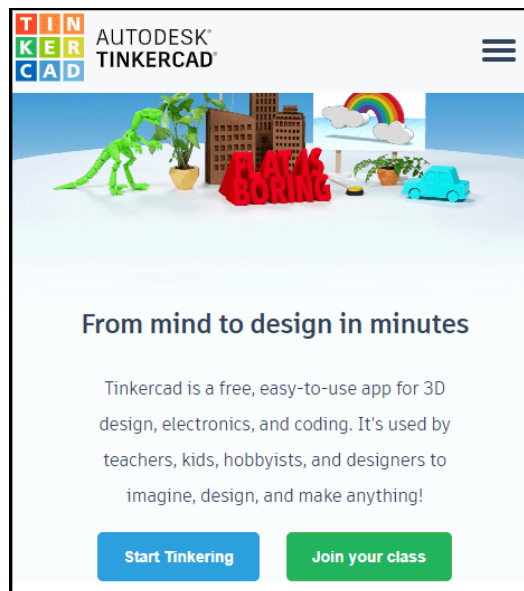
The Arduino simulator is a **virtual portrayal of the circuits of Arduino** in the real world. We can create many projects using a simulator without the need for any hardware.

The Simulator helps beginner and professional designers to learn, program, and create their projects without wasting time on collecting hardware equipments.

Autodesk Tinkercad Simulator.

The steps to access the TINKERCAD are listed below:

1. Open the official website of tinkercad. **URL:** <https://www.tinkercad.com/>



2. Click on the three horizontal lines present on the upper right corner.

3. Click on the '**Sign in**' option, if you have an account in Autodesk. Otherwise, click on the '**JOIN NOW**' option if you don't have an account.

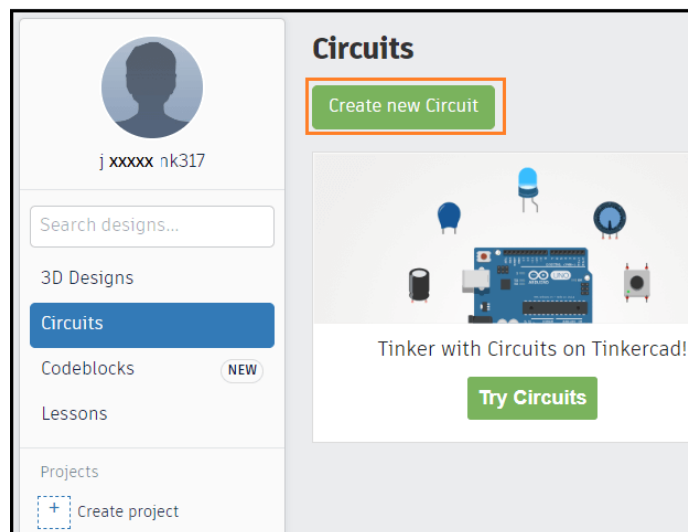
We can select any sign-in method. Specify the username and password.

We already have an account in Autodesk, so we will sign-in directly with the username and password.

The **JOIN** window will appear as:

Select the preference according to the requirements and sign-in using Gmail, etc.

4. Now, a window will appear, as shown below:



5. Click on the '**Create new circuit**' option to start designing the Arduino circuit, as shown above.

The '**Circuits**' option will also show the previous circuits created by user. The design option is used for creating the 3D design, which is of no use in Arduino.

6. We are now ready to start with the Autodesk Tinkercad. We can start creating our projects.

Features of Tinkercad

The features of Tinkercad are listed below:

- **Glow and move circuit assembly.** It means we can use the components of a circuit according to the project requirement. Glow here signifies the glowing of LED.
- **Integrated product design.** It means the electronic components used in the circuitry are real.
- **Arduino Programming.** We can directly write the program or code in the editor of the simulator.
- We can also consider some **ready-made examples** provided by the tinkercad for better understanding.
- **Realtime simulation.** We can prototype our designs within the browser before implementing them in real-time.

Advantages of using Tinkercad Simulation

There are various advantages of using simulator, which are listed below:

- It saves money, because there is no need to buy hardware equipments to make a project.
- The task to create and learn Arduino is easy for beginners.
- We need not to worry about the damage of board and related equipments.
- No messy wire structure required.
- It helps students to eliminate their mistakes and errors using simulator.
- It supports line to line debugging, and helps to find out the errors easily.
- We can learn the code and build projects anywhere with our computer and internet connection.
- We can also share our design with others.

UNIT 4

UNIT- IV Introduction to IoT

Introduction to Internet of Things: Characteristics of IoT, Design principles of IoT, IoT Architecture and Protocols, Enabling Technologies for IoT, IoT levels and IoT vs M2M. IoT Design Methodology: Design methodology, Challenges in IoT Design, IoT System Management, IoT Servers – Sensors.

Introduction to Internet of Things:

IoT comprises things that have unique identities and are connected to internet. By 2020 there will be a total of 50 billion devices /things connected to internet. IoT is not limited to just connecting things to the internet but also allow things to communicate and exchange data.

Definition:

A dynamic global n/w infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual —things have identities, physical attributes and virtual personalities and use intelligent interfaces, and are seamlessly integrated into information n/w, often communicate data associated with users and their environments.

Characteristics of IoT

1) Dynamic & Self Adapting: IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, users context or sensed environment.

Eg: the surveillance system is adapting itself based on context and changing conditions.

2) Self Configuring: allowing a large number of devices to work together to provide certain functionality.

3) Inter Operable Communication Protocols: support a number of interoperable communication protocols and can communicate with other devices and also with infrastructure.

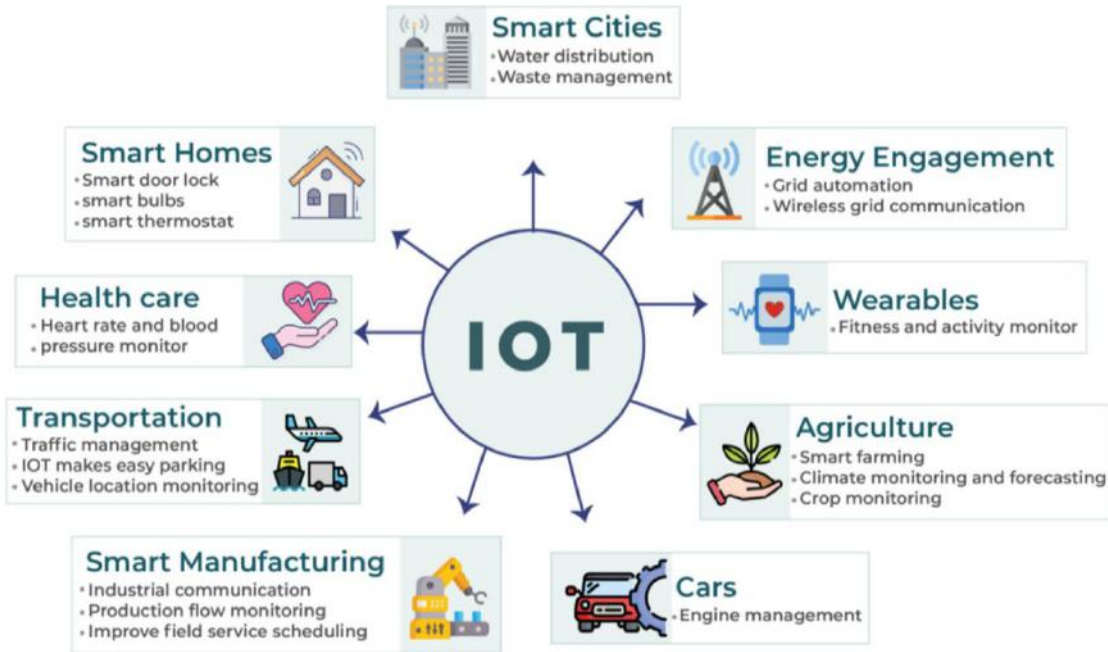
4) Unique Identity: Each IoT device has a unique identity and a unique identifier (IP address).

5) Integrated into Information Network: that allow them to communicate and exchange data with other devices and systems.

Applications of IoT:

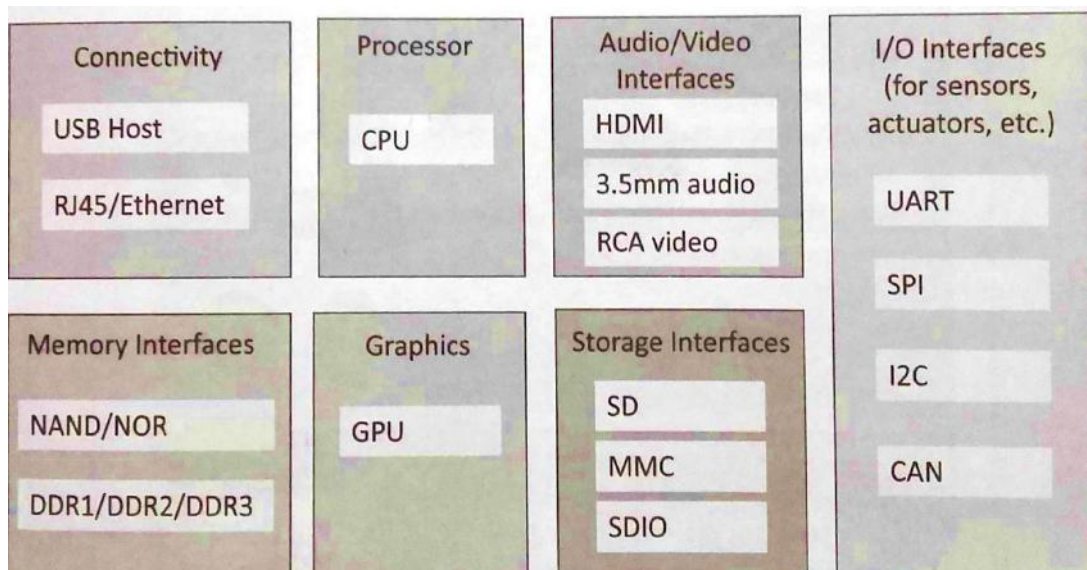
- 1) Home
- 2) Cities
- 3) Environment
- 4) Energy

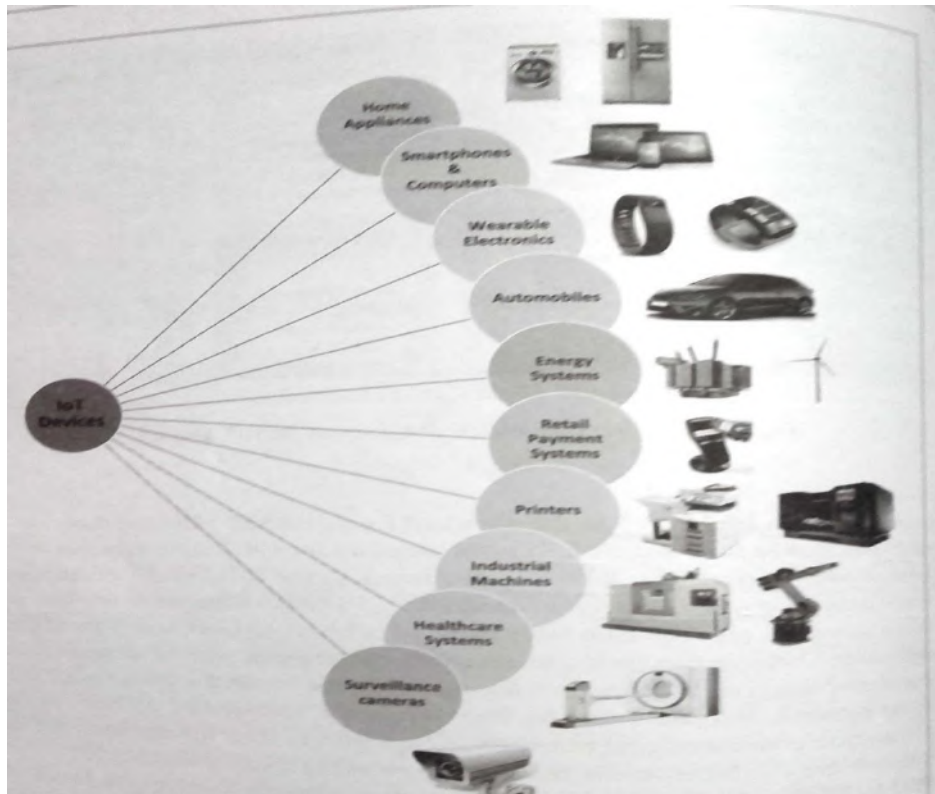
- 5) Retail
- 6) Logistics
- 7) Agriculture
- 8) Industry
- 9) Health & Life Style



Design principles of IoT:

Physical Design of IoT





The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.

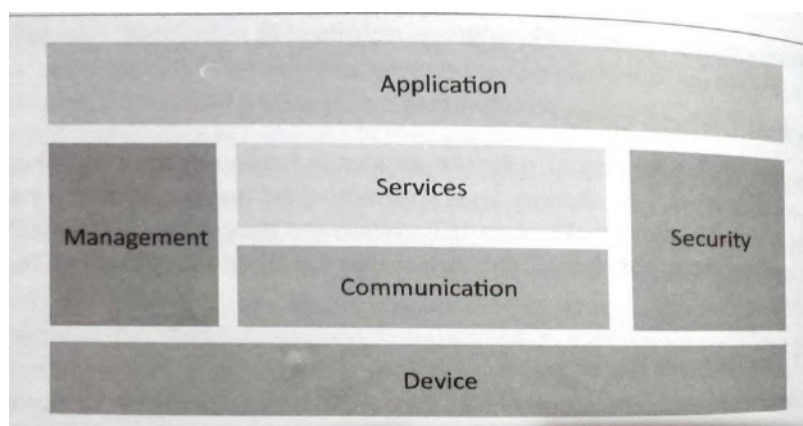
An IoT device may consist of several interfaces for communication to other devices both wired and wireless. These includes (i) I/O interfaces for sensors, (ii) Interfaces for internet connectivity (iii) memory and storage interfaces and (iv) audio/video interfaces.

LOGICAL DESIGN of IoT

Refers to an abstract represent of entities and processes without going into the low level specifics of implementation.

1) IoT Functional Blocks 2) IoT Communication Models 3) IoT Comm. APIs

1)IoT Functional Blocks:



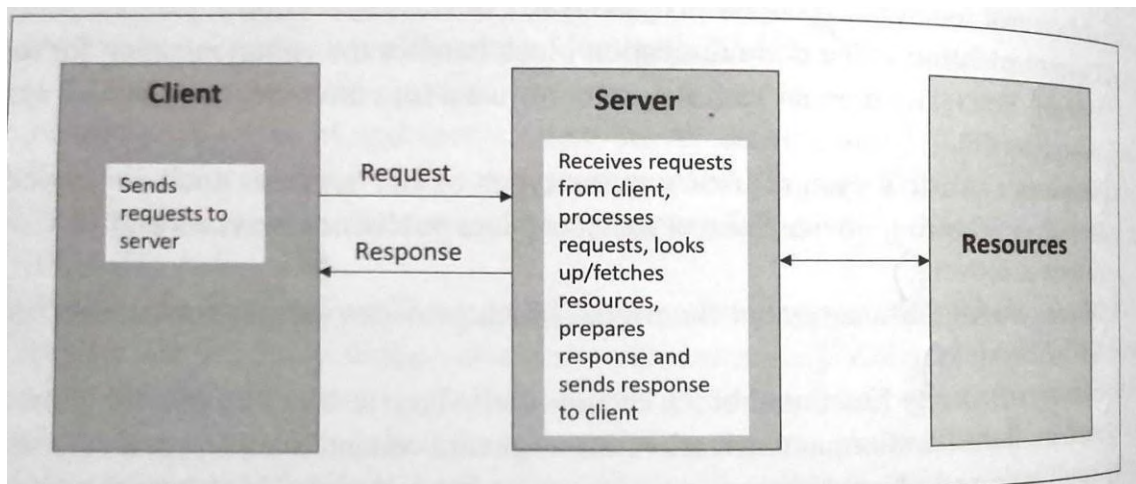
Provide the system the capabilities for identification, sensing, actuation, communication and management.

- Device:** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.
- Communication:** handles the communication for IoTsystem.
- Services:** for device monitoring, device control services, data publishing services and services for device discovery.
- Management:** Provides various functions to govern the IoT system.
- Security:** Secures IoT system and priority functions such as authentication ,authorization, message and context integrity and data security.
- Application:** IoT application provide an interface that the users can use to control and monitor various aspects of IoT system

2) IoT Communication Models:

- 1) Request-Response
- 2) Publish-Subscibe
- 3)Push-Pull
- 4) ExclusivePair

1) Request-Response Model:



In which the client sends request to the server and the server replies to requests. Is a stateless communication model and each request-response pair is independent of others.

2) Publish-Subscibe Model:

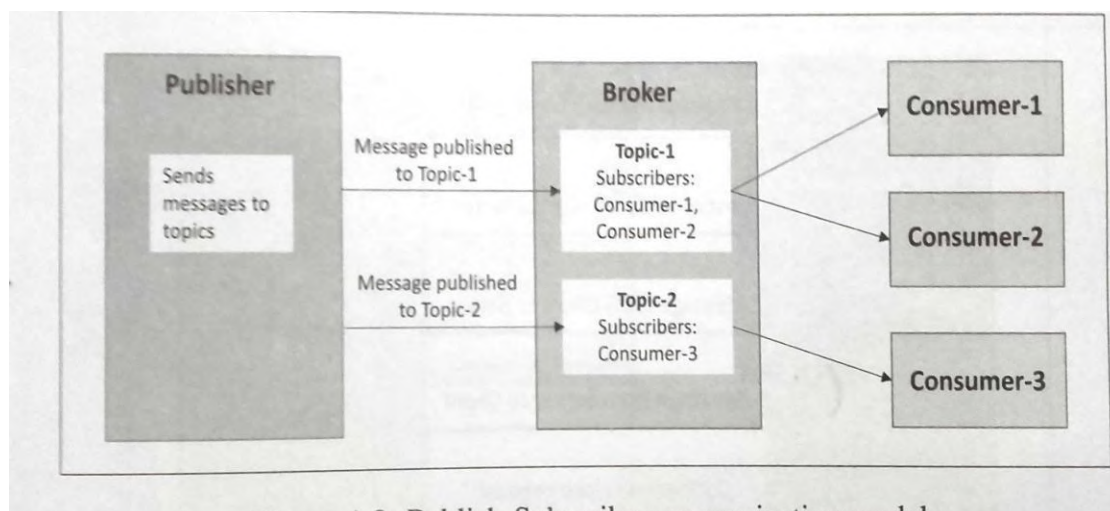
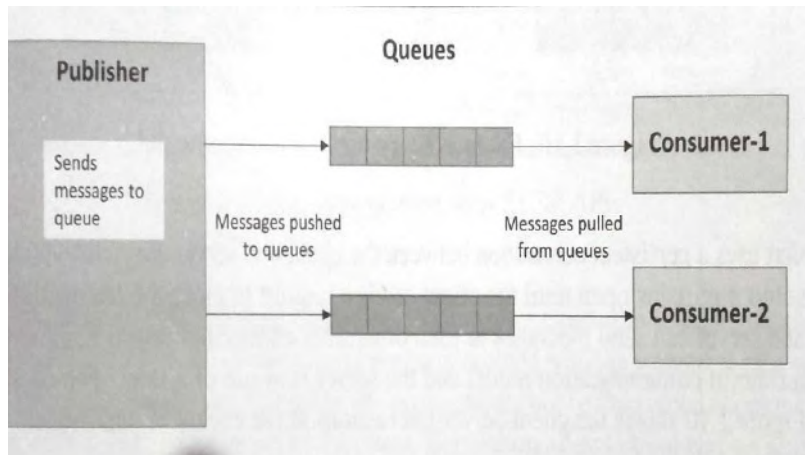


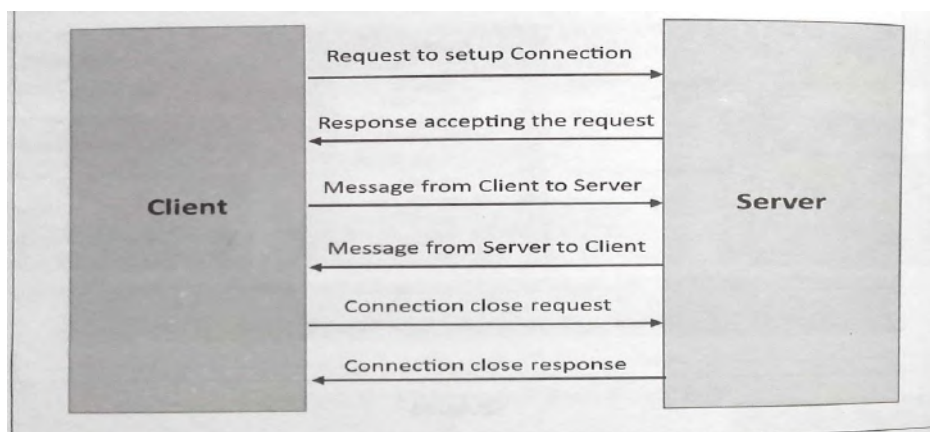
Figure 1.9: Publish-Subscribe communication model

Involves publishers, brokers and consumers. Publishers are source of data. Publishers send data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.

3) Push-Pull Model: in which data producers push data to queues and consumers pull data from the queues. Producers do not need to aware of the consumers. Queues help in decoupling the message between the producers and consumers.



4) Exclusive Pair: is bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once connection is set up it remains open until the client send a request to close the connection. Is a stateful communication model and server is aware of all the open connections.



3)IoT Communication APIs:

- a) REST based communication APIs(Request-Response Based Model)
- b) WebSocket based Communication APIs(Exclusive PairBased Model)

a) REST based communication APIs: Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a systems resources and have resource states are addressed and transferred. The REST architectural constraints.

b) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.

6 Principles of IoT design

1. Do your research

When designing IoT-enabled products, designers might make the mistake of forgetting why customers value these products in the first place. That's why it's a good idea to think about the value an IoT offering should deliver at the initial phase of your design.

When getting into IoT design, you're not building products anymore. You're building services and experiences that improve people's lives. That's why in-depth qualitative research is the key to figuring out how you can do that.

Assume the perspective of your customers to understand what they need and how your IoT implementation can solve their pain points. Research your target audience deeply to see what their existing experiences are and what they wish was different about them.

2. Concentrate on value

Early adopters are eager to try out new technologies. But the rest of your customer base might be reluctant to put a new solution to use. They may not feel confident with it and are likely to be cautious about using it.

If you want your IoT solution to become widely adopted, you need to focus on the actual tangible value it's going to deliver to your target audience.

What is the real end-user value of your solution? What might be the barriers to adopting new technology? How can your solution address them specifically?

Note that the features the early tech adopters might find valuable might turn out to be completely uninteresting for the majority of users. That's why you need to carefully plan which features to include and in what order, always concentrating on the actual value they provide.

3. Don't forget about the bigger picture

One characteristic trait of IoT solutions is that they typically include multiple devices that come with different capabilities and consist of both digital and physical touchpoints. Your solution might also be delivered to users in cooperation with service providers.

That's why it's not enough to design a single touchpoint well. Instead, you need to take the bigger picture into account and treat your IoT system holistically.

Delineate the role of every device and service. Develop a conceptual model of how users will perceive and understand the system. All the parts of your system need to work seamlessly together. Only then you'll be able to create a meaningful experience for your end-users.

4. Remember about the security

Don't forget that IoT solutions aren't purely digital. They're located in the real-world context, and the consequences of their actions might be serious if something goes wrong. At the same time, building trust in IoT solutions should be one of your main design drivers.

Make sure that every interaction with your product builds consumer trust rather than breaking it. In practice, it means that you should understand all the possible error situations that may be related to the context of its use. Then try to design your product in a way to prevent them. If error situations occur, make sure that the user is informed appropriately and provided with help.

Also, consider data security and privacy as a key aspect of your implementation. Users need to feel that their data is safe, and objects located in their workspaces or home can't be hacked.

That's why quality assurance and testing the system in the real-world context are so important.

5. Build with the context in mind

And speaking of context, it pays to remember that IoT solutions are located at the intersection of the physical and digital world. The commands you give through digital interfaces produce real-world effects. Unlike digital commands, these actions may not be easily undone.

In a real-world context, many unexpected things may happen. That's why you need to make sure that the design of your solution enables users to feel safe and in control at all times.

The context itself is a crucial consideration during IoT design. Depending on the physical context of your solution, you might have different goals in mind. For example, you might want to minimize user distraction or design devices that will be resistant to the changing weather conditions.

The social context is an important factor, as well. Don't forget that the devices you design for workspaces or homes will be used by multiple users.

6. Make good use of prototypes

IoT solutions are often difficult to upgrade. Once the user places the connected object somewhere, it might be hard to replace it with a new version – especially if the user would have to pay for the upgrade.

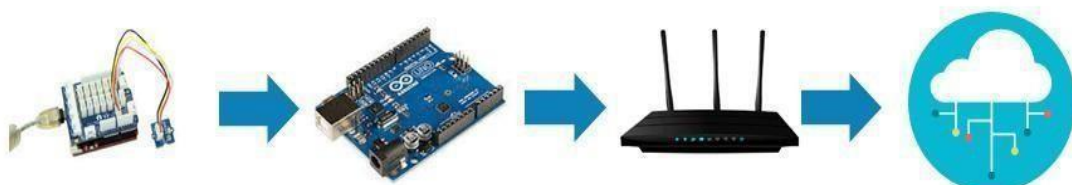
Even the software within the object might be hard to update because of security and privacy reasons. Make sure that your design practices help to avoid costly hardware iterations. Get your solution right from the start. From the design perspective, it means that prototyping and rapid iteration will become critical in the early stages of the project.

IoT Architecture and Protocols

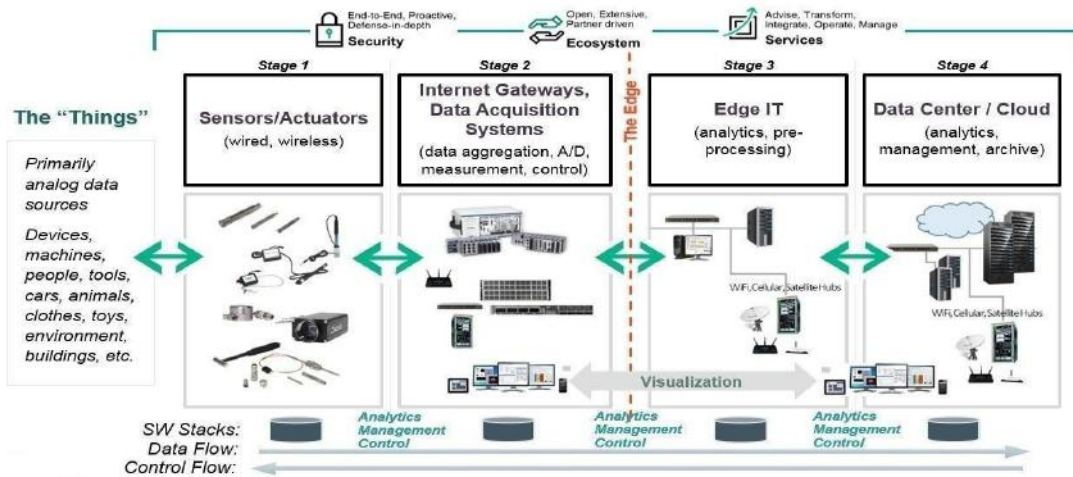
An IoT architecture is the system of numerous elements that range from sensors, protocols, actuators, to cloud services, and layers. Besides, devices and sensors the Internet of Things (IoT) architecture layers are distinguished to track the consistency of a system through protocols and gateways. Different architectures have been proposed by researchers and we can all agree that there is no single consensus on architecture for IoT. The most basic architecture is a three-layer architecture.

IoT architecture varies from solution to solution, based on the type of solution which we intend to build. IoT as a technology majorly consists of four main components, over which an architecture is framed.

- 1) Sensors
- 2) Devices
- 3) Gateway
- 4) Cloud



The 4 Stage IoT Solutions Architecture



Stage 1:- Sensors/actuators

- Sensors collect data from the environment or object under measurement and turn it into useful data. Think of the specialized structures in your cell phone that detect the directional pull of gravity and the phone's relative position to the —thing‖ we call the earth and convert it into data that your phone can use to orient the device.
- Actuators can also intervene to change the physical conditions that generate the data. An actuator might, for example, shut off a power supply, adjust an air flow valve, or move a robotic gripper in an assembly process.
- The sensing/actuating stage covers everything from legacy industrial devices to robotic camera systems, water level detectors, air quality sensors, accelerometers, and heart rate monitors. And the scope of the IoT is expanding rapidly, thanks in part to low-power wireless sensor network technologies and Power over Ethernet, which enable devices on a wired LAN to operate without the need for an A/C power source.

Stage 2:- The Internet gateway

- The data from the sensors starts in analog form. That data needs to be aggregated and converted into digital streams for further processing downstream. Data acquisition systems (DAS) perform these data aggregation and conversion functions. The DAS connects to the sensor network, aggregates outputs, and performs the analog-to-digital conversion. The Internet gateway receives the aggregated and digitized data and routes it over Wi-Fi, wired LANs, or the Internet, to Stage 3 systems for further processing. Stage 2 systems often sit in close proximity to the sensors and actuators.
- For example, a pump might contain a half-dozen sensors and actuators that feed data into a data aggregation device that also digitizes the data. This device might be physically attached to the pump. An adjacent gateway device or server would then process the data and forward it to the Stage 3 or Stage 4 systems. Intelligent gateways can build on additional, basic gateway functionality by adding such capabilities as analytics, malware protection, and data management services. These systems enable the analysis of data streams in real time.

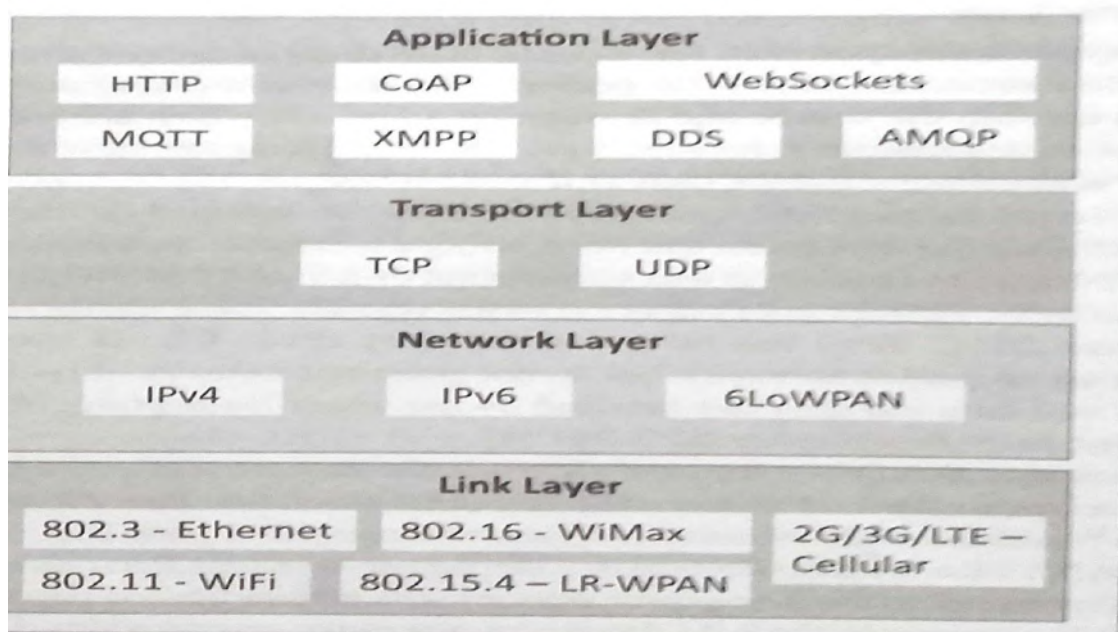
Stage 3:- Edge IT

- Once IoT data has been digitized and aggregated, it's ready to cross into the realm of IT. However, the data may require further processing before it enters the data center. This is where edge IT systems, which perform more analysis, come into play.
- Edge IT processing systems may be located in remote offices or other edge locations, but generally these sit in the facility or location where the sensors reside closer to the sensors, such as in a wiring closet.
- Because IoT data can easily eat up network bandwidth and swamp your data center resources, it's best to have systems at the edge capable of performing analytics as a way to lessen the burden on core IT infrastructure. You'd also face security concerns, storage issues, and delays processing the data. With a staged approach, you can preprocess the data, generate meaningful results, and pass only those on. For example, rather than passing on raw vibration data for the pumps, you could aggregate and convert the data, analyze it, and send only projections as to when each device will fail or need service.

Stage 4:- The data center and cloud

- Data that needs more in-depth processing, and where feedback doesn't have to be immediate, gets forwarded to physical data center or cloud-based systems.
- where more powerful IT systems can analyze, manage, and securely store the data. It takes longer to get results when you wait until data reaches Stage 4, but you can execute a more in-depth analysis, as well as combine your sensor data with data from other sources for deeper insights.
- Stage 4 processing may take place on-premises, in the cloud, or in a hybrid cloud system, but the type of processing executed in this stage remains the same, regardless of the platform.

IoT Protocols:



a) Link Layer :

Protocols determine how data is physically sent over the network's physical layer or medium. Local network connect to which host is attached. Hosts on the same link exchange data packets over the link layer using link layer protocols. Link layer determines how packets are coded and signaled by the h/w device over the medium to which the host is attached

Protocols:

- 802.3-Ethernet: IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 802.3 uses co-axial cable; 802.3i uses copper twisted pair connection; 802.3j uses fiber optic connection; 802.3ae uses Ethernet over fiber.
- 802.11-WiFi: IEEE802.11 is a collection of wireless LAN(WLAN) communication standards including extensive description of link layer. Eg: 802.11a operates in 5GHz band, 802.11b and 802.11g operates in 2.4GHz band, 802.11n operates in 2.4/5GHz band, 802.11ac operates in 5GHz band, 802.11ad operates in 60Ghzband.
- 802.16 - WiMax: IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMax provide data rates from 1.5 Mb/s to 1Gb/s.
- 802.15.4-LR-WPAN: IEEE802.15.4 is a collection of standards for low rate wireless personal area network(LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s.
- 2G/3G/4G-Mobile Communication: Data rates from 9.6kb/s(2G) to up to100Mb/s(4G).

B)Network/Internet Layer:

Responsible for sending IP datagrams from source n/w to destination n/w. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- IPv4: Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32 bit address. Allows total of 2^{32} addresses.
- IPv6: Internet Protocol version6 uses 128 bit address scheme and allows 2^{128} addresses.
- 6LOWPAN:(IPv6overLowpowerWirelessPersonalAreaNetwork)operates in 2.4 GHz frequency range and data transfer 250 kb/s.

C)Transport Layer:

Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control. Protocols:

- TCP: Transmission Control Protocol used by web browsers(along with HTTP and HTTPS), email(along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. Avoids n/w congestion and congestion collapse.

•UDP: User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery.

D)Application Layer:

Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.

Protocols:

•HTTP: Hyper Text Transfer Protocol that forms foundation of WWW. Follow request-response model Stateless protocol.

•CoAP: Constrained Application Protocol for machine-to-machine (M2M) applications with constrained devices, constrained environment and constrained n/w. Uses client- server architecture.

•WebSocket: allows full duplex communication over a single socket connection.

•MQTT: Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment.

•XMPP: Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.

•DDS: Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish-subscribe model.

•AMQP: Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model.

IoT Enabling Technologies

IoT is enabled by several technologies including Wireless Sensor Networks, Cloud Computing, Big Data Analytics, Embedded Systems, Security Protocols and architectures, Communication Protocols, Web Services, Mobile internet and semantic search engines.

1)Wireless Sensor Network(WSN):

Comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. Zig Bee is one of the most popular wireless technologies used by WSNs.

WSNs used in IoT systems are described as follows:

•Weather Monitoring System: in which nodes collect temp, humidity and other data, which is aggregated and analyzed.

•Indoor air quality monitoring systems: to collect data on the indoor air quality and concentration of various gases.

•Soil Moisture Monitoring Systems: to monitor soil moisture at various locations.

•Surveillance Systems: use WSNs for collecting surveillance data(motion data detection).

- Smart Grids : use WSNs for monitoring grids at various points.

- Structural Health Monitoring Systems: Use WSNs to monitor the health of structures(building, bridges) by collecting vibrations from sensor nodes deployed at various points in the structure.

2)Cloud Computing:

Services are offered to users in different forms.

- Infrastructure-as-a-service(IaaS):provides users the ability to provision computing and storage resources. These resources are provided to the users as a virtual machine instances and virtual storage.

- Platform-as-a-Service(PaaS): provides users the ability to develop and deploy application in cloud using the development tools, APIs, software libraries and services provided by the cloud service provider.

- Software-as-a-Service(SaaS): provides the user a complete software application or the user interface to the application itself.

3)Big Data Analytics:

Some examples of big data generated by IoT are

- Sensor data generated by IoT systems.

- Machine sensor data collected from sensors established in industrial and energy systems.

- Health and fitness data generated IoT devices.

- Data generated by IoT systems for location and tracking vehicles.

- Data generated by retail inventory monitoring systems.

4)Communication Protocols:

form the back-bone of IoT systems and enable network connectivity and coupling to applications.

- Allow devices to exchange data over network.

- Define the exchange formats, data encoding addressing schemes for device and routing of packets from source to destination.

- It includes sequence control, flow control and retransmission of lost packets.

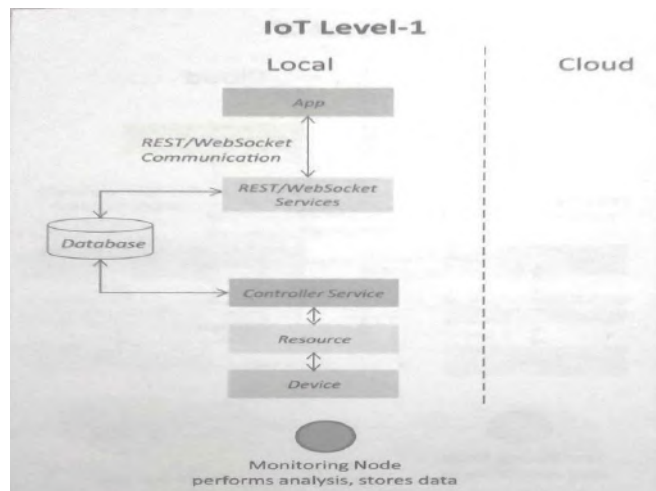
5)Embedded Systems:

is a computer system that has computer hardware and software embedded to perform specific tasks. Embedded System range from low cost miniaturized devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

IoT Levels and Deployment Templates

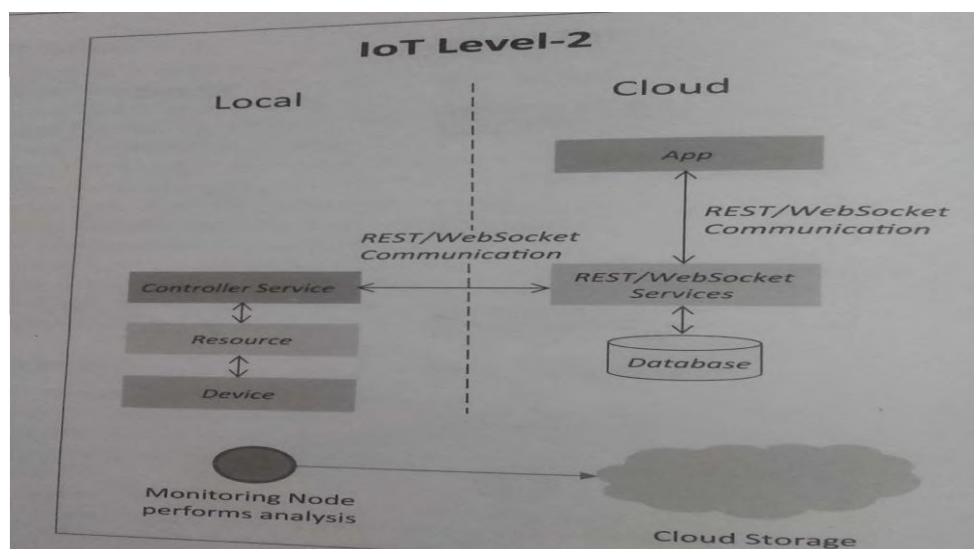
1)IoT Level1:

System has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application as shown in fig. Suitable for modeling low cost and low complexity solutions where the data involved is not big and analysis requirement are not computationally intensive. An e.g., of IoT Level1 is Home automation.



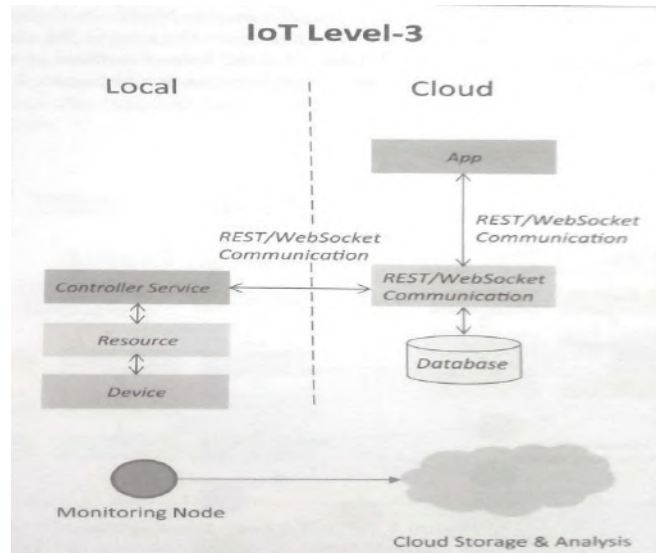
2)IoT Level2:

has a single node that performs sensing and/or actuating and local analysis as shown in fig. Data is stored in cloud and application is usually cloud based. Level2 IoT systems are suitable for solutions where data are involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself. An e.g., of Level2 IoT system for Smart Irrigation.



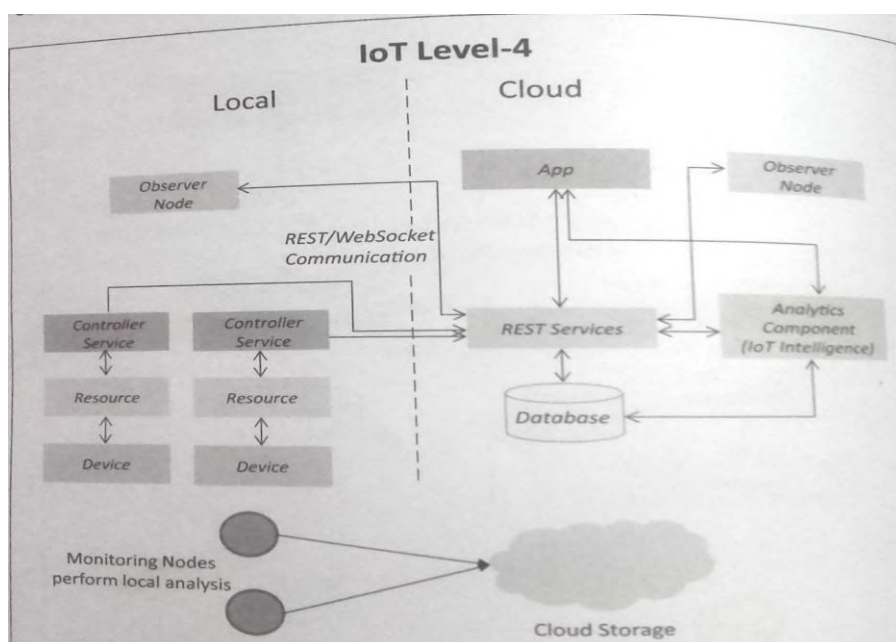
3)IoT Level3:

system has a single node. Data is stored and analyzed in the cloud application is cloud based as shown in fig. Level3 IoT systems are suitable for solutions where the data involved is big and analysis requirements are computationally intensive. An example of IoT level3 system for tracking package handling.



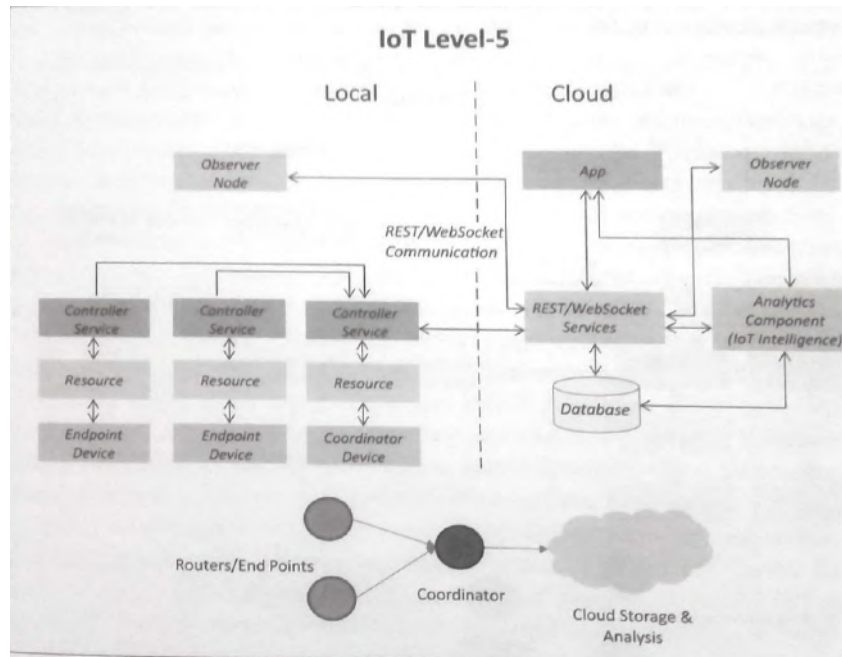
4)IoT Level4:

System has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud based as shown in fig. Level4 contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. An example of a Level4 IoT system for Noise Monitoring.



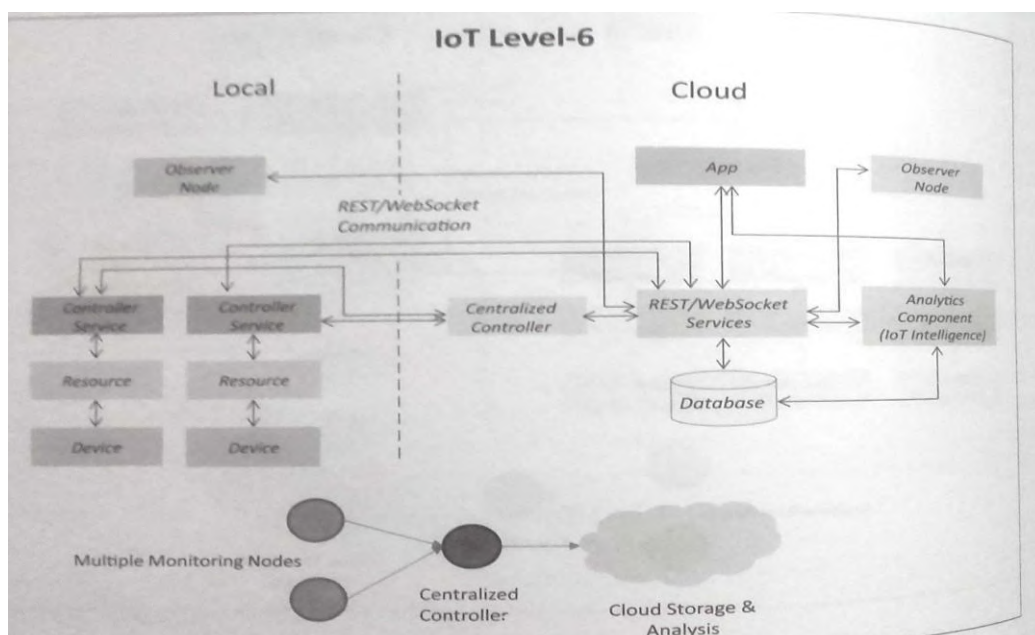
5)IoT Level5:

System has multiple end nodes and one coordinator node as shown in fig. The end nodes that perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud. Data is stored and analyzed in the cloud and application is cloud based. Level5 IoT systems are suitable for solution based on wireless sensor network, in which data involved is big and analysis requirements are computationally intensive. An example of Level5 system for Forest Fire Detection.



6)IoT Level6:

System has multiple independent end nodes that perform sensing and/or actuation and sensed data to the cloud. Data is stored in the cloud and application is cloud based as shown in fig. The analytics component analyses the data and stores the result in the cloud data base. The results are visualized with cloud based application. The centralized controller is aware of the status of all the end nodes and sends control commands to nodes. An example of a Level6 IoT system for Weather Monitoring System.



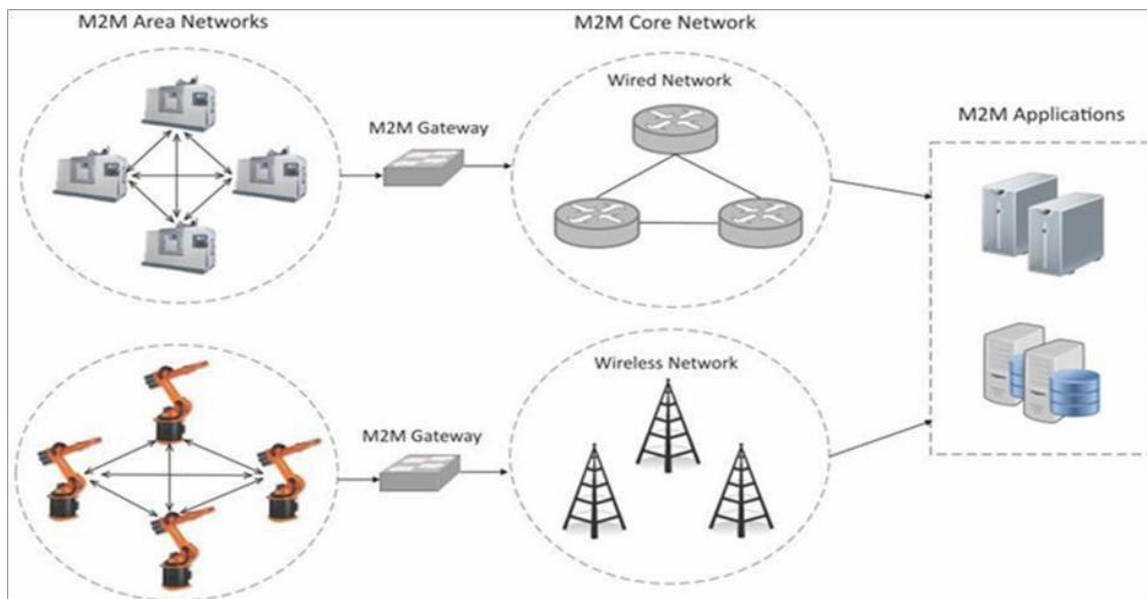
IoT and M2M

M2M:

Machine-to-Machine (M2M) refers to networking of machines(or devices) for the purpose of remote monitoring and control and data exchange.

- Term which is often synonymous with IoT is Machine-to-Machine (M2M).
- IoT and M2M are often used interchangeably.

Fig. Shows the end-to-end architecture of M2M systems comprises of M2M area networks, communication networks and application domain.



- An M2M area network comprises of machines(or M2M nodes) which have embedded network modules for sensing, actuation and communicating various communication protocols can be used for M2M LAN such as ZigBee, Bluetooth, M-bus, Wireless M-Bus etc., These protocols provide connectivity between M2M nodes within an M2M area network.

- The communication network provides connectivity to remote M2M area networks. The communication network provides connectivity to remote M2M area network. The communication network can use either wired or wireless network(IP based). While the M2M are networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based network. Since non-IP based protocols are used within M2M area network, the M2M nodes within one network cannot communicate with nodes in an external network.

- To enable the communication between remote M2M are network, M2M gateways are used

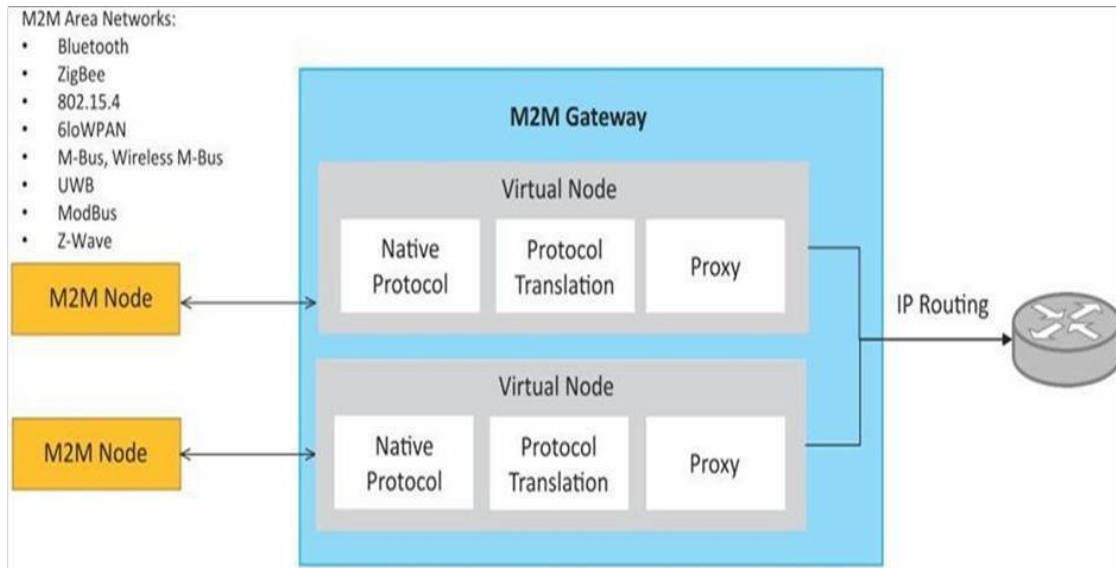


Fig. Shows a block diagram of an M2M gateway. The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable Ip-connectivity for M2M are networks. M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol(IP). With an M2M gateway, each mode in an M2M area network appears as a virtualized node for external M2M area networks.

Differences between IoT and M2M

1)Communication Protocols:

- Commonly uses M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus tec.,
- In IoT uses HTTP, CoAP, WebSocket , MQTT ,XMPP ,DDS ,AMQP etc.,

2)Machines in M2M Vs Things in IoT:

- Machines in M2M will be homogenous whereas Things in IoT will be heterogeneous.

3)Hardware Vs Software Emphasis:

- the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.

4)Data Collection &Analysis

- M2M data is collected in point solutions and often in on-premises storage infrastructure.
- The data in IoT is collected in the cloud (can be public, private or hybrid cloud).

5)Applications

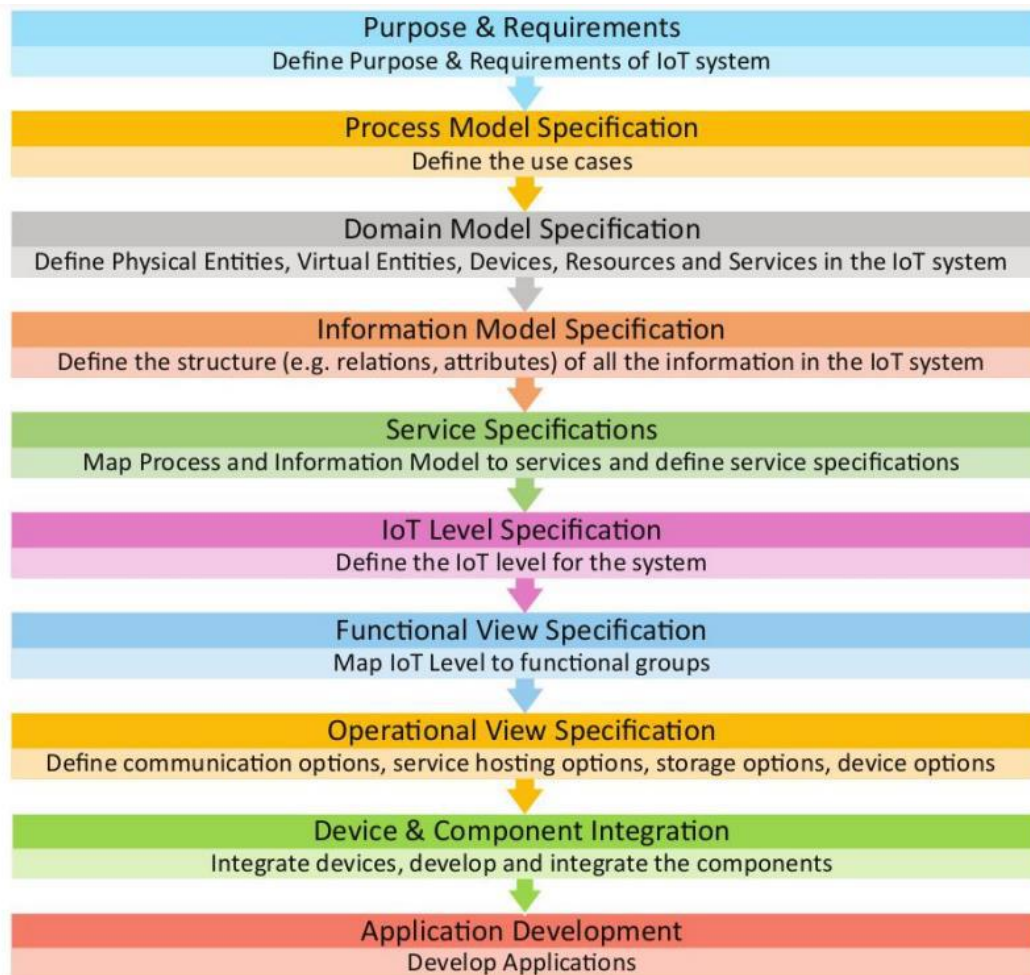
- M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premisis enterprise applications.
- IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

PARAMETERS	M2M	IOT
Abbreviation for	Machine to Machine	Internet of Things
Philosophy	M2M is Concept where two or more machines can communicate with each other and carry out certain functions without human intervention. Some degree of intelligence can be observed in M2M model.	IOT is an ecosystem of connected devices (via Internet) where the devices have ability to collect and transfer data over a network automatically without human intervention. IOT helps objects to interact with internal and/or external environment which in turn control the decision making.
Connection Type	Point to Point	Through IP Network using various Communication types
Communication protocols	Old proprietary protocols and communication techniques	Internet protocols used commonly
Value Chain	Linear	Multi-sided
Focus Area	For monitoring and control of 1 or few infrastructure/assets.	To address everyday needs of humans.
Sharing of collected data	Data collected is not shared with other applications	Data is shared with other applications (like weather forecasts, social media etc.) improve end user experience
Device dependency	Devices usually don't rely over Internet connection	Devices usually rely over Internet connection
Device in scope	Limited devices in scope	Large number of device sin scope
Scalability	Less scalable than IOT	More scalable due to cloud based architecture
Example	Remote monitoring, fleet control	Smart Cities, smart agriculture etc.
Business Type	B2B	B2B and B2C
Technology Integration	Vertical	Vertical and Horizontal
Open APIs	Not supported	Supported
Related terms	Sensors , Data and Information	End users, devices, wearables, Cloud and Big Data
https://ipwithease.com		

IoT Design Methodology:

Design methodology

- Designing IoT systems can be a complex and challenging task as these systems involve interactions between various components such as IoT devices and network resources, web services, analytics components, application and database servers.
- IoT system designers often tend to design IoT systems keeping specific products/services in mind.
- So that designs are tied to specific product/service choices made. But it make updating the system design to add new features or replacing a particular product/service choice for a component becomes very complex, and in many cases may require complete redesign of the system



Step 1: Purpose & Requirements Specification

- The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements, ...) are captured.

Step 2: Process Specification

- The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.

Step 3: Domain Model Specification

- The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

Step 4: Information Model Specification

- The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the

information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations.

Step 5: Service Specifications

- The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.

Step 6: IoT Level Specification

- The sixth step in the IoT design methodology is to define the IoT level for the system.

Step 7: Functional View Specification

- The seventh step in the IoT design methodology is to define the Functional View. The Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.

Step 8: Operational View Specification

- The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc

Step 9: Device & Component Integration • The ninth step in the IoT design methodology is the integration of the devices and components.

Step 10: Application Development • The final step in the IoT design methodology is to develop the IoT application.

Design Challenges

In IoT, several automated devices that are connected; communicate with each other through the Internet, and in a small network and a couple of devices, connectivity is seamless. But when IoT is deployed globally, and the number of devices and sensors connect and communicate, connectivity issues arise. And also, the Internet is not just a network; it includes a heterogeneous network having cell towers, slow connectivity, fast connectivity, proxy servers, firewalls, and different companies with different standards and technologies, all things that can disrupt connectivity. So, here the design of the entire IoT system holds utmost importance and is treated as an essential component of IoT as the overall success of the procedure depends on great design. So, here, we will analyze factors which challenge the design of IoT technology.

The various design challenges in IoT are as follows.

1. Power and Battery Life
2. Security and Compliances
3. Tests and Certifications
4. Emerging Standards
5. Designing for Everyone

1.Power and Battery Life:

At the trim level of IoT setup may be energy efficient. Still, when in a high level of IoT setup, things become more complex in high-performance devices where processor, displays, and communication interfaces require varying amounts of power there, power usage management becomes difficult. So minimal battery drain and long battery life are needed, and it must achieve low power consumption and energy efficiency.

2.Security and Compliances:

It is essential in the IoT Connectivity—monitoring and Maintaining security across connectivity change. Many IoT/M2M devices also come with several internet connectivity options. Many security pitfalls allow hackers to access or take control of the device and create connectivity issues.

3.Tests and Certifications:

Testing and certification are significant parts of designing any product related to IoT. Boards with wireless components—or any radiating component—must undergo strict certification processes before they can be permitted to be sold in different parts of the world. At the same time, these certification rules may vary from continent to continent or even from country to country. Aside from being a tedious process, this requires high costs, which the system designer should include when planning budgets for their circuit designs. For any Bluetooth module or chip present onboard, several certifications are required—from the FCC (for the US), CE (for European countries), IC (for Canada) and the list goes on.

4.Emerging Standards:

Despite IoT being depicted as a connected ecosystem where devices work in harmony, the reality is different. As with any untested frontier, plenty of companies are racing to become the dominant players in the emerging space; even when some product lines are completely walled off and are often designed to work exclusively with trusted providers, while, other systems are completely open where the biggest challenges for developers are usually coping with the potential interference between various equipment.

To help overcome these types of challenges, the "Open Connectivity Foundation" is currently developing an open standard to overcome previously mentioned issues of devices being produced independently of one another.

The biggest takeaway from the draft specs is that the full operability of the system needs to be engineered at all layers of the development stack i.e. vertical services, platform, and connectivity – to ensure a seamless user experience. The bulk of the OCF set of standards

leverages abstraction to streamline development workflows while guaranteeing that the data protocols are all dynamic and have layer agnostic capabilities. Therefore, here the five methods of the above-discussed standards include:

creating,
retrieving,
updating,
deleting, and
notifying.

There's also the IEEE standard which has an extensive and exclusive line of standards for the Internet of things

5.Designing for Everyone:

Perhaps one of the biggest challenges for any IoT system development or requirement is accommodating all the users' needs and being genuinely successful; it can't only target connected devices to a tech-savvy audience. Smart homes, involve leveraging an entire ecosystem of devices. Locks, thermostats, lighting, alarms, and more – are the foundations of living at home with more comfort.

There is also a lot of machine-to-machine (M2M) projects and systems such as intelligent power grids, general building automation, vehicle-to-vehicle communication, and wearable communication devices. It seems overwhelming, right? It doesn't have to be.

In the past, visuals were mostly the cornerstone of good successful user experience platforms; however, the future is now all about conversational UIs (UI that interact, know and serve the needs of the user). This opens an entirely brand new can of worms as user experience professionals now need to handle both linguistics and general visual design

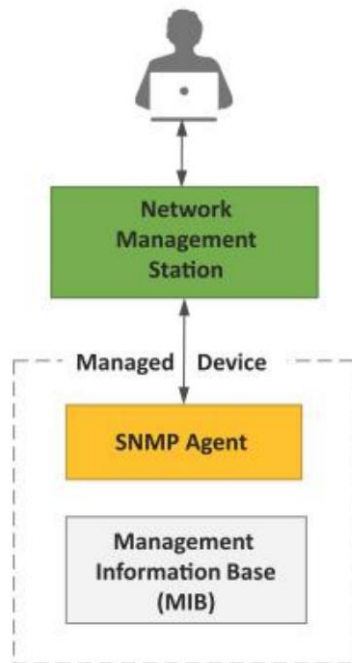
IoT System Management

Need for IoT Systems Management

- Automating Configuration
- Monitoring Operational & Statistical Data
- Improved Reliability
- System Wide Configurations
- Multiple System Configurations
- Retrieving & Reusing Configurations

Simple Network Management Protocol (SNMP)

- SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc. •
- SNMP component include
- Network Management Station (NMS)
- Managed Device
- Management Information Base (MIB)
- SNMP Agent that runs on the device



Limitations of SNMP

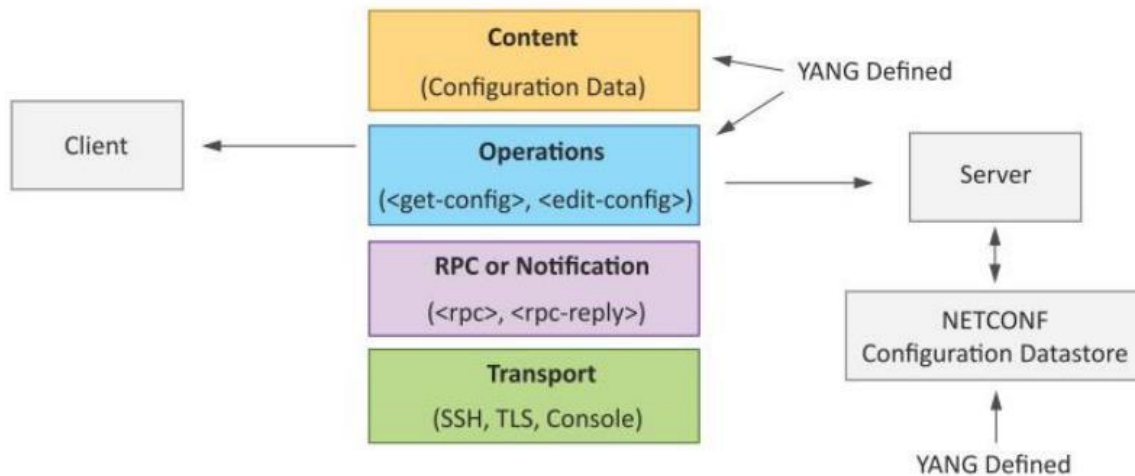
- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features.

Network Operator Requirements

- Ease of use
- Distinction between configuration and state data
- Fetch configuration and state data separately
- Configuration of the network as a whole
- Configuration transactions across devices
- Configuration deltas
- Dump and restore configurations
- Configuration database schemas
- Comparing configurations
- Role-based access control
- Consistency of access control lists:
- Multiple configuration sets

NETCONF

- Network Configuration Protocol (NETCONF) is a session-based network management protocol. NETCONF allows retrieving state or configuration data and manipulating configuration data on network devices



- NETCONF works on SSH transport protocol.
- Transport layer provides end-to-end connectivity and ensure reliable delivery of messages.
- NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages.
- The RPC layer provides mechanism for encoding of RPC calls and notifications.
- NETCONF provides various operations to retrieve and edit configuration data from network devices.
- The Content Layer consists of configuration and state data which is XML-encoded.
- The schema of the configuration and state data is defined in a data modeling language called YANG.
- NETCONF provides a clear separation of the configuration and state data.
- The configuration data resides within a NETCONF configuration datastore on the server

YANG

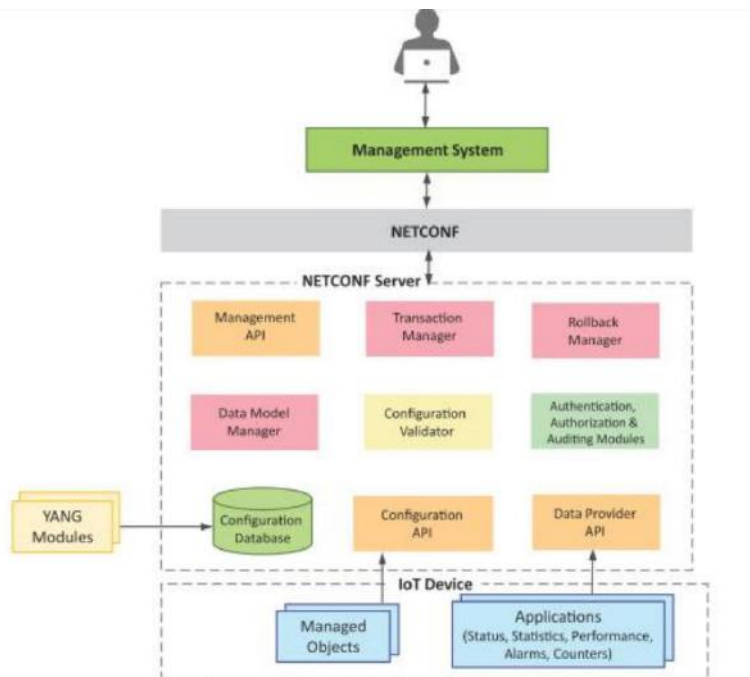
- YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol
- YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.
- YANG modules defines the data exchanged between the NETCONF client and server.
- A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure.
- The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs.
- Leaf nodes are organized using 'container' or 'list' constructs.
- A YANG module can import definitions from other modules.
- Constraints can be defined on the data nodes, e.g. allowed values.
- YANG can model both configuration data and state data using the 'config' statement.

IoT Systems Management with NETCONF-YANG

Management System

- Management API
- Transaction Manager
- Rollback Manager
- Data Model Manager

- Configuration Validator
- Configuration Database
- Configuration API
- Data Provider API



IoT Servers – Sensors.

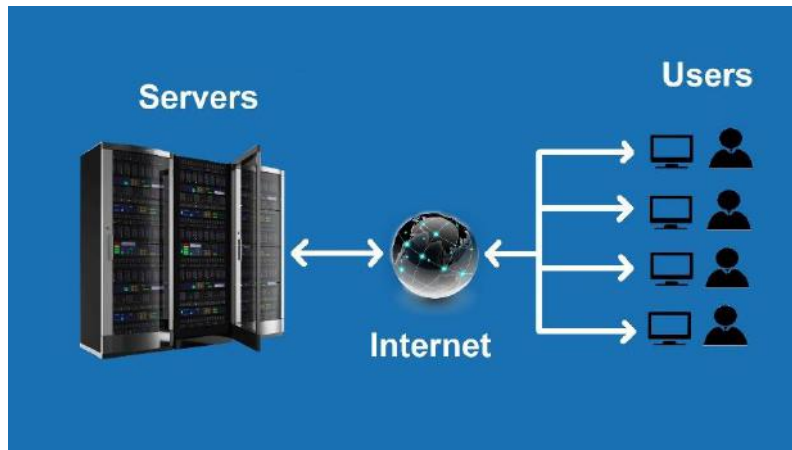
Server

A server is a computer, a device or a program that is dedicated to managing network resources. They are called that because they “serve” another computer, device, or program called “client” to which they provide functionality.

There are a number of categories of servers, including print servers, file servers, network servers and database servers. In theory, whenever computers share resources with client machines they are considered servers.

However, servers are often referred to as dedicated because they carry out hardly any other tasks apart from their server tasks.

Most frequently client–server systems are implemented by the request–response model., i.e., a client sends a request to the server. In this model server performs some action and sends a response back to the client, typically with a result or acknowledgement. Designating a computer as server-class hardware means that it is specialized for running servers on it. This implies that it is more powerful and reliable than standard personal computers. But large computing clusters may be composed of many relatively simple, replaceable server components.



Types of Servers and their applications:

1.Application server –

These servers hosts web apps (computer programs that run inside a web browser) allowing users in the network to run and use them preventing the installation a copy on their own computers. These servers need not be part of the World Wide Web. Their clients are computers with a web browser.

2.Catalog server –

These servers maintains an index or table of contents of information that can be found across a large distributed network. Distributed network may include computers, users, files shared on file servers, and web apps. Examples of catalog servers are Directory servers and name servers. Their clients are any computer program that needs to find something on the network. Example can be a Domain member attempting to log in, an email client looking for an email address, or a user looking for a file

3.Communications server –

These servers maintains an environment needed for one communication endpoint to find other endpoints and then communicates with them. These servers may or may not include a directory of communication endpoints and a presence detection service, depending on the openness and security parameters of the network. Their clients are communication endpoints.

4.Computing server –

These servers share vast amounts of computing resources which include CPU and random-access memory over a network. Any computer program that needs more CPU power and RAM than a personal computer can probably afford can use these types of servers. The client must be a networked computer to implement the client–server model which is necessity.

5.Database server –

These servers maintains and shares any form of database over a network. A database is a organized collections of data with predefined properties that may be displayed in a table. Clients of these servers are spreadsheets, accounting software, asset management software or virtually any computer program that consumes well-organized data, especially in large volumes.

6.Fax server –

These servers share one or more fax machines over a network which eliminates the hassle of physical access. Any fax sender or recipient are the clients of these servers.

7.File server –

Shares files and folders, storage space to hold files and folders, or both, over a network. Networked computers are the intended clients, even though local programs can be clients.

8.Game server –

These servers enable several computers or gaming devices to play multiplayer games. Personal computers or gaming consoles are their clients.

9.Mail server –

These servers make email communication possible in the same way as a post office makes snail mail communication possible. Clients of these servers are senders and recipients of email.

10.Print server –

These servers share one or more printers over a network which eliminates the hassle of physical access. Their clients are computers in need of printing something.

11.Proxy server –

This server acts as an intermediary between a client and a server accepting incoming traffic from the client and sending it to the server. Reasons to use a proxy server include content control and filtering, improving traffic performance, preventing unauthorized network access or simply routing the traffic over a large and complex network. Their clients are any networked computer.

12.Web server –

These servers host web pages. A web server is responsible for making the World Wide Web possible. Each website has one or more web servers. Their clients are computers with a web browser.

SENSORS

Sensors are devices that provide an output signal based on measuring an environmental phenomenon such as measuring temperature, humidity, pressure, altitude, ambient light, distance etc.

These devices are used to give quantitative and qualitative measurements of an environmental factor for the purposes of monitoring data to either record or take action.

For example, a temperature sensor can be used to monitor the ambient temperature. Based on the temperature sensor's measurement and output, heating or cooling can be enabled to bring the ambient temperature of the room to the optimal temperature

SENSORS IN IOT

Sensors have been around for decades being used in many different applications. But recently, with the advent of IoT sensors, today must encompass the ability to process its data, communicate with other sensors and platforms, forming a crucial part in the entire IoT ecosystem

CLASSIFICATION OF SENSORS:

Due to the sheer number of sensors available, for simplification sensors are divided into 5 core classifications depending on how they work

1. Active and Passive Sensors
2. Contact and Non-contact Sensors
3. Absolute and relative sensors
4. Analog and Digital Sensors
5. Miscellaneous Sensors

1. ACTIVE AND PASSIVE SENSORS

Active Sensors are sensors that require a dedicated external power supply in order to function. Examples include GPS and ultrasonic sensors.

Passive sensors on the other hand do not require any external supply and can receive enough electrical signal from the environment to function. Examples include thermal sensors, NFC tags, etc

2. CONTACT AND NON-CONTACT SENSORS

Contact sensors are sensors that require physical contact with the environmental stimulus the sensor is measuring. Examples include touch sensors, temperature sensors, strain gauges, etc

Non-contact sensors are sensors that do not require direct contact with the environmental stimulus it measures. Examples include optical sensors, magnetic sensors, infrared thermometers, etc

3. ABSOLUTE AND RELATIVE SENSORS

Absolute sensors, as its name suggests, provide an absolute reading of the stimulus. For example, thermistors always give out the absolute temperature readings

Relative sensors provide measurements relative to something that is either fixed or variable. Thermocouple is an example of a relative sensor, where the temperature difference is measured as opposed to direct measurement

4. ANALOG AND DIGITAL SENSORS

Analog sensors produce a continuous output signal proportional to the measurement. Examples include thermometers, LDR, pressure sensors etc

Digital sensors are sensors that convert the measurement into a digital signal. Examples include Inertial Measurement Units, ultrasonic sensors, etc

5. MISCELLANEOUS SENSORS

There are many more types of sensors that may not necessarily fit into the above categories. Those sensors will be classified as miscellaneous sensors and these include biological, chemical, radioactive sensors, etc

TOP 8 IOT SENSORS

1. Temperature Sensors

Temperature sensors measure the amount of heat energy in a source, allowing them to detect temperature changes and convert these changes to data. Machinery used in manufacturing often requires environmental and device temperatures to be at specific levels. Similarly, within agriculture, soil temperature is a key factor for crop growth.

2. Humidity Sensors

These types of sensors measure the amount of water vapor in the atmosphere of air or other gases. Humidity sensors are commonly found in heating, vents and air conditioning (HVAC) systems in both industrial and residential domains. They can be found in many other areas including hospitals, and meteorology stations to report and predict weather.

3. Pressure Sensors

A pressure sensor senses changes in gases and liquids. When the pressure changes, the sensor detects these changes, and communicates them to connected systems. Common use cases include leak testing which can be a result of decay. Pressure sensors are also useful in the manufacturing of water systems as it is easy to detect fluctuations or drops in pressure.

4. Proximity Sensors

Proximity sensors are used for non-contact detection of objects near the sensor. These types of sensors often emit electromagnetic fields or beams of radiation such as infrared. Proximity sensors have some interesting use cases. In retail, a proximity sensor can detect the motion between a customer and a product in which he or she is interested. The user can be notified of any discounts or special offers of products located near the sensor. Proximity sensors are also used in the parking lots of malls, stadiums and airports to indicate parking availability. They can also be used on the assembly lines of chemical, food and many other types of industries.

5. Level Sensors

Level sensors are used to detect the level of substances including liquids, powders and granular materials. Many industries including oil manufacturing, water treatment and beverage and food manufacturing factories use level sensors. Waste management systems provide a common use case as level sensors can detect the level of waste in a garbage can or dumpster.

6. Accelerometers

Accelerometers detect an object's acceleration i.e. the rate of change of the object's velocity with respect to time. Accelerometers can also detect changes to gravity. Use cases for accelerometers include smart pedometers and monitoring driving fleets. They can also be used as anti-theft protection alerting the system if an object that should be stationary is moved.

7. Gyroscope

Gyroscope sensors measure the angular rate or velocity, often defined as a measurement of speed and rotation around an axis. Use cases include automotive, such as car navigation and electronic stability control (anti-skid) systems. Additional use cases include motion sensing for video games, and camera-shake detection systems.

8. Gas Sensors

These types of sensors monitor and detect changes in air quality, including the presence of toxic, combustible or hazardous gasses. Industries using gas sensors include mining, oil and gas, chemical research and manufacturing. A common consumer use case is the familiar carbon dioxide detectors used in many homes.

9. Infrared Sensors

These types of sensors sense characteristics in their surroundings by either emitting or detecting infrared radiation. They can also measure the heat emitted by objects. Infrared sensors are used in a variety of different IoT projects including healthcare as they simplify the monitoring of blood flow and blood pressure. Televisions use infrared sensors to interpret the signals sent from a remote control. Another interesting application is that of art historians using infrared sensors to see hidden layers in paintings to help determine whether a work of art is original or fake or has been altered by a restoration process.

10. Optical Sensors

Optical sensors convert rays of light into electrical signals. There are many applications and use cases for optical sensors. In the auto industry, vehicles use optical sensors to recognize signs, obstacles, and other things that a driver would notice when driving or parking. Optical sensors play a big role in the development of driverless cars. Optical sensors are very common in smart phones. For example, ambient light sensors can extend battery life. Optical sensors are also used in the biomedical field including breath analysis and heart-rate monitors.

UNIT 5

UNIT- V Arduino in IoT

Basics of Arduino: Introduction to Arduino – Types of Arduino – Arduino Toolchain – Arduino Programming Structure – Sketches – Pins -Input/Output From Pins Using Sketches – Introduction to Arduino Shields – Integration of Sensors and Actuators with Arduino-Connecting LEDs with Arduino, Connecting LCD with Arduino – Tinkercad Arduino simulation.

Introduction to Arduino

Arduino is a software as well as hardware platform that helps in making electronic projects. It is an open source platform and has a variety of controllers and microprocessors. There are various types of Arduino boards used for various purposes.

The Arduino is a single circuit board, which consists of different interfaces or parts. The board consists of the set of digital and analog pins that are used to connect various devices and components, which we want to use for the functioning of the electronic devices. Most of the Arduino consists of 14 digital I/O pins. The analog pins in Arduino are mostly useful for fine-grained control. The pins in the Arduino board are arranged in a specific pattern. The other devices on the Arduino board are USB port, small components (voltage regulator or oscillator), microcontroller, power connector, etc.. Arduino also simplifies the process of working with microcontrollers.

Features of Arduino:

- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

History of Arduino:

The project began in the Interaction Design Institute in Ivrea, Italy. Under the supervision of Casey Reas and Massimo Banzi, the Hernando Bar in 2003 created the Wiring (a development platform). It was considered as the master thesis project at IDII. The Wiring platform includes the PCB (Printed Circuit Board). The PCB is operated with the ATmega168 Microcontroller. The ATmega168 Microcontroller was an IDE. It was based on the library and processing functions, which are used to easily program the microcontroller.

In 2005, Massimo Banzi, David Cuartielles, David Mellis, and another IDII student supported the ATmega168 to the Wiring platform. They further named the project as Arduino.

The project of Arduino was started in 2005 for students in Ivrea, Italy. It aimed to provide an easy and low-cost method for hobbyists and professionals to interact with the environment using the actuators and the sensors. The beginner devices were simple motion detectors, robots, and thermostats.

In mid-2011, the estimated production of Arduino commercially was 300,000. In 2013, the Arduino boards in use were about 700,000.

Around April 2017, Massimo Banzi introduced the foundation of Arduino as the "new beginning for Arduino". In July 2017, Musto continued to pull many Open Source licenses and the code from the websites of the Arduino. In October 2017, Arduino introduced its collaboration with the ARM Holdings. The Arduino continues to work with architectures and technology vendors.

Hardware and Software

Arduino boards are generally based on microcontrollers from Atmel Corporation like 8, 16 or 32 bit AVR architecture based microcontrollers. The important feature of the Arduino boards is the standard connectors. Using these connectors, we can connect the Arduino board to other devices like LEDs or add-on modules called Shields.

The Arduino boards also consists of on board voltage regulator and crystal oscillator. They also consist of USB to serial adapter using which the Arduino board can be programmed using USB connection.

In order to program the Arduino board, we need to use IDE provided by Arduino. The Arduino IDE is based on Processing programming language and supports C and C++.

Types of Arduino

Arduino makes several different boards, each with different capabilities. In addition, part of being open source hardware means that others can modify and produce derivatives of Arduino boards that provide even more form factors and functionality.

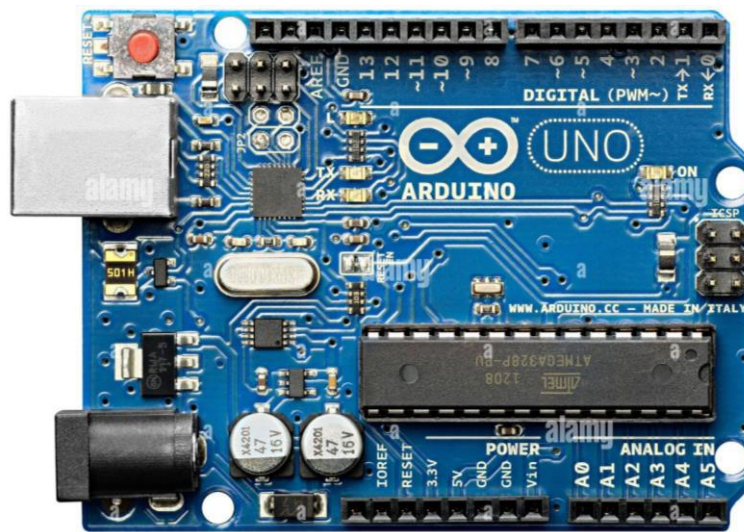
The Arduino boards are provided as open source that helps the user to build their projects and instruments according to their need. This electronic platform contains microcontrollers,

connections, LEDs and many more. There are various types of Arduino boards present in the market that includes Arduino UNO, Red Board, LilyPad Arduino, Arduino Mega, Arduino Leonardo. All these Arduino boards are different in specifications, features and uses and are used in different type of electronics project.

Entry-Level Arduino Boards

Arduino's entry-level category contains the microcontroller boards that most DIYers choose to use for their projects, as they offer straightforward features and come with heaps of documentation. This also means that they can lack the niche features that come with enhanced and IoT Arduino boards.

1.Arduino UNO



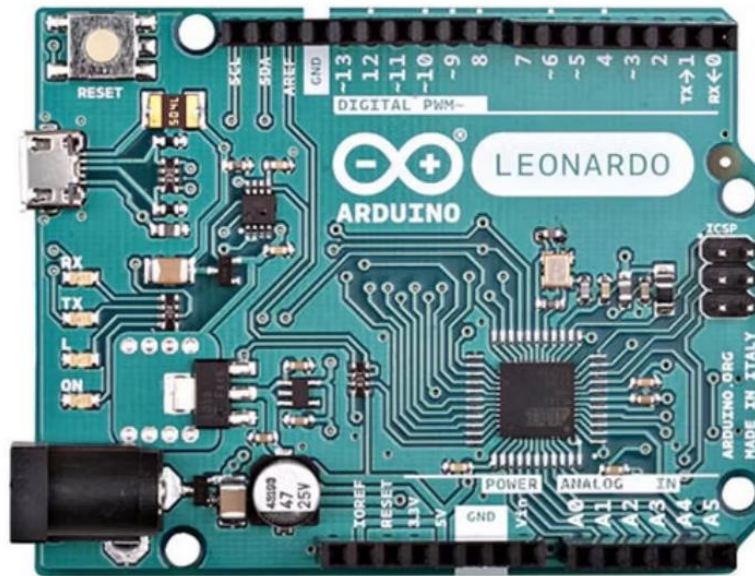
The development of Arduino UNO board is considered as new compared to other Arduino boards. This board comes up with numerous features that helps the user to use this in their project. The Arduino UNO uses the Atmega16U2 microcontroller that helps to increase the transfer rate and contain large memory compared to other boards. No extra devices are needed for the Arduino UNO board like joystick, mouse, keyboard and many more. The Arduino UNO contain SCL and SDA pins and also have two additional pins fit near to RESET pin.

The board contains 14 digital input pins and output pins in which 6 pins are used as PWM, 6 pins as analog inputs, USB connection, reset button and one power jack. The Arduino UNO board can be attached to computer system buy USB port and also get power supply to board from computer system. The Arduino UNO contains flash memory of size 32 KB that is used to the data in it. The other feature of the Arduino UNO is compatibility with other shield and can be combined with other Arduino products.

Basic Specs:

- Microcontroller: ATmega328P
- Memory: 2kB SRAM, 32kB flash, and 1kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: Replaceable chip

2.Arduino Leonardo



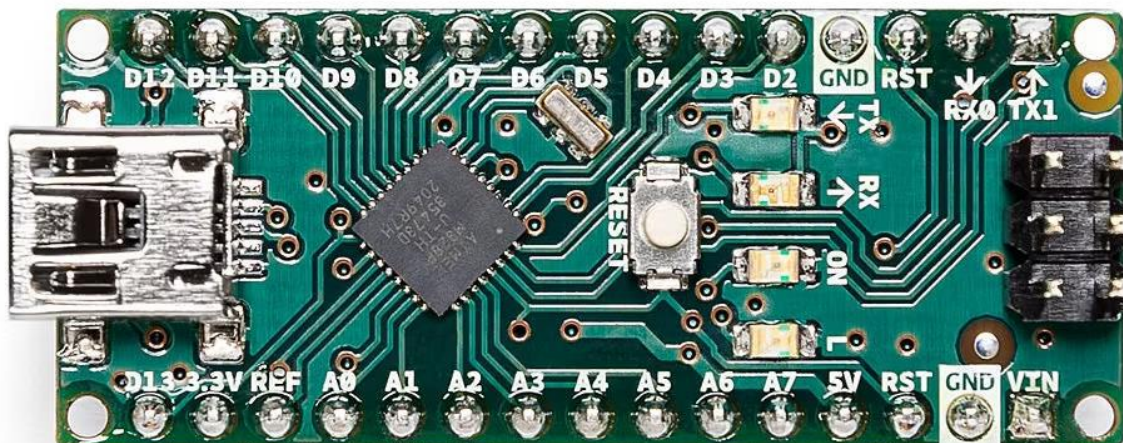
The Arduino Leonardo comes with essentially the same specifications as an Uno R3, only it features a micro-USB connector, has 20 digital and 17 analog pins, and has the ability to be used as a human interface device thanks to the ATmega32U4 chip that powers it. This means that your Leonardo can be used as a keyboard or mouse with a USB cable attached.

This type of Arduino is ideal for simple projects that need to interact with machines like computers, providing a huge range of different ideas to try for yourself.

Basic Specs:

- Microcontroller: ATmega32U4
- Memory: 2.5kB SRAM, 32kB flash, and 1kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: HID connectivity

3.Arduino Nano / Nano Every



The Arduino Nano and Nano Every are the smallest microcontroller boards offered by the company. Both boards feature the same pin layout, with 14 digital pins and 8 analog pins, though the Nano Every has a beefier microcontroller chip and improved program memory. These boards both come with pre-soldered headers that make them ideal for use with breadboards, but they lack the power jack that comes on larger boards.

Their breadboard compatibility makes these small boards great for those who like to make circuits that change all the time, like school teachers and prototype makers.

Basic Specs:

- Microcontroller: ATmega32U4 (Nano); ATmega4809 (Nano Every)
- Memory: 2kB SRAM, 32kB flash, and 1kB EEPROM (Nano); 6kB SRAM, 48kB flash, and 256B EEPROM (Nano Every)
- Communication: UART, IC2, and SPI
- Special Features: Breadboard-compatible and extremely small

4.Arduino Micro



The Arduino Micro boasts very similar features to the Leonardo, only the board is much smaller and only features 12 analog pins alongside its 20 digital ones. At only 18mm wide and 48mm long, this board is one of the smallest Arduino has ever made, making it ideal for creating a keyboard, mouse, and or other HID devices that need to be tiny.

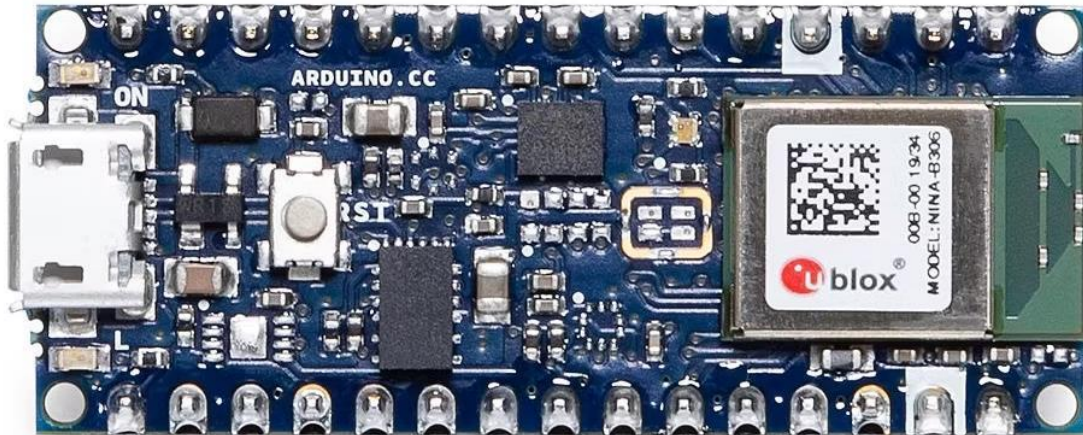
Basic Specs:

- Microcontroller: ATmega32U4
- Memory: 2.5kB SRAM, 32kB flash, and 1kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: HID connectivity and small form-factor

Enhanced Arduino Boards

Enhanced Arduino boards offer the features required to complete projects with greater complexity, while also providing improved performance for DIYers looking to push the limits.

1.Arduino Nano 33 BLE / Nano 33 BLE Sense



The Nano 33 BLE / Nano 33 BLE Sense is designed as an improved version of the Arduino Nano / Nano Every, featuring the same pin layout to make it nice and easy for DIYers. Both boards have a 32-bit Arm Cortex-M4 CPU running at 64MHz built into their nRF52840 chips, with 1MB of flash memory and 256kB of SRAM, making these boards incredibly powerful despite their small size.

They only come with 14 digital pins, but are packed with a host of sensors that don't come with regular Nanos. This sensor array includes an accelerometer, a gyroscope, and a magnetometer with 3-axis resolution, and the board comes with Bluetooth Low Energy (BLE) that makes it easy to transmit the data it collects.

Alongside all of these great features, the Nano 33 BLE Sense is also able to run edge computing applications using machine learning models from TensorFlow Lite.

Basic Specs:

- Microcontroller: nRF52840
- Memory: 256kB SRAM and 1MB flash
- Communication: UART, IC2, and SPI
- Special Features: Sensors, Bluetooth, and AI (Sense only)

2.Arduino MKR Zero

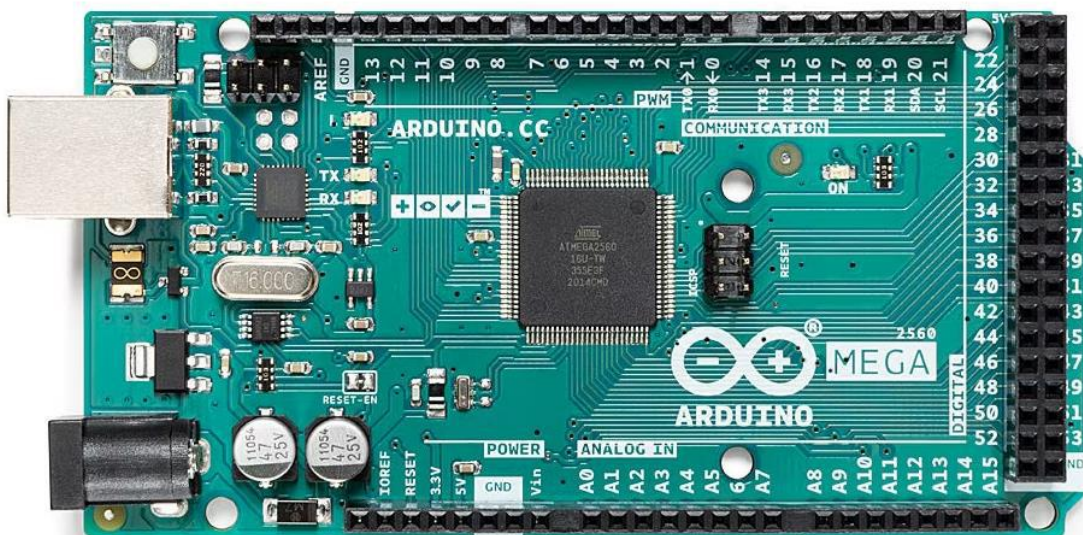


The Arduino MKR Zero is designed with music-making and other complex projects in mind, featuring a powerful Arm Cortex-M0 32-bit SAMD21 processor, native battery support, and a built-in microSD card reader. The board comes with 8 digital pins, 7 analog input pins, and 1 analog output pin. Thanks to the expandable storage that comes with this board, it is an excellent option for those working with a lot of code and a range of hardware components.

Basic Specs:

- Microcontroller: Arm Cortex-M0 32-bit SAMD21
- Memory: 32kB SRAM and 256kB flash
- Communication: UART, IC2, and SPI
- Special Features: Built-in battery connector, microSD card reader, powerful hardware

3.Arduino Mega 2560 R3



The Arduino Mega 2560 is similar to an Arduino Uno, only it features 54 digital pins, 16 analog pins, and 4 serial ports, along with being much larger and more powerful than the standard Uno. This board is great for DIYers in need of more pins, memory, or processing power without sacrificing the functionality that comes with regular Arduinos.

Basic Specs:

- Microcontroller: ATmega2560
- Memory: 8kB SRAM, 256kB flash, and 4kB EEPROM
- Communication: UART, IC2, and SPI
- Special Features: Large form-factor and serial ports

Arduino Toolchain

A toolchain is a set of software development tools used simultaneously to complete complex software development tasks or to deliver a software product. Each tool in the chain is itself a piece of software that serves a different function and is optimized to work together with other tools in the chain. Example programs in a toolchain are an assembler, linker and debugger.

Companies can customize toolchains to suit their software needs. Some software vendors also offer their own toolchains. One of the main benefits of using a toolchain instead of a disparate set of tools is the consistent programming environment throughout the development process and the smooth transition between tools as development progresses.

Example:

In programming, a toolchain is just a series of programming tools that work together in order to complete a task.

If we were cooking something, like carrot soup, the toolchain might include a carrot peeler, a produce knife, a pot, some water, and a stove.



Together, these tools help you create the desired output, carrot soup.

When we're developing programs for Arduino, we'll also be using a toolchain. This all happens behind the scenes. On our end, we literally only have to press a button for the code to get to the board, but it wouldn't happen without the toolchain.

The Arduino has a similar toolchain. When we start writing our code, and we become the author. We do this in the Arduino IDE, which is akin to a text editor. We also write the code in a programming language called C++, with a file type extension of .ino

The code that we write is called human readable code since it's meant for us to read and, hopefully, understand. However, the Arduino hardware doesn't understand C++ code.

It needs what's called machine code. In order to generate that machine code, we use a software tool called a compiler. Remember that Verify button in the Arduino IDE that looks like the checkmark?



When you press this button, it compiles the code using compiler software called AVR-GCC. This compiler software does a bunch of stuff. The main thing it does it rearrange some code and check for errors. This is like the professional editor at the publishing company. The compiler also translates the human readable code into machine code. There are some in-between file types and other fuzzy things that go on, but the final output of the compiler is a machine language saved as a .hex file.

In order to get this .hex file loaded onto our Arduino's integrated circuit, we need to press the Upload button.



This begins another piece of software called AVRDUDE, which sends the file to the integrated circuit.

Normally, we would have to use some external hardware to load that circuit. However, in Arduino's infinite wisdom, they loaded a small piece of software onto the integrated circuit that automatically comes with an Arduino board.

That piece of software is called a bootloader. It works with ARVDUDE to take the outputted .hex file and put it onto the flash memory of that Arduino's integrated circuit using only the USB cable.

Again, all we had to do was press the Upload button. This whole process happens behind the scenes.

Now the process is complete

What is the purpose of a toolchain?

The purpose of a toolchain is to have a group of linked software tools that are optimized for a specific programming process. The output generated by one tool in the chain is used by the next tool as the input.

By integrating tools together, the toolchain considers the various tool dependencies and, as a result, streamlines the software development process. For example, a programming language such as C++ may require language-specific tools, like a debugger or compiler. A development team that knows C++ can integrate one of these tools into their chain to avoid bottlenecks in the development process.

What should a toolchain include?

Toolchains may vary depending on the team using them and the product being delivered. However, basic toolchains often include the following components:

Assembler -- converts assembly language into code.

Linker -- links a set of program files together into a single program.

Debugger -- used to test and debug programs.

Compiler -- used to generate executable code from the source code of a program.

Runtime libraries -- used to interface with an operating system (OS). It enables the program to reference external functions and resources. An API is an example of this.

Developers will often write a script that links these tools together and automates some of the process. Complex software products will likely require more tools than this, but this basic assortment is generally involved in any software toolchain in some form, regardless of the product being delivered.

Some coding languages and platforms offer their own toolchains and enable customization within the integrated development environment platform. Apple's Xcode is one example of this.

Benefits of using toolchains

The main benefit of using a software toolchain is that it streamlines the development lifecycle. To a degree, the software development process can be automated, which is especially useful for DevOps implementations. In DevOps implementations, teams often employ a continuous delivery strategy and need all tools working simultaneously to be as efficient as possible.

Additionally, using a pre-built toolchain from a third party allows development teams to bypass the step of building a toolchain from scratch, which can be difficult and time consuming. Developers can use these preexisting toolchains and then customize them to the specific use case.

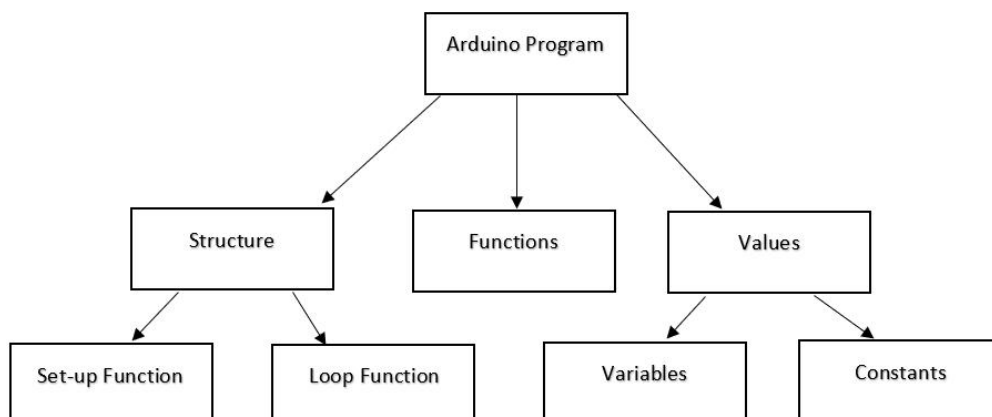
Using toolchains can also help expedite handoffs between teams that use different software tools by increasing visibility, security and productivity.

Arduino Programming Structure

Sketch – The first new terminology is the Arduino program called “sketch”, Different languages can be used to execute different functions by using electronic machines. These languages aid in giving commands to the machine. There are a lot of different programming languages, and each language has its own commands, syntax, and structure of writing a program. The language used for Arduino is C++. The Arduino program structure is briefly explained in this discourse.

Arduino Programing Overview

The Arduino program is divided into three main parts that are structure, values, and functions.



1. Structure

Structure consist of two main functions –

Setup() function:

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup() function will only run once, after each powerup or reset of the Arduino board.

Eg:

```
1 int buttonPin = 3;
2
3 void setup() {
4   // put your setup code here, to executed once:
5   Serial.begin(9600);
6   pinMode(buttonPin, INPUT);
7   Serial.println("This is setup code");
8 }
9
10 void loop() {
11   // ...
12 }
```

Output:



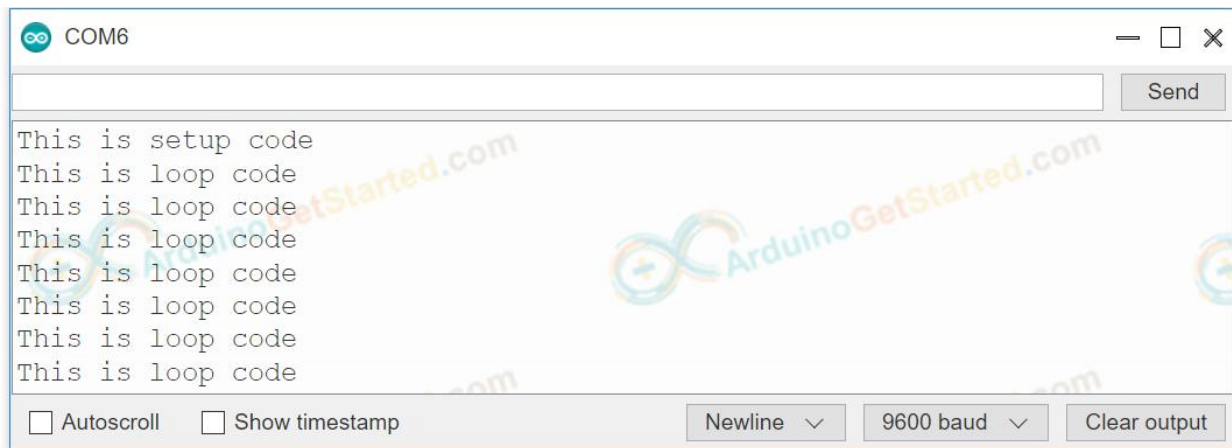
Loop() function:

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Eg:

```
1 void setup() {
2   // put your setup code here, to executed once:
3   Serial.begin(9600);
4   Serial.println("This is setup code");
5 }
6
7 void loop() {
8   // put your main code here, to run repeatedly:
9   Serial.println("This is loop code");
10  delay(1000);
11 }
```

Output:



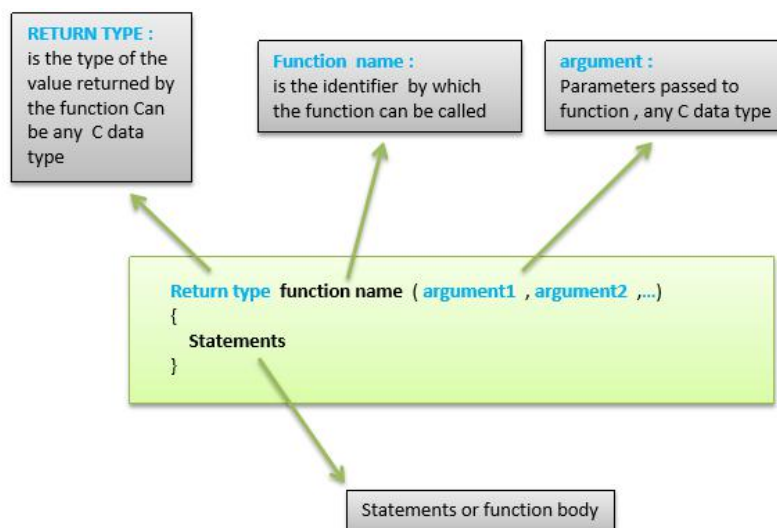
```
COM6
This is setup code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
This is loop code
```

2.Functions

Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Standardizing code fragments into functions has several advantages –

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought about and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it modular, and using functions often makes the code more readable.



Function Declaration

A function is declared outside any other functions, above or below the loop function.

We can declare the function in two different ways –

The first way is just writing the part of the function called **a function prototype** above the loop function, which consists of –

- Function return type
- Function name
- Function argument type, no need to write the argument name

Function prototype must be followed by a semicolon (;).

The following example shows the demonstration of the function declaration using the first method.

Example

```
int sum_func (int x, int y) // function declaration {
  int z = 0;
  z = x+y;
  return z; // return the value
}

void setup () {
  Statements // group of statements
}

Void loop () {
  int result = 0;
  result = Sum_func (5,6); // function call
}
```

3.Values

A value is a piece of data that can be manipulated by the program. This could be a number, a character, a string of text, or a logical value (such as "true" or "false"). In Arduino programs, values are often stored in variables, which are named containers that can hold a single value. These values can then be used in calculations, compared to other values, or used to control the behavior of the program. For example, a variable named "x" might be used to store the current temperature reading from a sensor, which could then be used to control a heating or cooling system.

Values Are 2 Types

1.Variable

2.Constant

1.Variable:

The variables are defined as the place to store the data and values. It consists of a name, value, and type.

The variables can belong to any data type such as int, float, char, etc.

Consider the below example:

```
int pin = 8;
```

2.Constant:

The constants in Arduino are defined as the predefined expressions. It makes the code easy to read. The constants in Arduino are defined as:

Logical level Constants:

The logical level constants are true or false.

The value of true and false are defined as 1 and 0. Any non-zero integer is determined as true in terms of Boolean language. The true and false constants are type in lowercase rather than uppercase (such as HIGH, LOW, etc.).

Pin level Constants:

The digital pins can take two value HIGH or LOW.

In Arduino, the pin is configured as INPUT or OUTPUT using the pinMode() function. The pin is further made HIGH or LOW using the digitalWrite() function.

Input/Output From Pins Using Sketches

In the Arduino platform, a sketch is a program written in the Arduino programming language. Sketches are used to control the behavior of the Arduino board, including reading input from and writing output to the board's pins.

Input and output can be performed using the digital and analog pins on the Arduino board. Digital pins can be used to read digital inputs, such as the state of a button, or to write digital outputs, such as turning on an LED. Analog pins, on the other hand, can be used to read analog inputs, such as the output of a sensor, or to write analog outputs, such as dimming an LED.

To read and write to the pins, the sketch must use the appropriate Arduino functions. For example, the *digitalRead()* function can be used to read the state of a digital input pin, while the *analogWrite()* function can be used to write an analog output to a pin. These functions are typically used within the setup() and loop() functions of the sketch, which are special functions that are called automatically when the sketch is executed.

Here is an example of a simple sketch that reads a digital input from pin 2 and writes the state of that input to an LED connected to pin 13:

```
void setup() {  
  // Set the LED pin as an output  
  pinMode(13, OUTPUT);  
}  
  
void loop() {  
  // Read the state of the input pin  
  int inputState = digitalRead(2);  
  
  // Write the state of the input to the LED  
  digitalWrite(13, inputState);  
}
```

In this sketch, the `setup()` function is used to configure pin 13 as an output, and the `loop()` function is used to read the input from pin 2 and write it to the LED on pin 13. This sketch will continue to run indefinitely, reading the input and updating the LED every time the `loop()` function is called.

Introduction to Arduino Shields

Arduino shields:

Arduino shields are the boards, which are plugged over the Arduino board to expand its functionalities. There are different varieties of shields used for various tasks, such as Arduino motor shields, Arduino communication shields, etc.

Shield is defined as the hardware device that can be mounted over the board to increase the capabilities of the projects. It also makes our work easy. For example, Ethernet shields are used to connect the Arduino board to the Internet.

The pin position of the shields is similar to the Arduino boards. We can also connect the modules and sensors to the shields with the help of the connection cable.

Arduino motor shields help us to control the motors with the Arduino board.

Why do we need Shields?

The advantages of using Arduino shields are listed below:

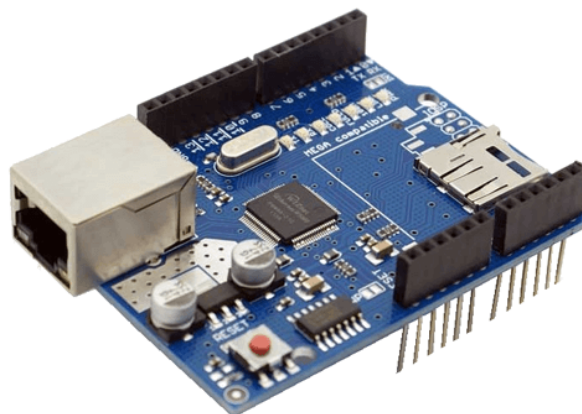
- It adds new functionalities to the Arduino projects.
- The shields can be attached and detached easily from the Arduino board. It does not require any complex wiring.
- It is easy to connect the shields by mounting them over the Arduino board.
- The hardware components on the shields can be easily implemented.

Types of Shields

The popular Arduino shields are listed below:

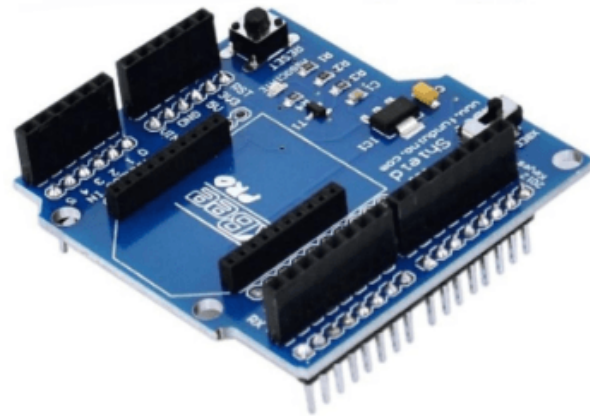
- **Ethernet shield**
- **Xbee Shield**
- **Proto shield**
- **Relay shield**
- **Motor shield**
- **LCD shield**
- **Bluetooth shield**
- **Capacitive Touchpad Shield**

1.Ethernet shield



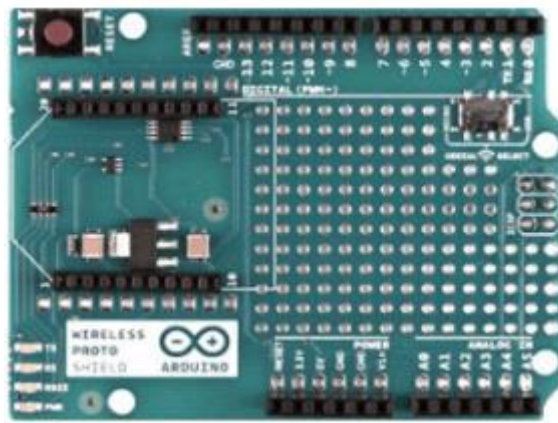
- The Ethernet shields are used to connect the Arduino board to the Internet. We need to mount the shield on the top of the specified Arduino board.
- The USB port will play the usual role to upload sketches on the board.
- The latest version of Ethernet shields consists of a micro SD card slot. The micro SD card slot can be interfaced with the help of the SD card library.
- We can also connect another shield on the top of the Ethernet shield. It means that we can also mount two shields on the top of the Arduino board.

2.Xbee Shield



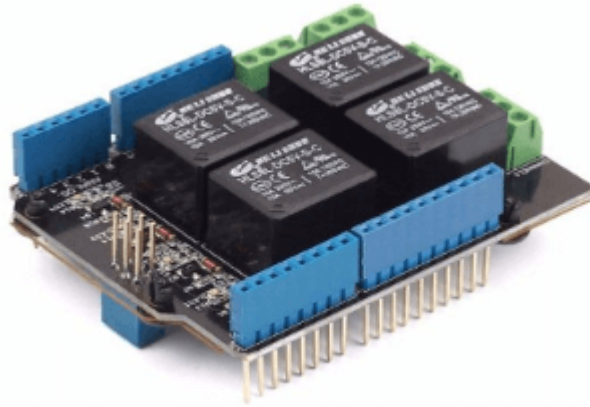
- We can communicate wirelessly with the Arduino board by using the Xbee Shield with Zigbee.
- It reduces the hassle of the cable, which makes Xbee a wireless communication model.
- The Xbee wireless module allows us to communicate outdoor upto 300 feet and indoor upto 100 feet.
- It can also be used with different models of Xbee.

3.Proto shield



- Proto shields are designed for custom circuits.
- We can solder electronic circuits directly on the shield.
- The shield consists of two LED pads, two power lines, and SPI signal pads.
- The IOREF (Input Output voltage REference) and GND (Ground) are the two power lines on the board.
- We can also solder the SMD (Surface Mount Device) ICs on the prototyping area. A maximum of 24 pins can be integrated onto the SMD area.

4. Relay shield



- The Arduino digital I/O pins cannot bear the high current due to its voltage and current limits. The relay shield is used to overcome such situation. It provides a solution for controlling the devices carrying high current and voltage.
- The shield consists of four relays and four LED indicators.
- It also provides NO/NC interfaces and a shield form factor for the simple connection to the Arduino board.
- The LED indicators depicts the ON/OFF condition of each relay.
- The relay used in the structure is of high quality.
- The NO (Normally Open), NC (Normally Closed), and COM pins are present on each relay.
- The applications of the Relay shield include remote control, etc.

5. Motor shield



- The motor shield helps us to control the motor using the Arduino board.
- It controls the direction and working speed of the motor. We can power the motor shield either by the external power supply through the input terminal or directly by the Arduino.
- We can also measure the absorption current of each motor with the help of the motor shield.
- The motor shield is based on the L298 chip that can drive a step motor or two DC motors. L298 chip is a full bridge IC. It also consists of the heat sinker, which increases the performance of the motor shield.

- It can drive inductive loads, such as solenoids, etc.
- The operating voltage is from 5V to 12V.
- The applications of the motor shield are intelligent vehicles, micro-robots, etc.

6.LCD shield



- The keypad of LCD (Liquid Crystal Display) shield includes five buttons called as up, down, left, right, and select.
- There are 6 push buttons present on the shield that can be used as a custom menu control panel.
- It consists of the 1602 white characters, which are displayed on the blue backlight LCD.
- The LED present on the board indicates the power ON.
- The five keys present on the board helps us to make the selection on menus and from board to our project.
- The LCD shield is popularly designed for the classic boards such as Duemilanove, UNO, etc.

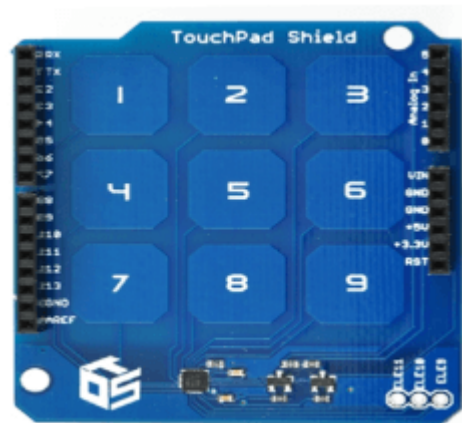
7.Bluetooth shield



- The Bluetooth shield can be used as a wireless module for transparent serial communication.
- It includes a serial Bluetooth module. D0 and D1 are the serial hardware ports in the Bluetooth shield, which can be used to communicate with the two serial ports (from D0 to D7) of the Arduino board.

- We can install Groves through the two serial ports of the Bluetooth shield called a Grove connector. One Grove connector is digital, while the other is analog.
- The communication distance of the Bluetooth shield is upto 10m at home without any obstacle in between.

8.Capacitive Touchpad shield



- It has a touchpad interface that allows to integrate the Arduino board with the touch shield.
- The Capacitive touchpad shield consists of 12 sensitive touch buttons, which includes 3 electrode connections and 9 capacitive touch pads.
- The board can work with the logic level of 3.3V or 5V.
- We can establish a connection to the Arduino project by touching the shield.

Integration of Sensors and Actuators with Arduino

Sensor

A sensor is a device that detects and responds to some type of input from the physical environment. The input can be light, heat, motion, moisture, pressure or any number of other environmental phenomena. The output is generally a signal that is converted to a human-readable display at the sensor location or transmitted electronically over a network for reading or further processing.

Sensors play a pivotal role in the internet of things (IoT). They make it possible to create an ecosystem for collecting and processing data about a specific environment so it can be monitored, managed and controlled more easily and efficiently. IoT sensors are used in homes, out in the field, in automobiles, on airplanes, in industrial settings and in other environments.

Actuators

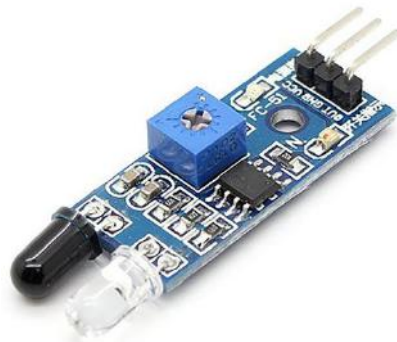
An actuator is a device that actuates something. In simple words, it produces motion. It helps other devices move and interact with their surrounding environment. The actuator takes in energy and uses it to help the device move. Servo motors and DC motors are types of electric actuators.

Integration of IR Sensor with Arduino

WHAT IS IR SENSOR?

An IR sensor is a device that measures the Infrared radiation in its surroundings and gives an electric signal as an output. An IR sensor can measure the heat of an object as well as can detect the motion of the objects.

IR technology is used in our day-to-day life and also in industries for different purposes. For example, TVs use an IR sensor to understand the signals which are transmitted from a remote control. The main benefits of IR sensors are low power usage, their simple design & their convenient features. IR signals are not noticeable by the human eye. The IR radiation in the electromagnetic spectrum can be found in the regions of the visible & microwave. Usually, the wavelengths of these waves range from $0.7 \mu\text{m}$ to $1000\mu\text{m}$. The IR spectrum can be divided into three regions near-infrared, mid, and far-infrared. The near IR region's wavelength ranges from $0.75 - 3\mu\text{m}$, the mid-infrared region's wavelength ranges from 3 to $6\mu\text{m}$ & the far IR region's infrared radiation's wavelength is higher than $6\mu\text{m}$.



As the Infrared Radiation is invisible to our eyes, it can be detected by the Infrared Sensor. An IR sensor is a photodiode that is sensitive to IR light. When IR light falls on the photodiode, the resistances and the output voltages will change in proportion to the magnitude of the IR light received.

TYPES OF IR SENSORS

1. Active IR Sensor.
2. Passive IR Sensor.

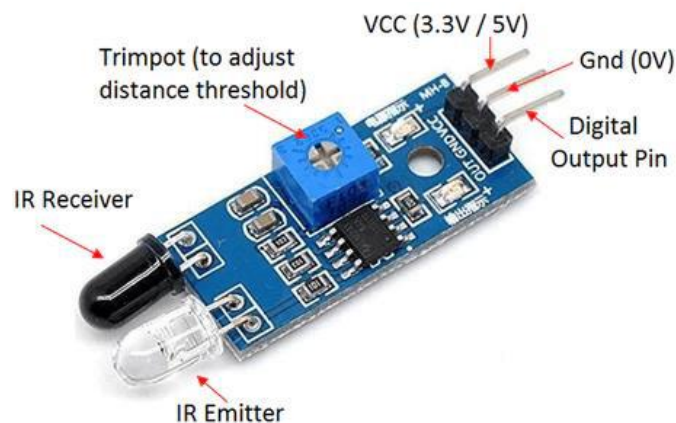
1. ACTIVE IR SENSOR

Active infrared sensors both emit and detect infrared radiation. Active IR sensors have two parts: a light-emitting diode (LED) as an Infrared Radiation transmitter and a Photodiode as an Infrared Radiation receiver. When an object comes close to the sensor, the infrared light from the LED reflects off of the object and is detected by the receiver. Active IR sensors act as proximity sensors, and they are commonly used in obstacle detection systems (such as in robots).

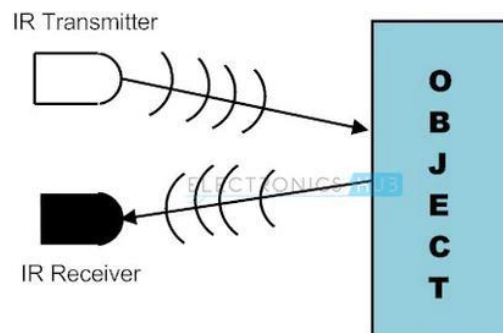
2. PASSIVE IR SENSOR

Passive infrared (PIR) sensors only detect infrared radiation and do not emit it from an LED. PIR sensors are most commonly used in motion-based detection, such as in-home security systems. When a moving object that generates infrared radiation enters the sensing range of the detector, the difference in IR levels between the two pyroelectric elements is measured. The sensor then sends an electronic signal to an embedded computer, which in turn triggers an alarm.

HOW DOES AN IR SENSOR MODULE WORK:



As we know till now that the active IR Sensors emit Infrared waves and as-well-as detect the Infrared ways. The IR sensor module exactly works on the same phenomenon. The IR Sensor module contains an Infrared LED and an infrared photodiode.

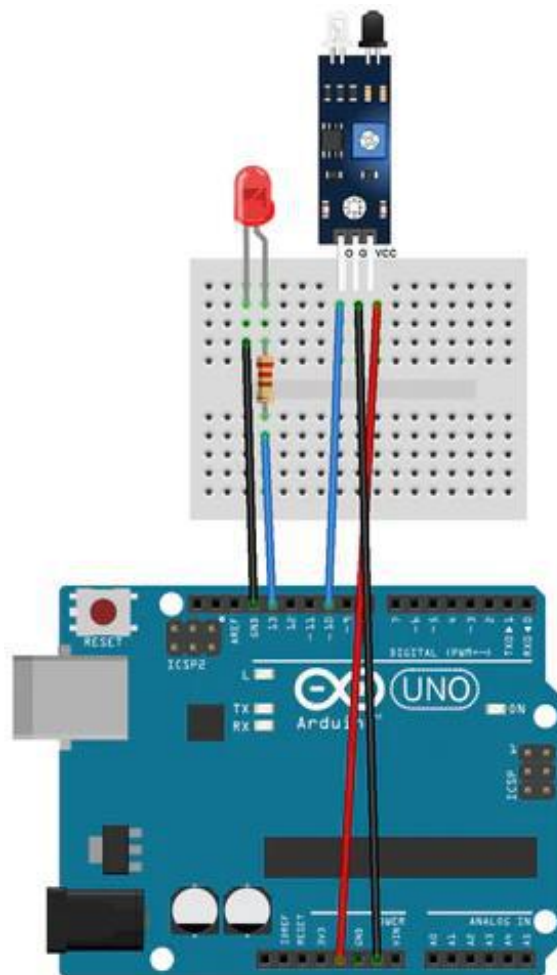


INTERFACING IR SENSOR MODULE WITH ARDUINO

HARDWARE REQUIRED

We need the following components:

1. Arduino
2. Breadboard
3. Jumper wires
4. IR Sensor module
5. LED
6. 220-ohm resistor



ARDUINO CODE FOR INTERFACING IR SENSOR MODULE WITH ARDUINO

```
int LEDpin = 13;
int obstaclePin = 10;
int hasObstacle = LOW; // LOW MEANS NO OBSTACLE

void setup() {
  pinMode(LEDpin, OUTPUT);
  pinMode(obstaclePin, INPUT);
  Serial.begin(9600);
}

void loop() {
  hasObstacle = digitalRead(obstaclePin);
  if (hasObstacle == HIGH) {
    Serial.println("Stop something is ahead!!");
    digitalWrite(LEDpin, HIGH);
  }
  else {
    Serial.println("Path is clear");
    digitalWrite(LEDpin, LOW);
  }
  delay(200);
}
```

UPLOADING THE CODE TO THE ARDUINO BOARD:

Step 1: Connect the Arduino board with the computer using a USB cable.

Step 2: Next type the code mentioned above.

Step 3: Select the right board and port.

Step 4: Upload the code to the Arduino.

Step 5: The LED will glow when any obstacle is detected by the IR sensor.

Integration of Actuator with Arduino

Integrating servo motor with Arduino Uno. Servo Motor is an electrical linear or rotary actuator which enables precise control of linear or angular position, acceleration or velocity. Usually servo motor is a simple motor controlled by a servo mechanism, which consists of a positional sensor and a control circuit. Servo motor is commonly used in applications like robotics, testing automation, manufacturing automation, CNC machine etc. The main characteristics of servo motor are low speed, medium torque, and accurate position.

Components Required

- Arduino Uno
- Servo Motor
- Jumper Wires

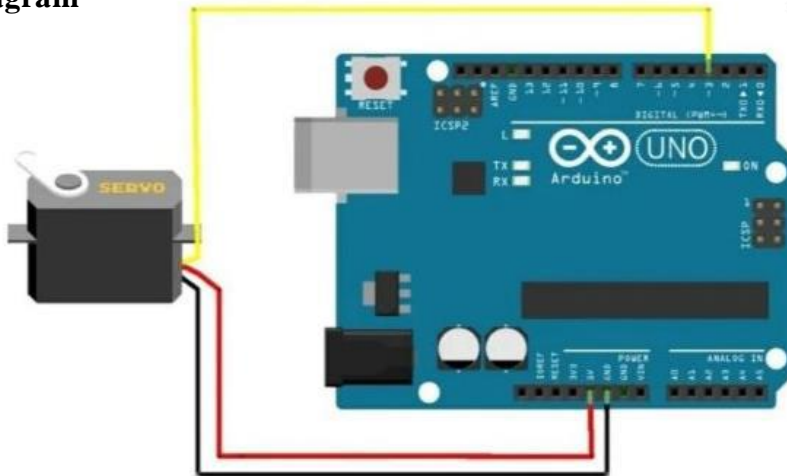
Hobby Servo Motor



Specifications

- Servo Motor consists of three pins: VCC, GROUND, and PWM signal pin.
- The VCC and GND is for providing power supply, the PWM input pin is used for controlling the position of the shaft by varying pulse width.
- Minimum Rotational Angle : 0 degrees
- Maximum Rotational Angle : 180 degrees
- Operating Voltage : +5V
- Torque : 2.5 kg/cm
- Operating Speed : 0.1 s/60 degree
- DC Supply Voltage : 4.8V to 6V

Circuit Diagram



Description

- The VCC pin (Red Color) of the Servo Motor is connected to the 5V output of the Arduino Board.
- The GND pin (Brown Color) of the Servo Motor is connected to the GND pin of the Arduino Board.
- The PWM input pin (Yellow Color) of the Servo Motor is connected to the PWM output pin of the Arduino board.

Working

The hobby servo motor which we use here consists of 4 different parts as below.

- Simple DC Motor
- Potentiometer
- Control Circuit Board
- Gear Assembly

Potentiometer is connected to output shaft of the servo motor helps to detect position. Based on the potentiometer value, the control circuit will rotate the motor to position the shaft to a certain angle. The position of the shaft can be controlled by varying the pulse width provided in the PWM input pin. Gear assembly is used to improve the torque of the motor by reducing speed.

Program

```
#include <Servo.h>

Servo servo;

int angle = 10;

void setup() {
    servo.attach(3);
    servo.write(angle);
}

void loop()
{
    for(angle = 10; angle < 180; angle++)
    {
        servo.write(angle);
        delay(15);
    }
    for(angle = 180; angle > 10; angle--)
    {
        servo.write(angle);
        delay(15);
    }
}
```

Code Explanation

- The position of the shaft is kept at 10 degrees by default and the Servo PWM input is connected to the 3rd pin of the Arduino Uno.
- In the first for loop, motor shaft is rotated from 10 degrees to 180 degrees step by step with a time delay of 15 milliseconds.
- Once it reaches 180 degree, it is programmed to rotate back to 10 degree step by step with a delay of 15 milliseconds in the second for loop.

Functioning

Wire the circuit properly as per the circuit diagram and upload the program. You can see that the servo motor is rotating as per the program.

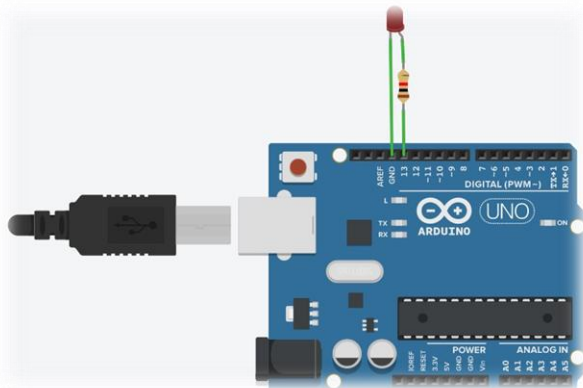
Connecting LEDs with Arduino

Light-emitting diode (LED) is a widely used standard source of light in electrical equipment. It has a wide range of applications ranging from your mobile phone to large advertising billboards. They mostly find applications in devices that show the time and display different types of data. In this Program we will Connect The LED and Blink it. Blinking means turning ON an LED for a few seconds, then OFF for seconds, repeatedly.

REQUIRED COMPONENTS

S.N.	COMPONENTS	QUANTITY
1.	Arduino	1
2.	LED	1
3.	Resistor (280 ohm)	1
4.	Connecting Wires	Few

CIRCUIT DIAGRAM



CIRCUIT EXPLANATION

Here, The anode (+) and cathode (-) terminals of LED are connected to pin 13 and ground (GND) of Arduino Uno respectively and a resistor of 280 ohm is placed between the LED anode terminal and pin number 13 which help us to limit the current and prevent LED from burning.

STEP 2: CODE

```
void setup()
{
pinMode(13, OUTPUT);
```

```
}  
void loop() {  
digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000);  
}
```

Functioning

Wire the circuit properly as per the circuit diagram and upload the program. You can see that the the Blinking an LED as per the program.

Connecting LCD with Arduino

Display devices are used to visually display the information we are working with. LCD (Liquid Crystal Display) screens are one of many display devices that makers use. We have libraries to control specific LCD functions which make it ridiculously easy to get up and running with LCDs.

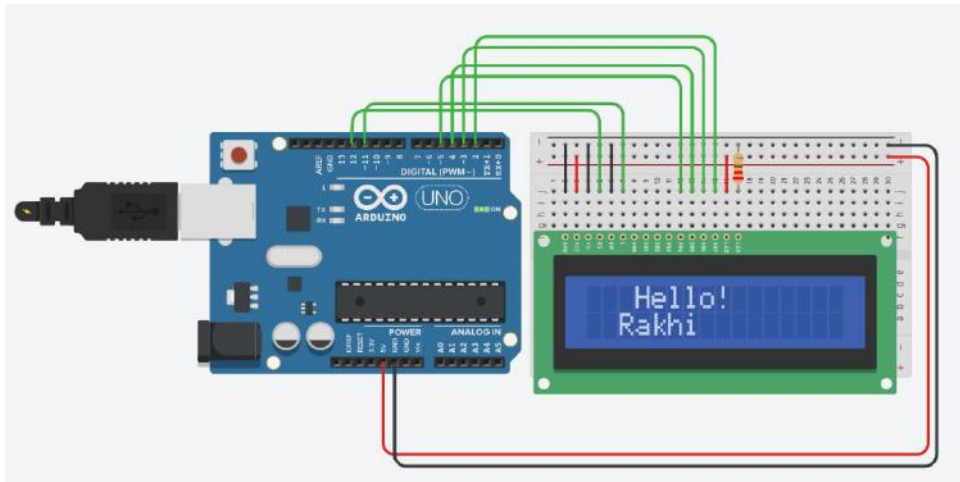
We are going to connect an LCD screen to our Arduino and make it print some stuff to the display, grab whatever you need and get started

- **Solderless Breadboard**
- **Arduino Board**
- **USB Cable**
- **Potentiometer (10k ohm)**
- **LCD Screen (16x2)**
- **Jumper Cables**

What is the LCD and how do it work?

A liquid crystal display is essentially a liquid crystal sandwiched between 2 pieces of glass, the liquid crystal reacts depending on current applied. Our LCD is a white on black, 16 by 2 character LCD that we will use to display symbols. Graphical LCDs also exist, but today we are just focusing on the character variety.

Each character is off by default and is a matrix of small dots of liquid crystal. These dots make up the numbers and letters that we display on screens. The actual coding that goes into making these characters appear is quite complicated, luckily for you, the people over at Arduino.cc have made a library of functions for the LCD that we can import into our sketch, The LCD also has a backlight and contrast control options. The backlight will shine through the pieces of glass the screen is made of to display the characters on the screen. The contrast will control how dark (or light) the characters appear. We will use a variable resistor to control contrast, and we will set the backlight to being on.



Circuit Explanation:

This circuit is the basic circuit for Arduino projects with LCD Display. If you know how to wire up a LCD Display with an Arduino then you can do any Arduino Projects including the LCD Display and the Arduino. You can use any Arduino flavor for this circuit, but just remember to put a current limiting resistor between the the pin 16 of the LCD Display to GND. Also if you want you can connect a 10k ohm Potentiometer to pin 3 of the LCD Display, remember you need to connect pin 1 of the POT to VCC - 5V and pin 2 of the POT to GND.

Code:

```
#include<LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  lcd.begin(16, 2);
}

void loop()
{
  lcd.setCursor(0,0);
  lcd.print(" Hello!");
  lcd.setCursor(2,1);
  lcd.print("Rakhi");
}
```

Functioning

Wire the circuit properly as per the circuit diagram and upload the program. You can see that the LCD Display With Some text as per the program.

Tinkercad Arduino simulation.

Arduino Simulator

The Arduino simulator is a **virtual portrayal of the circuits of Arduino** in the real world. We can create many projects using a simulator without the need for any hardware.

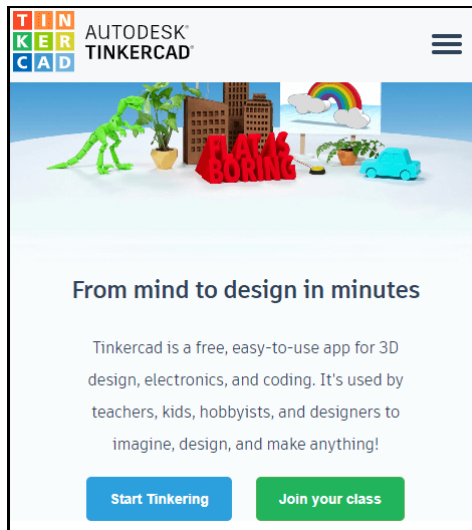
The Simulator helps beginner and professional designers to learn, program, and create their projects without wasting time on collecting hardware equipments.

we are using the **Autodesk Tinkercad Simulator**.

The steps to access the TINKERCAD are listed below:

1. Open the official website of tinkercad. : <https://www.tinkercad.com/>

A window will appear, as shown below:



2. Click on the three horizontal lines present on the upper right corner.
3. Click on the '**Sign in**' option, if you have an account in Autodesk. Otherwise, click on the '**JOIN NOW**' option if you don't have an account, as shown below:

The **SIGN IN** window will appear as:

We can select any sign-in method. Specify the username and password.

We already have an account in Autodesk, so we will sign-in directly with the username and password.

The **JOIN** window will appear as:

4. Now, a window will appear,

5. Click on the 'Create new circuit' option to start designing the Arduino circuit, as shown above.

6. The 'Circuits' option will also show the previous circuits created by user. The design option is used for creating the 3D design, which is of no use in Arduino

Autodesk Tinkercad Simulator

It is also a simulator that is used to design virtual circuits.

Features of Tinkercad

The features of Tinkercad are listed below:

- **Glow and move circuit assembly.** It means we can use the components of a circuit according to the project requirement. Glow here signifies the glowing of LED.
- **Integrated product design.** It means the electronic components used in the circuitry are real.
- **Arduino Programming.** We can directly write the program or code in the editor of the simulator.
- We can also consider some **ready-made examples** provided by the tinkercad for better understanding.
- **Realtime simulation.** We can prototype our designs within the browser before implementing them in real-time.

Advantages of using Simulator

There are various advantages of using simulator, which are listed below:

- It saves money, because there is no need to buy hardware equipments to make a project.
- The task to create and learn Arduino is easy for beginners.
- We need not to worry about the damage of board and related equipments.
- No messy wire structure required.
- It helps students to eliminate their mistakes and errors using simulator.
- It supports line to line debugging, and helps to find out the errors easily.
- We can learn the code and build projects anywhere with our computer and internet connection.
- We can also share our design with others.

SSSS