UNIT I **Introduction to IOT**

Introduction - Definition and Characteristics of IoT - Physical design - IoT Protocols - Logical design - IoT communication models, IoT Communication APIs - Enabling technologies - Wireless Sensor Networks, Cloud Computing, Embedded Systems, IoT Levels and Templates - Domain specific IoTs -home, city ,Environment, energy, Agriculture and Industry

1.1 Introduction

- The Internet of Things represents the whole way from collecting data, processing it, taking an action corresponding to the signification of this data to storing everything in the cloud. All this is made possible by the internet
- The Internet of things has become a very widely spread concept in the last few years. The reason for this is mainly the need to computerize and control most of the surrounding objects and have access to data in real time.
- Example: Parking sensors, about phones which can check the weather and so on

1.1.1 Definition & Characteristics of IoT 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

i) **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, user‘s context or sensed environment.

Eg: The surveillance system comprising of a number of surveillance cameras. The surveillance camera can adapt modes based on whether it is day or night. The surveillance system is adapting itself based on context and changing conditions.

ii) **Self Configuring:**

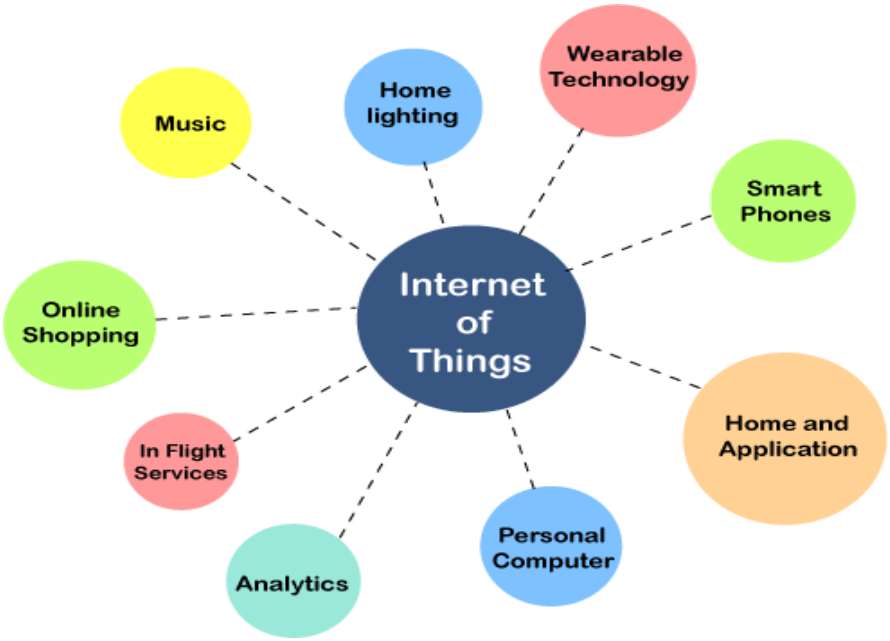
IOT devices have self configuring capability,allowing a large number of devices to work together to provide certain functionality. These devices have the ability configure themselves setup networking, and fetch latest software upgrades with minimal manual or user interaction.

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

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

v) **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 &LifeStyle

Physical Design of IoT :

The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities.

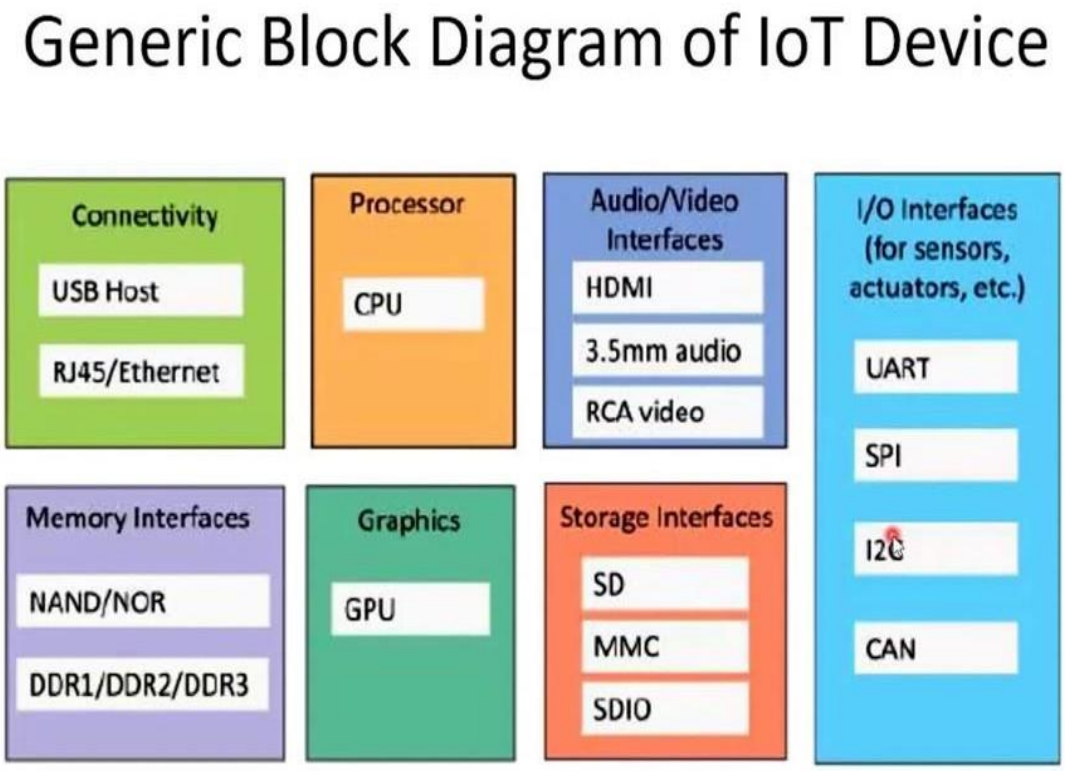
IoT devices can:

- Exchange data with other connected devices and applications (directly or indirectly), or
- Collect data from other devices and process the data locally or
- Send the data to centralized servers or cloud-based application back-ends for processing the data,
- Perform some tasks locally and other tasks within the IoT infrastructure, based on temporal and space constraints

Generic block diagram of an IoT Device

• An IoT device may consist of several interfaces for connections to other devices, both wired and wireless.
• I/O interfaces for sensors
• Interfaces for Internet connectivity

• Memory and storage interfaces
• Audio/video interfaces.



- HDMI: High definition multimedia Interface.
- 3.5mm: Audio Jack which headphone adapter.
- RCA: Radio corporation of America.
- UART: Universal Asynchronous Receiver Transmitter.
- SPI: Serial Peripheral Interface.
- I2C: Inter integrated circuit
- CAN: Controller Area Network used for Micro-controllers and devices to communicate.
- SD: Secure digital (memory card)
- MMC: multimedia card
- SDIO: Secure digital Input Output
- GPU: Graphics processing unit.
- DDR: Double data rate

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 signalled 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 overfiber.
- 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)

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*32addresses.
- IPv6: Internet Protocol version6 uses 128 bit address scheme and allows 2*128 addresses.
- 6LOWPAN:(IPv6 over Low power Wireless Personal Area Network) 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 congestioncollapse.
- 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 requestresponse 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.

LOGICAL DESIGN of IoT

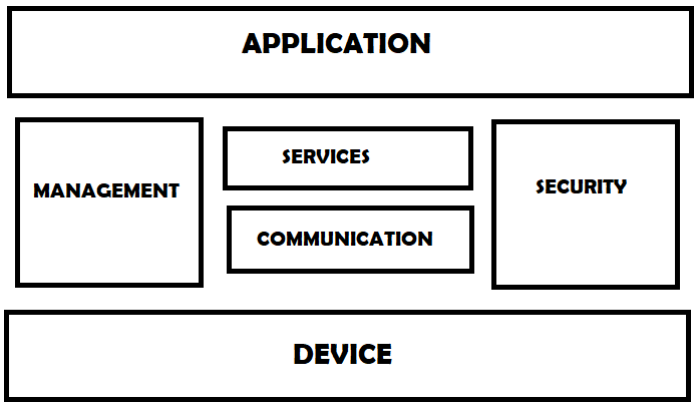
Refers to an abstract represent of entities and processes without going into the low level specifies of implementation.

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 IoT system.
- 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.



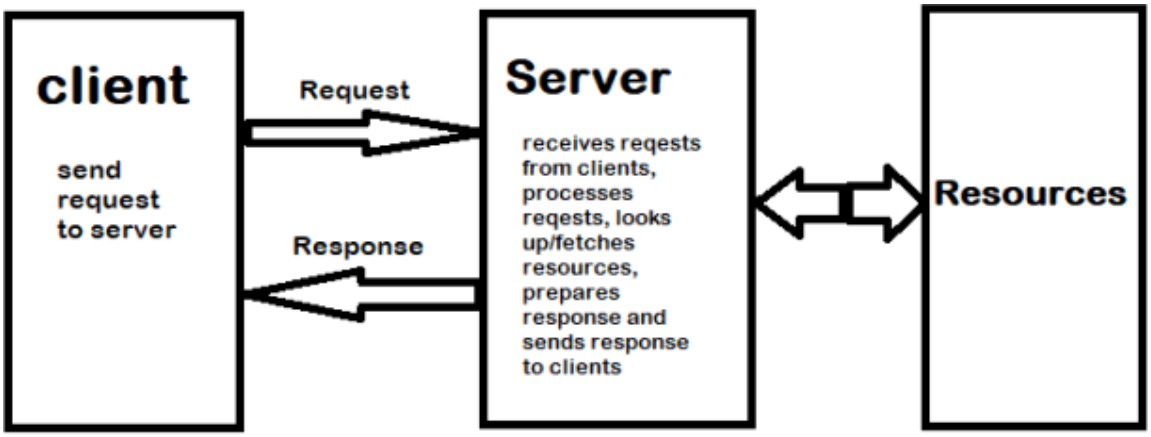
2) IoT Communication Models:

A) Request-Response

B) Publish-Subscribe
C)Push-Pull
D) Exclusive Pair

A) Request-Response

Request-Response is a communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves resource representations, prepares the response, and then sends the response to the client.



Request-Response Communication Model

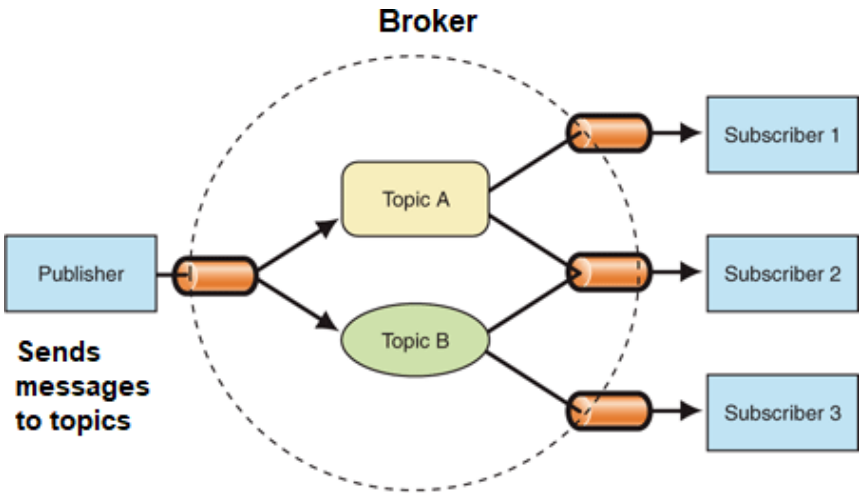
B) Publish-Subscribe communication model:

a. Publish-Subscribe is a communication model that involves publishers, brokers and consumers.

b. Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers.

c. Consumers subscribe to the topics which are managed by the broker.

d. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers

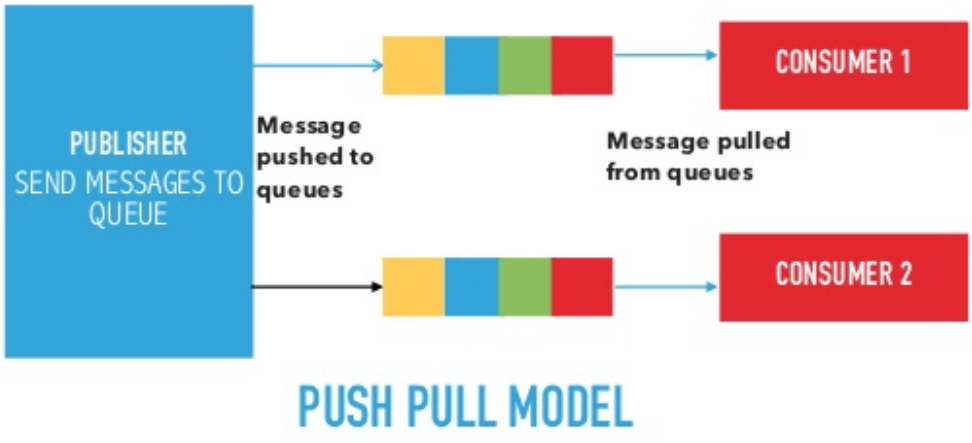


C) Push-Pull communication model:

a. Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers.

b. Queues help in decoupling the messaging between the producers and consumers.

c. Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate at which the consumers pull.



PUSH PULL MODEL

D) Exclusive Pair communication model:

a. Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.

b. Once the connection is setup it remains open until the client sends a request to close the connection.

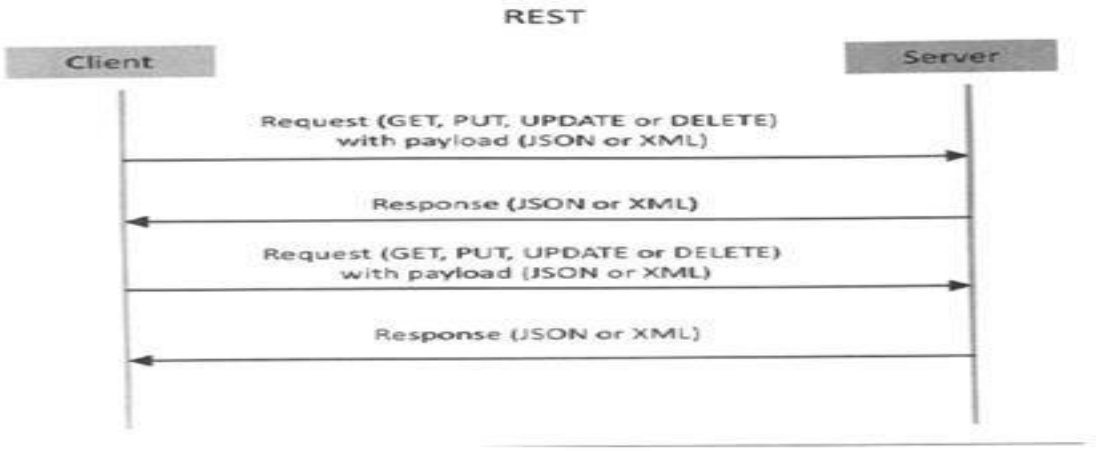
c. Client and server can send messages to each other after connection setup.

EXCLUSIVE PAIR COMMUNICATION MODEL

3) IoT Communication APIs: a) REST based communication APIs(Request-Response Based Model)
b) WebSocket based Communication APIs(Exclusive PairBasedModel)

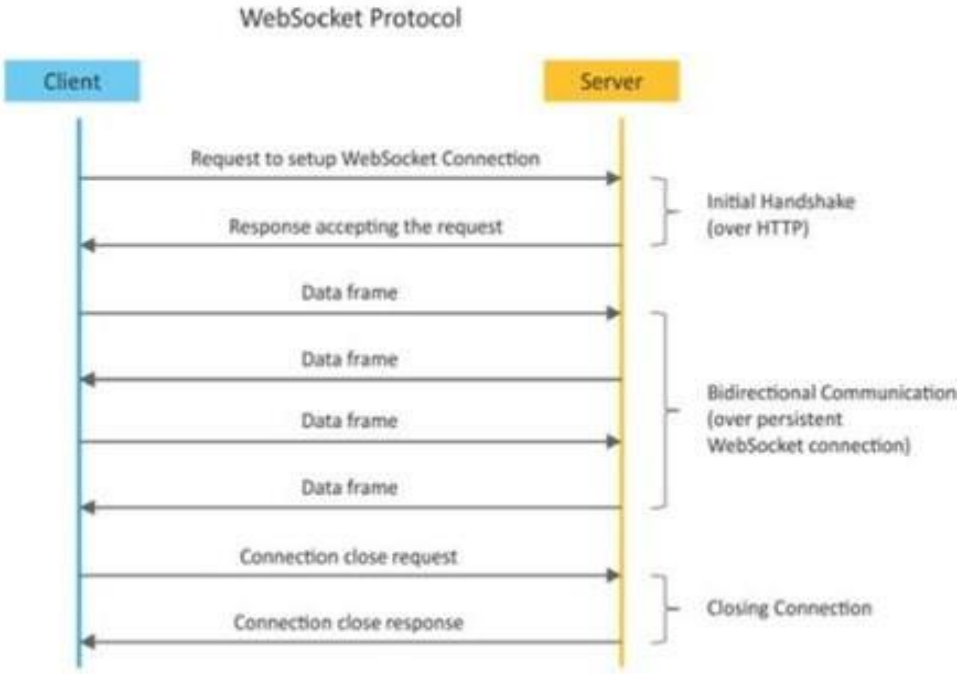
Request-Response model used by REST:

RESTful webservice is a collection of resources which are represented by URIs. RESTful web API has a base URI(e.g: http://example.com/api/tasks/). The clients and requests to these URIs using the methods defined by the HTTP protocol(e.g: GET, PUT, POST or DELETE). A RESTful web service can support various internet media types.



Request-response model used by REST

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.

1.4 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.

1.4.1 Wireless Sensor Networks

A wireless sensor network comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. A WSN consist of a number of end nodes and routers and a co-ordinator. The coordinator collects the data from all the nodes. Coordinator also acts as a gateway that connects the WSN to the internet.

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.

WSNs are enabled by wireless communication protocols such as IEEE 802.15.4. Zig Bee is one of the most popular wireless technologies used by WSNs .Zig Bee specifications are based on IEEE 802.15.4. Zig Bee operates 2.4 GHz frequency and offers data rates upto 250 KB/s and range from 10 to 100meters.

1.4.2 Cloud Computing

Cloud computing is a transformative computing paradigm that involves delivering applications and services over the internet. Cloud computing involves provisioning of computing, networking and storage resources on demand and providing these resources as metered services to the users, in a “pay as you go”. Cloud computing resources can be provisioned on-demand by the users, without requiring interactions with the cloud service provider. The process of provisioning resources is automated.

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. The cloud service provider manages the underlying cloud infrastructure including servers, network, operating systems, storage, and application software.

1.4.3 Big data Analysis

Big data is defined as collections of data sets whose volume , velocity or variety is so large that it is difficult to store, manage, process and analyze the data using traditional databases and data processing tools.

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.

The underlying characteristics of Big Data are

Volume: There is no fixed threshold for the volume of data for big data. Big data is used for massive scale data.

Velocity: Velocity is another important characteristics of Big Data and the primary reason for exponential growth of data.

Variety: Variety refers to the form of data. Big data comes in different forms such as structured or unstructured data including test data, image , audio, video and sensor data .

1.4.4 Communication Protocols:

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.

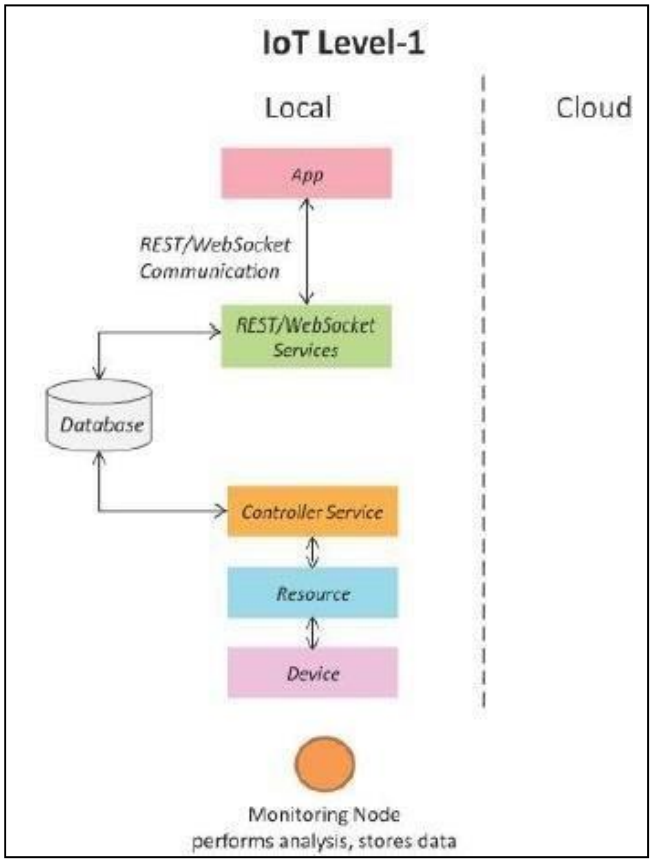
1.4.5 Embedded Systems:

Embedded Systemsis a computer system that has computer hardware and software embedded to perform specific tasks. Key components of embedded system include microprocessor or micro controller, memory (RAM, ROM, Cache), networking units (Ethernet Wi-Fi Adaptor), input/output units (Display, Keyboard, etc..,) and storage (Flash memory). Embedded System range from low cost miniaturized devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

1.5 IOT Levels and Deployment Templates.

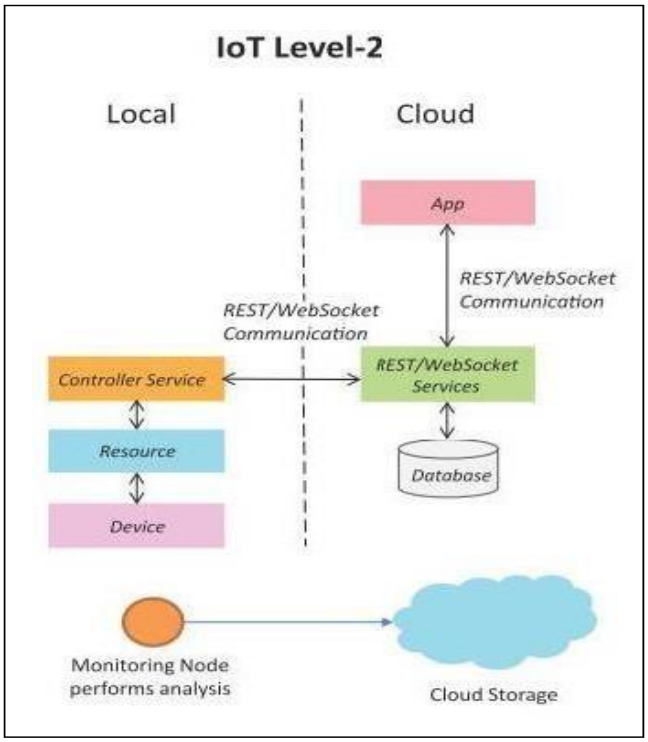
1.5.1 IoT Level-1

Level-1 IoT systems has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application. 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 isHomeautomation.The system consist of a single node that allows controlling the lights and appliances in a home the device used in this system interfaces with the lights and appliances using electronic rely switches. The status information of each light or appliances is maintained in a local database. REST services deployed locally allow retrieving and updating the state of each lighter appliance in the status database. The controller service continuously monitors the state of each light or appliance by retrieving the light from the database.



1.5.2 IoT Level 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



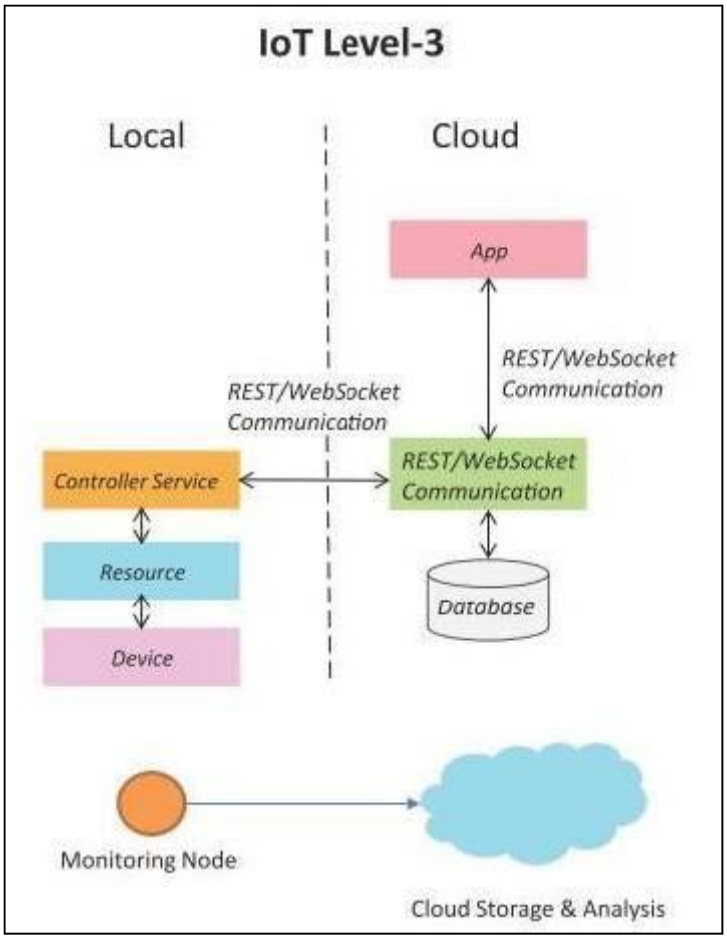
be done locally itself. An e,g., of Level2 IoT system for Smart Irrigation.

The system consists of a single node that monitors the soil moisture level and controls the irrigation system. The device used system collects soil moisture data from sensors. The controller service continuously monitors the moisture level. A cloud based REST web service is used for storing and retrieving moisture data which is stored in a cloud database. A cloud based application is used for visualizing the moisture level over a period of time which can help in making decision about irrigation schedule.

1.5.3 IoT Level 3

This 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.

The system consists of a single node that monitors the vibration levels for the package being shipped . The device in this system uses accelerometer and gyroscope sensor for monitoring vibration levels. The controller serves in the sensor data to the cloud in a real time using a websocket service. The data is stored in the cloud and also visualizing the cloud based applications . The analysis components in the cloud can trigger alerts if the vibration level becomes greater than the threshold.



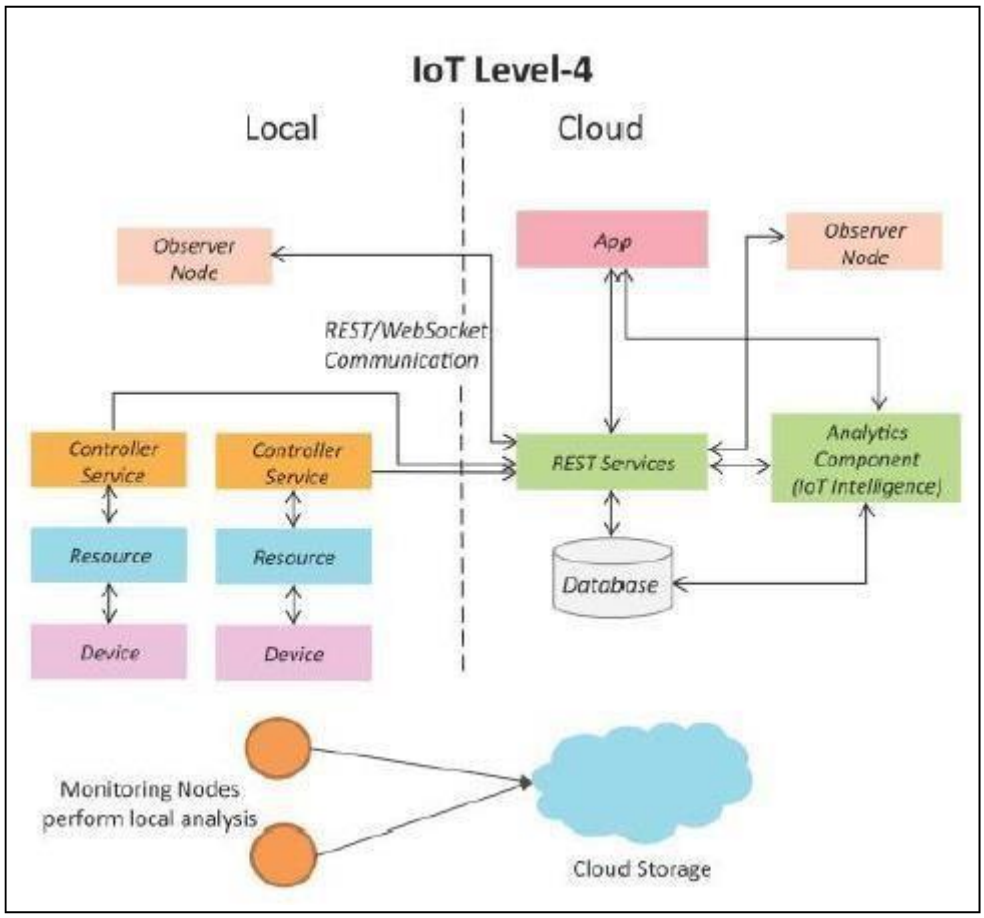
1.5.4 IoT Level 4

This 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. Level 4 IoT systems are suitable for solutions where multiple nodes are required, the data involved in big and the analysis requirements are computationally intensive.

Example : IoT System for Noise Monitoring.

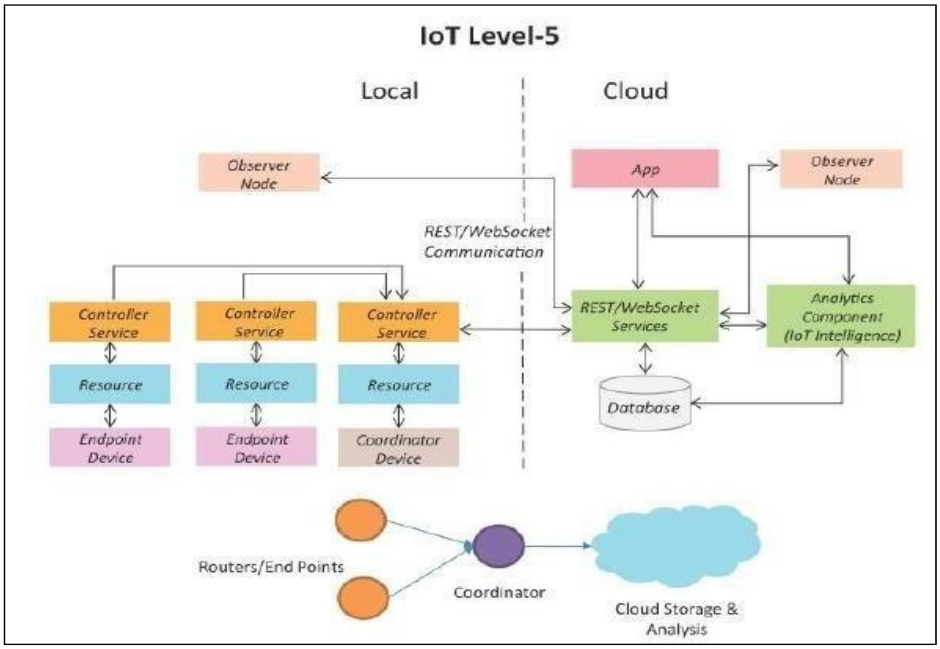
The system consists of multiple nodes placed in different locations for monitoring noise levels in an area. The nodes in this example are equipped with sound sensors. Nodes are independent of each other. Each

nodes runs its owner controller service that sends the data to the cloud . The data is stored in cloud database. The analysis of data collected from a number of nodes is done in the cloud. A cloud based application is used for visualizing the aggregated data.



1.5.5 IoT Level 5

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 are high intensive.



Example :IoT system for Forest Fire Detection.

The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and CO_2 levels in a forest. The end nodes in this example are equipped with various sensors such as temperature, humidity and CO_2. The coordinator node collects the data from the end nodes and act as a gateway that provides internet connectivity to the IoT system. The controller service on the coordinator device sends the collected data to the cloud. The data is stores in a cloud database. The analysis of data is done in the computing cloud to aggregate the data and make predictions. A cloud based applications is used for visualizing the data

1.5.6 IoT Level 6.

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 the cloud based applications. The centralized controller is aware of the status of all end nodes and sends control commands tothe nodes.

Example weather monitoring system

The system consists of multiple nodes placed in different locations for monitoring temperatures, humidity and pressure in an area. the end nodes are equipped with various sensors(such as temperature, humidity and pressure).the end nodes send the data to the cloud real time using a websocket service. the data is stored in a cloud database. The analysis of data is done in a cloud to aggregate a data and make predictions. A cloud based application is used for visualizing the data.

LEVELS	COST	COMPUTATION OF ANALYSIS	NODES	DATA STORAGE LOCATION	APPLICATION
1	Low	Low	Single	Local	Local
2	High	Low	Single	Cloud	Cloud
3	High	High	Single	Cloud	Cloud
4	High	Low	Multiple	Cloud	Cloud
5	High	High	Multiple	Cloud	Cloud
6	High	High	Multiple	Cloud	Cloud

DOMAIN SPECIFIC IoTs

1) Home Automation:

a) **Smart Lighting:** helps in saving energy by adapting the lighting to the ambient conditions and switching on/off or diming the light when needed.

b) **Smart Appliances:** make the management easier and also provide status information to the users remotely.

c) **Intrusion Detection:** use security cameras and sensors(PIR sensors and door sensors) to detect intrusion and raise alerts. Alerts can be in the form of SMS or email sent to the user.

d) **Smoke/Gas Detectors:** Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire. Alerts raised by smoke detectors can be in the form of signals to a fire alarm system. Gas detectors can detect the presence of harmful gases such as CO, LPGetc.,

2) Cities:

a) **Smart Parking:** make the search for parking space easier and convenient for drivers. Smart parking are powered by IoT systems that detect the no. of empty parking slots and send information over internet to smart application backends.

b) **Smart Lighting:** for roads, parks and buildings can help in saving energy.

c) **Smart Roads:** Equipped with sensors can provide information on driving condition, travel time estimating and alert in case of poor driving conditions, traffic condition and accidents.

d) **Structural Health Monitoring:** uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.

e) **Surveillance:** The video feeds from surveillance cameras can be aggregated in cloud based scalable storage solution.

f) **Emergency Response:** IoT systems for fire detection, gas and water leakage detection can help in generating alerts and minimizing their effects on the critical infrastructures.

3) Environment:

a) **Weather Monitoring:** Systems collect data from a no. of sensors attached and send the data to cloud based applications and storage back ends. The data collected in cloud can then be analyzed and visualized by cloud based applications.

b) **Air Pollution Monitoring:** System can monitor emission of harmful gases(CO2, CO, NO, NO2 etc.,) by factories and automobiles using gaseous and meteorological sensors. The collected data can be analyzed to make informed decisions on pollutions control approaches.

c) **Noise Pollution Monitoring:** Due to growing urban development, noise levels in cities have increased and even become alarmingly high in some cities. IoT based noise pollution monitoring systems use a no. of noise monitoring systems that are deployed at different places in a city. The data on noise levels from the station is collected on servers or in the cloud. The collected data is then aggregated to generate noise maps.

d) **Forest Fire Detection:** Forest fire can cause damage to natural resources, property and human life. Early detection of forest fire can help in minimizing damage.

e) **River Flood Detection:** River floods can cause damage to natural and human resources and human life. Early warnings of floods can be given by monitoring the water level and flow rate. IoT based river flood monitoring system uses a no. of sensor nodes that monitor the water level and flow rate sensors.

4) Energy:

a) **Smart Grids:** is a data communication network integrated with the electrical grids that collects and analyze data captured in near-real-time about power transmission, distribution and consumption. Smart grid technology provides predictive information and recommendations to utilities, their suppliers, and their customers on how best to manage power. By using IoT based sensing and measurement technologies, the health of equipment and integrity of the grid can be evaluated.

b) **Renewable Energy Systems:** IoT based systems integrated with the transformers at the point of interconnection measure the electrical variables and how much power is fed into the grid. For wind energy systems, closed-loop controls can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides power support.

c) **Prognostics:** In systems such as power grids, real-time information is collected using specialized electrical sensors called Phasor Measurment Units(PMUs) at the substations. The information received from PMUs must be monitored in real-time for estimating the state of the system and for predicting failures.

5) Retail:

a) **Inventory Management:** IoT systems enable remote monitoring of inventory using data collected by RFIDreaders.

b) **Smart Payments:** Solutions such as contact-less payments powered by technologies such as Near Field Communication(NFC) and Bluetooth.

c) **Smart Vending Machines:** Sensors in a smart vending machines monitors its operations and send the data to cloud which can be used for predictive maintenance.

6) Logistics:

a) **Route generation & scheduling:** IoT based system backed by cloud can provide first response to the route generation queries and can be scaled upto serve a large transportation network.

b) **Fleet Tracking:** Use GPS to track locations of vehicles inreal-time.

c) **Shipment Monitoring:** IoT based shipment monitoring systems use sensors such as temp, humidity, to monitor the conditions and send data to cloud, where it can be analyzed to detect foods poilage.

d) **Remote Vehicle Diagnostics:** Systems use on-board IoT devices for collecting data on Vehicle operations(speed, RPMetc.,) and status of various vehicle subsystems.

7) Agriculture:

a) **Smart Irrigation:** to determine moisture amount in the soil.

b) **Green House Control:** to improve productivity.

8) Industry:

a) Machine diagnosis and prognosis

b) Indoor Air Quality Monitoring

9) Health and Lifestyle:

a) Health & Fitness Monitoring

b) Wearable Electronics

Sensors:

- These form the front end of the IoT devices. These are the so-called "Things" of the system.
- Their main purpose is to collect data from its surroundings (sensors) or give out data to its surrounding (actuators).
- These have to be uniquely identifiable devices with a unique IP address so that they can be easily identifiable over a large network.
- These have to be active in nature which means that they should be able to collect real-time data.
- These can either work on their own (autonomous in nature) or can be made to work by the user depending on their needs (user-controlled).
- Examples of sensors are gas sensor, water quality sensor, moisture sensor, etc.

Different Types Of Sensors And Their Uses:

The following is a list of different types of sensors that are commonly used in various applications with examples. All these types are used for measuring one of the physical properties like Temperature, Resistance, Capacitance, Conduction, Heat Transfer etc.

Type of Sensor	**Used For**
Temperature Sensor	Controlling HVAC systems in homes and offices
Proximity Sensor	Detecting objects in automatic doors
Accelerometer Sensor	Screen orientation in smartphones
IR Sensor (Infrared Sensor)	Remote controls for TVs and other devices
Pressure Sensor	Monitoring tire pressure in vehicles
Light Sensor	Adjusting screen brightness on smartphones
Ultrasonic Sensor	Parking assistance in cars
Flow and Level Sensor	Managing water levels in tanks
Smoke, Gas and Alcohol Sensor	Detecting smoke and gas leaks in homes
Microphone (Sound Sensor)	Voice recognition in smart speakers
Touch Sensor	Touchscreens on smartphones and tablets
Color Sensor	Color detection in industrial sorting machines
Humidity Sensor	Controlling humidity levels in greenhouses

Magnetic Sensor (Hall Effect Sensor)	Detecting the position of a rotating object
Position Sensor	Tracking the position of machine parts
Tilt Sensor	Detecting the tilt of gaming controllers
PIR Sensor	Motion detection in security systems
Strain and Weight Sensor	Weighing items on digital scales
Gyroscope Sensor	Stabilizing drones during flight
Optical Sensor	Adjusting lighting in smart home systems
Capacitive Sensor	Touchpads on laptops
Piezoelectric Sensor	Detecting vibrations in musical instruments
Thermal Sensor	Temperature control in ovens
RFID Sensor	Tracking inventory in warehouses
Chemical Sensor	Monitoring air quality

Actuators:

- Actuators are devices that turn electrical signals into physical motions or movements.
- These devices are used to control and manipulate physical items in the real world using data from sensors in an IoT system.
- Actuators are essential to the operation of IoT devices because they enable the automation and control of physical systems and machinery.
- Actuators in IoT include motors, solenoids, hydraulic and pneumatic systems, and other devices that may control or manipulate physical things.

Examples of Actuators in Internet of Things

- **Smart Home Systems:** Actuators are an important part of smart home systems. They let customers remotely operate various equipment such as lights, heating systems, and security systems. A smart thermostat, for example, can use temperature sensors to change the temperature in a home, and an actuator can activate the heating or cooling system as needed.
- **Industrial Automation:** Actuators are frequently employed in industrial automation to control machines and other systems. Hydraulic actuators, for example, might be used to control the movement of robotic arms in a factory, while electric actuators could be used

to regulate the position of a conveyor belt.

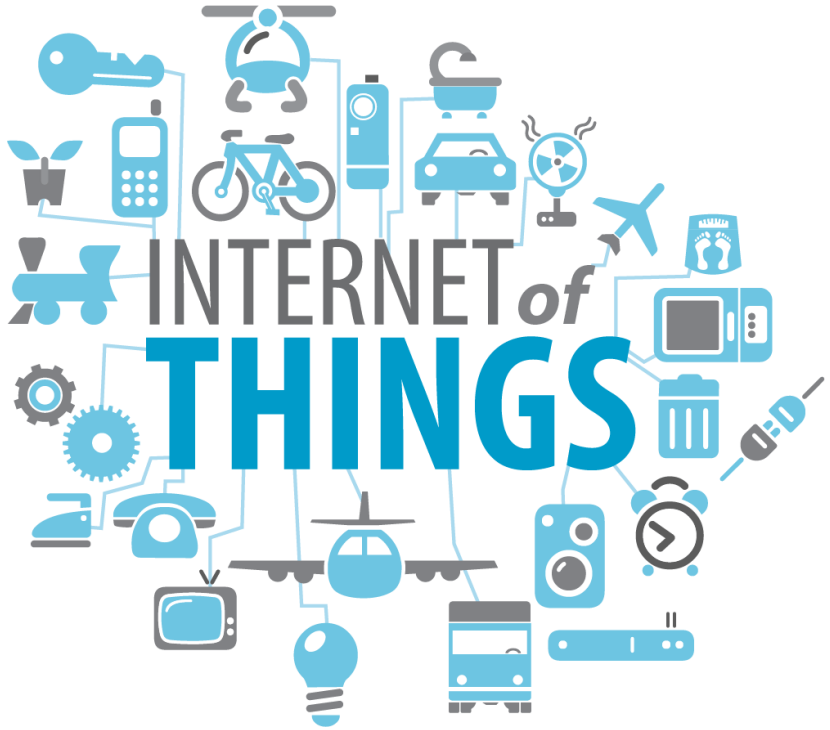
- **Agriculture:** Actuators are rapidly being employed to automate numerous processes in agriculture, such as irrigation and harvesting. An actuator, for example, can control the flow of water in an irrigation system or the position of a robotic arm used to harvest crops.
- **Healthcare:** Actuators are employed in a variety of healthcare applications, including prostheses and medical equipment. An actuator, for example, can regulate the movement of a prosthetic limb or the location of a surgical tool during treatment.

IOT & M2M

- Introduction
- M2M Vs IoT
- SDN & NFV
- IoT System Management
- SNMP
- Network Operator Requirements
- NETCONF
- YANG
- NETOPEER

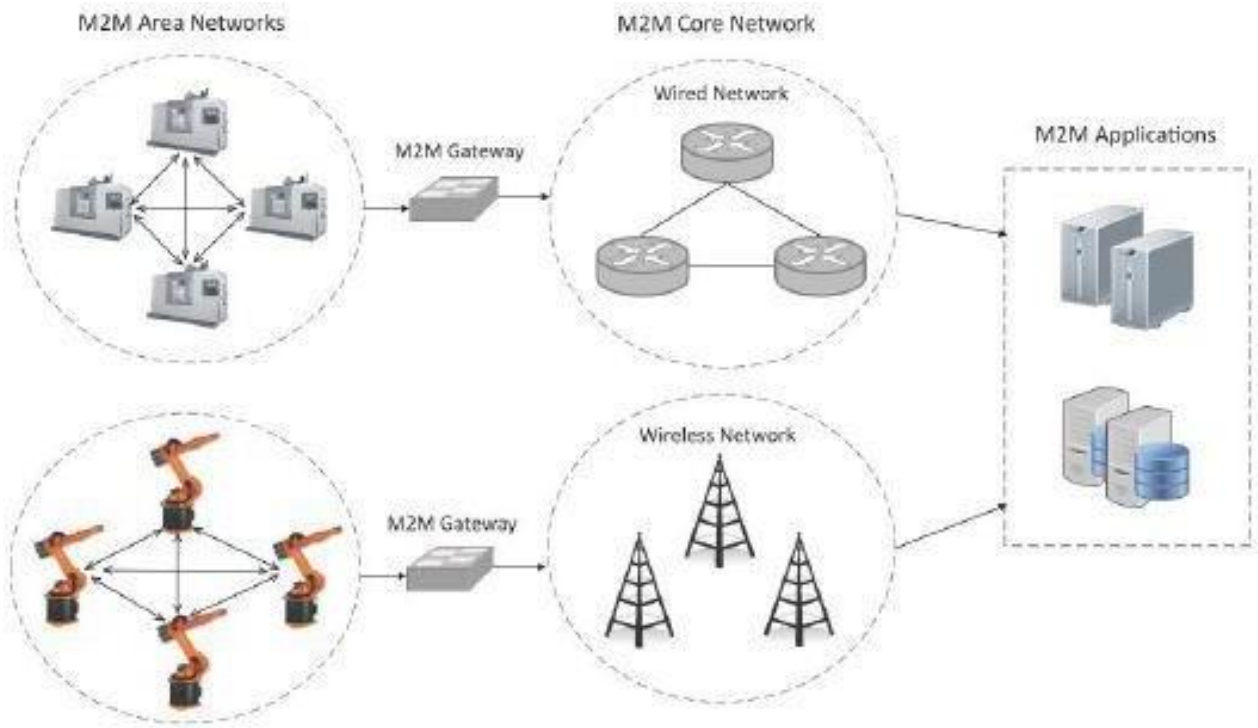
INTRODUCTION

- IoT
- M2M
  - TV-Remote
  - Mobile-AC



MACHINE-TO-MACHINE (M2M)

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

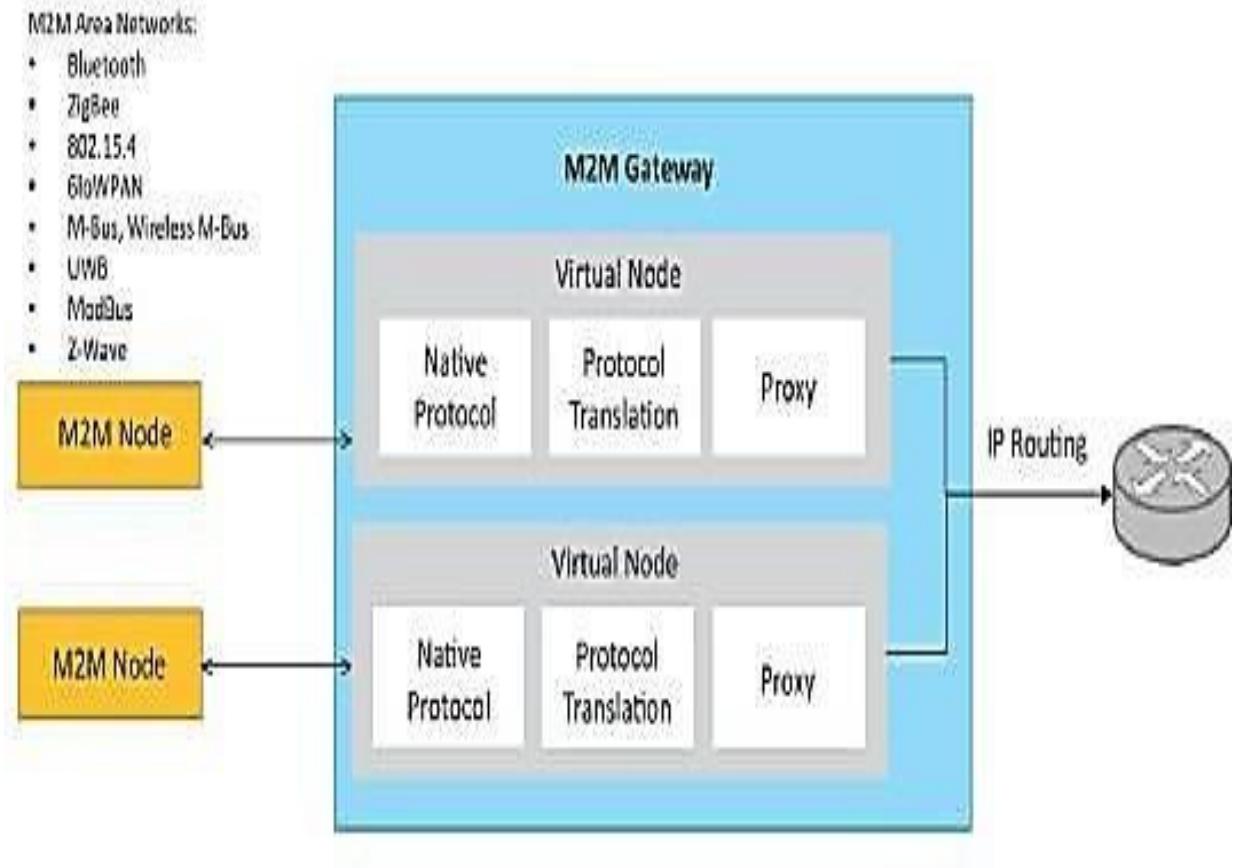


MACHINE-TO-MACHINE (M2M)

- An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication.
- Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.
- The communication network provides connectivity to remote M2M area networks.
- The communication network can use either wired or wireless networks (IPbased).
- While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks.

MACHINE-TO-MACHINE (M2M)

- Since non-IP based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in an external network.
- To enable the communication between remote M2M area networks, M2M gateways are used.



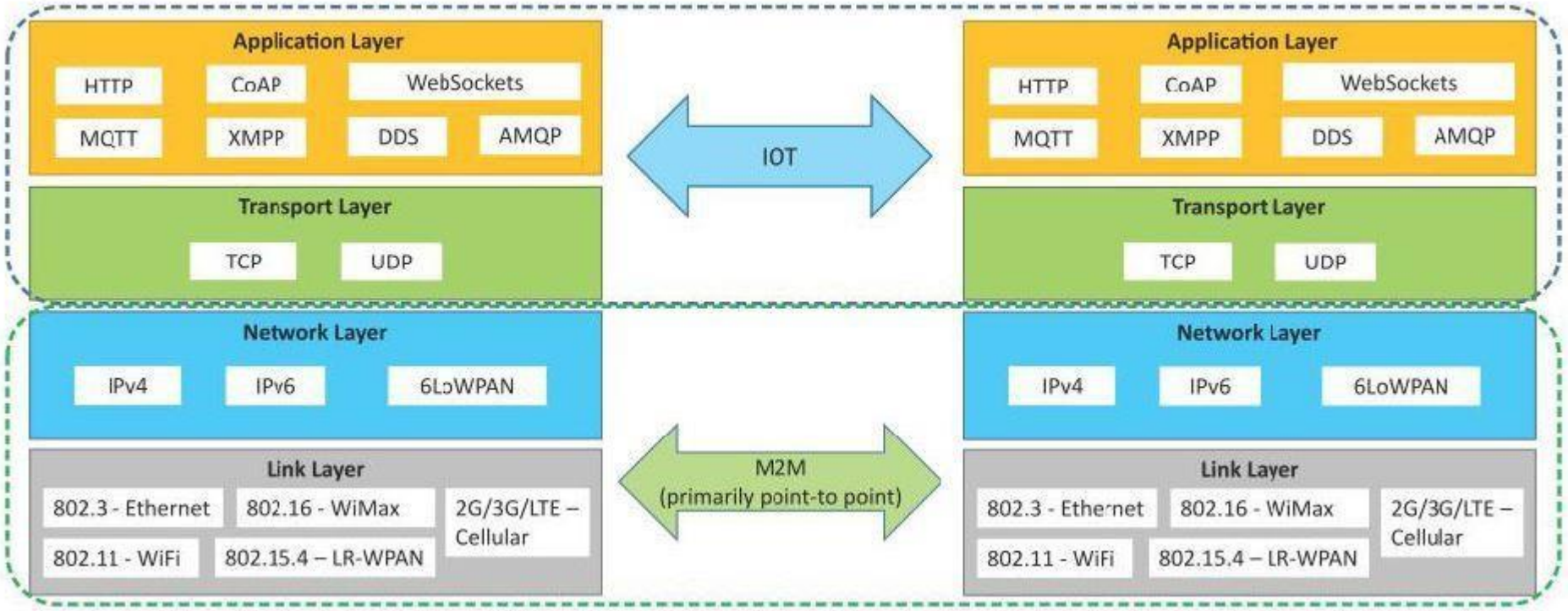
DIFFERENCE BETWEEN IOT &MACHINE-TO-MACHINE (M2M)

- Communication Protocols
  - M2M and IoT can differ in how the communication between the machines or devices happens.
  - M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks.
- Machines in M2M vs Things in IoT
  - The "Things" in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical states.
  - M2M systems, in contrast to IoT, typically have homogeneous machine types within an M2M area network.

DIFFERENCE BETWEEN IOT &MACHINE-TO-MACHINE (M2M)

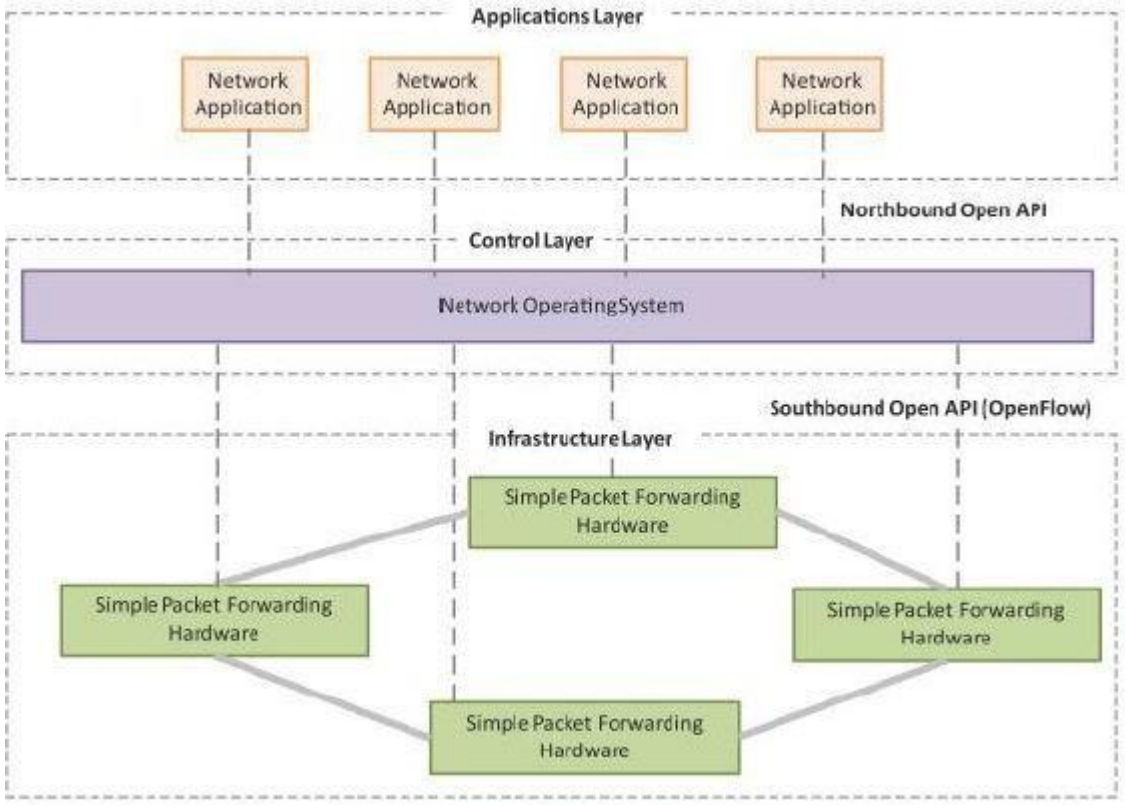
- Hardware vs Software Emphasis
  - While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
- Data Collection & Analysis
  - M2M data is collected in point solutions and often in on-premises storage infrastructure.
  - In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud).
- Applications
  - M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and onpremisis 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.

IOT & MACHINE-TO-MACHINE (M2M) COMMUNICATION



SDN

- Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller.
- Software-based SDN controllers maintain a unified view of the network and make configuration, management and provisioning simpler.
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.

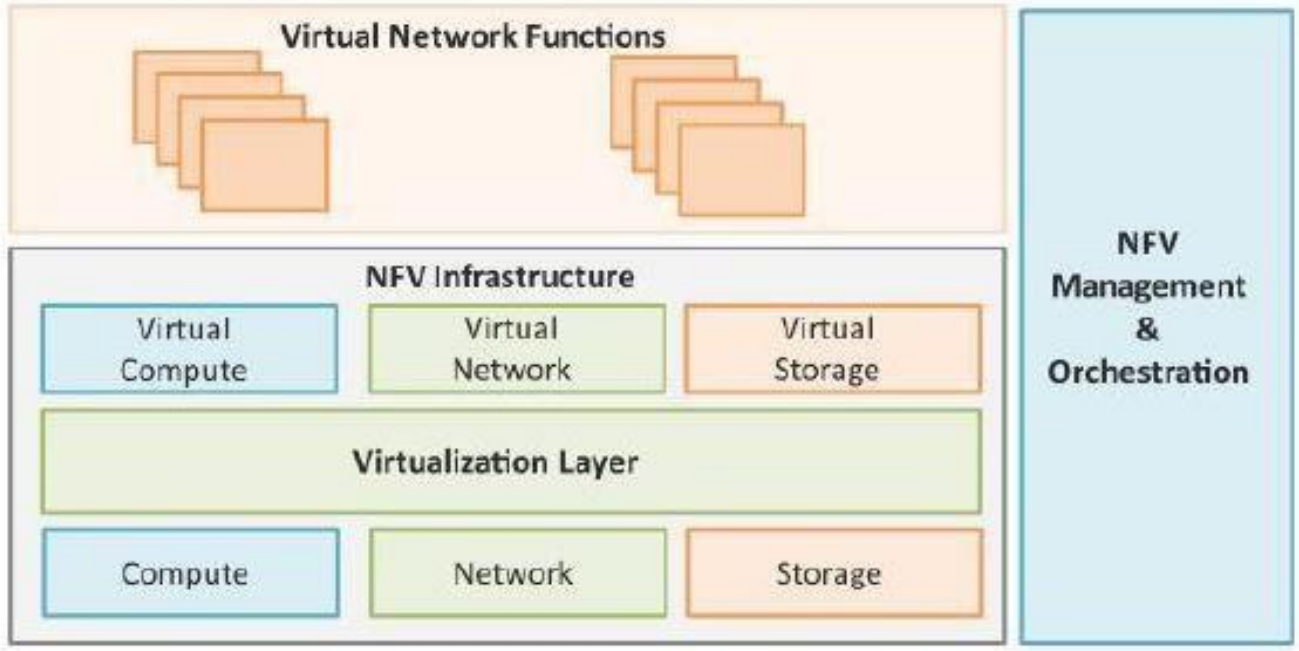


KEY ELEMENTS OF SDN

- Centralized Network Controller
  - With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.
- Programmable Open APIs
  - SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).
- Standard Communication Interface (OpenFlow)
  - SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface).
  - OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface.

NETWORK FUNCTION VIRTUALIZATION (NFV)

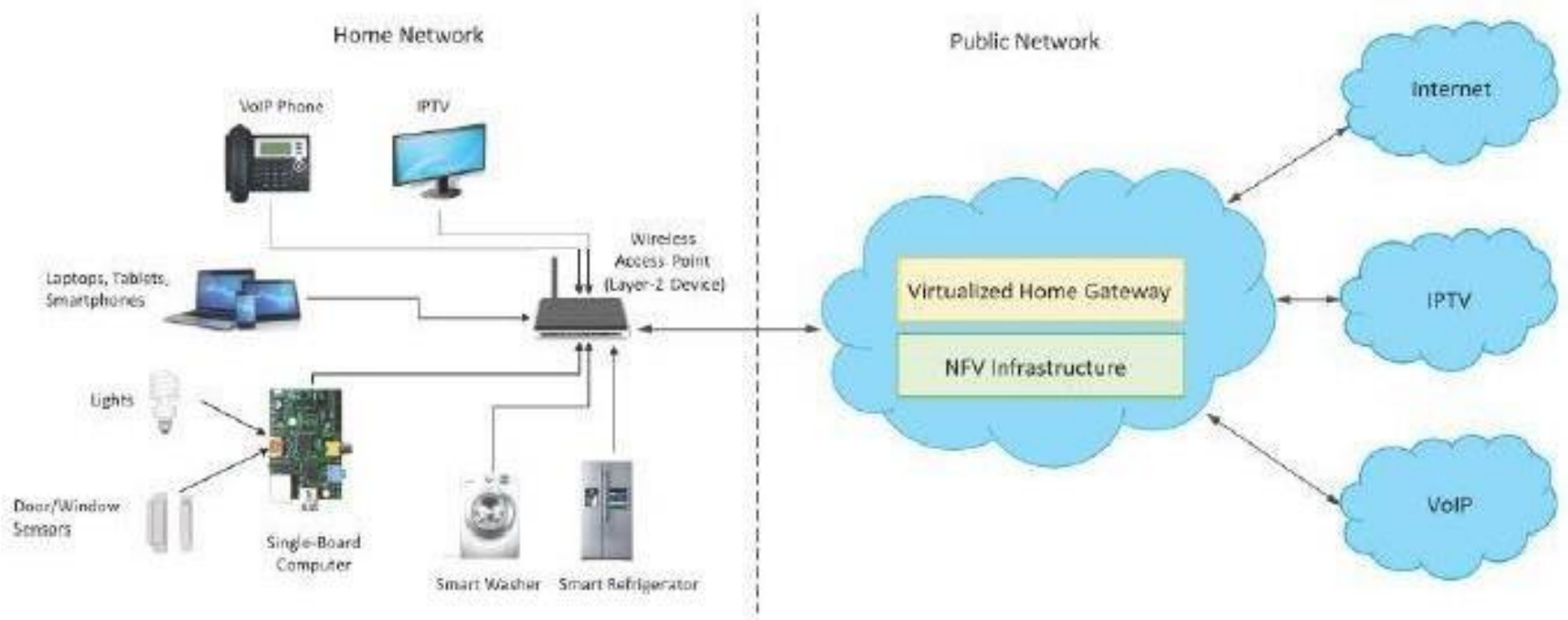
- Network Function Virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage.
- NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run.



KEY ELEMENTS OF NFV

- Virtualized Network Function (VNF):
  - VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
- NFV Infrastructure (NFVI):
  - NFVI includes compute, network and storage resources that are virtualized.
- NFV Management and Orchestration:
  - NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs.

NVF-USE CASE


Home Network
Public Network
VoIP Phone
IPTV
Laptops, Tablets, Smartphones
Wireless Access Point (Layer-2 Device)
Lights
Door/Window Sensors
Single-Board Computer
Smart Washer
Smart Refrigerator
Virtualized Home Gateway
NFV Infrastructure
Internet
IPTV
VoIP

NVF-USE CASE

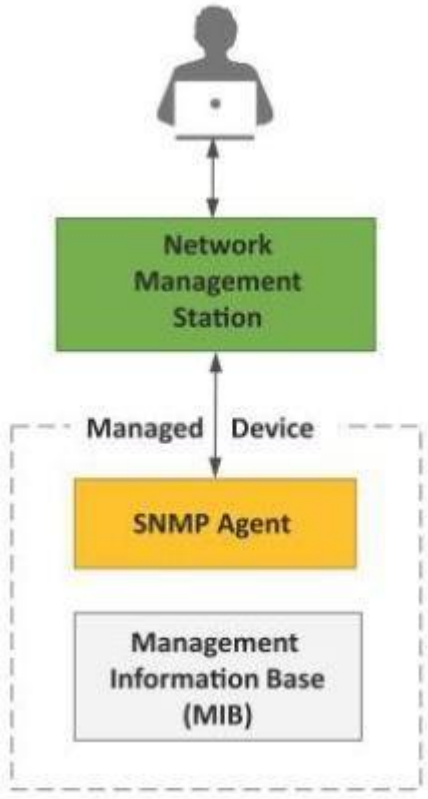
- NFV can be used to virtualize the Home Gateway.
- The NFV infrastructure in the cloud hosts a virtualized Home Gateway.
- The virtualized gateway provides private IP addresses to the devices in the home.
- The virtualized gateway also connects to network services such as VoIP and IPTV.

NEED FOR IOT SYSTEM 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



DRAWBACKS OF SNMP

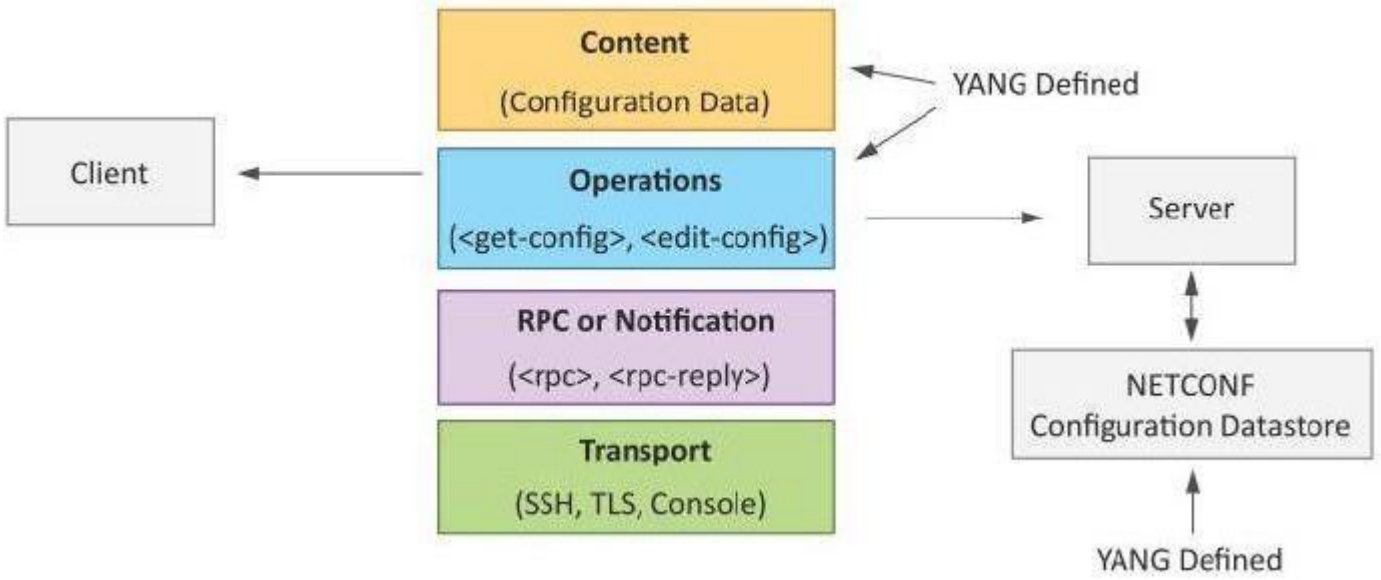
- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement 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 services
- Configuration deltas
- Dump and restore configurations
- Configuration validation
- Configuration database schemas
- Comparing configurations
- Role-based access control
- Consistency of access control lists:
- Multiple configuration sets
- Support for both data-oriented and task oriented access control

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

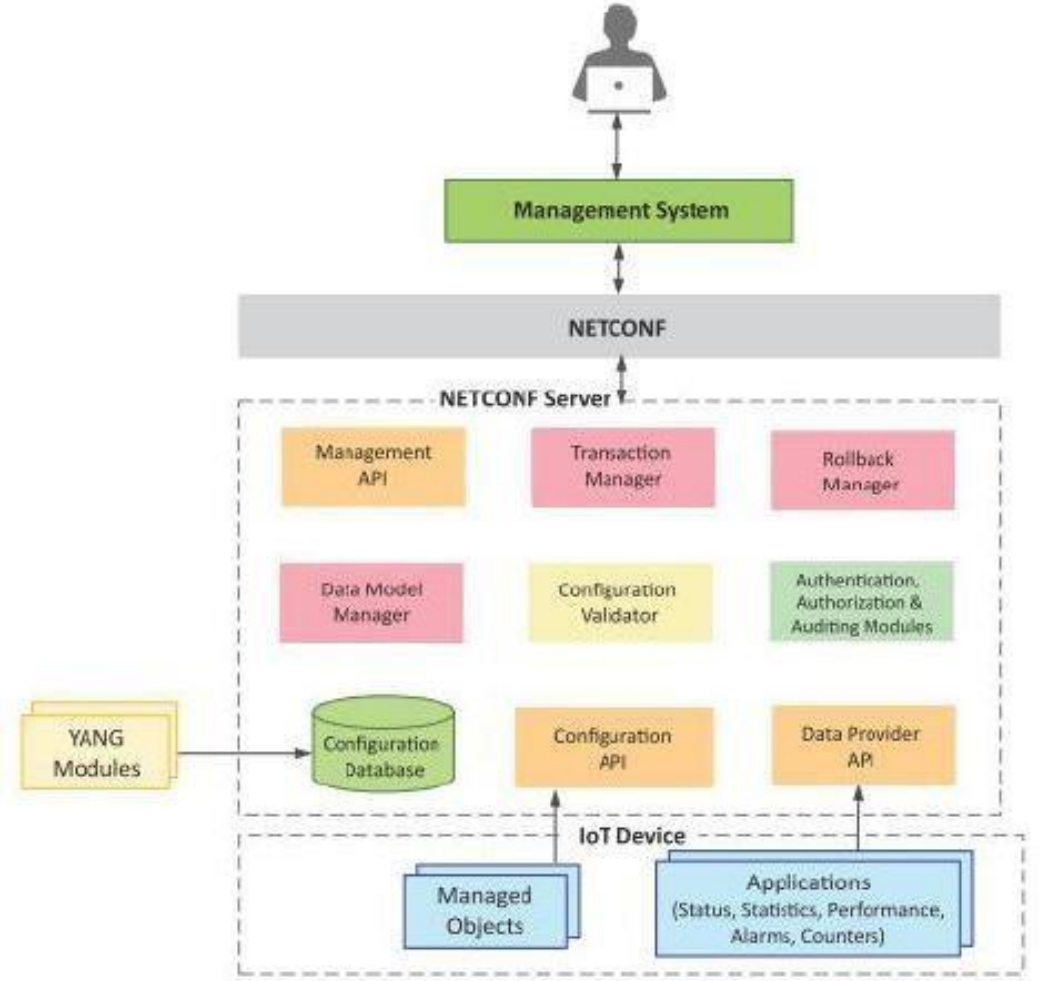
- NETCONF works on SSH transport protocol.
- Transport layer provides end-to-end connectivity and ensure reliable delivery of messages.
- NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages.
- The RPC layer provides mechanism for encoding of RPC calls and 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 modelling language called YANG.
- NETCONF provides a clear separation of the configuration and state data.
- The configuration data resides within a NETCONF configuration data store on the server.

YANG

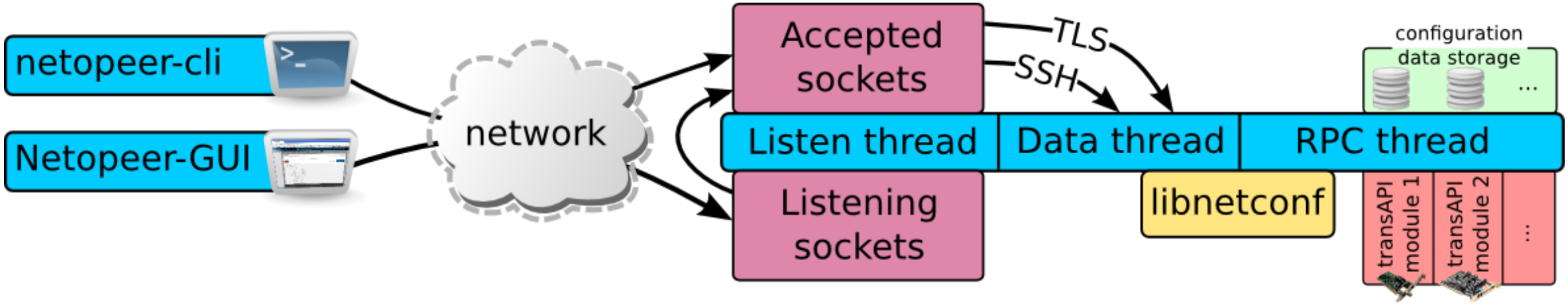
- YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol
- YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.
- YANG modules defines the data exchanged between 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



NETOPEER-NETCONF-YANG



UNIT III - IoT Physical Devices and Endpoints

UNIT III: IoT Physical Devices and Endpoints- Introduction to Arduino and Raspberry Pi- Installation, Interfaces (serial, SPI, I2C), Programming – Python program with Raspberry PI with focus on interfacing external gadgets, controlling output, reading input from pins.

IoT Physical Devices and Endpoints

- In an IoT ecosystem, physical devices and endpoints serve as the foundation for data collection, processing, and communication.
- These devices include sensors, actuators, microcontrollers, and single-board computers that interact with the physical world.
- IoT physical devices play a crucial role in enabling real-world applications such as smart homes, industrial automation, healthcare, and environmental monitoring.
- IoT devices can be broadly classified into two categories: microcontroller-based devices (e.g., Arduino) and single-board computers (e.g., Raspberry Pi).
- Microcontrollers are suitable for low-power, dedicated tasks, while single-board computers offer higher computational capabilities and more flexibility for complex applications.
- These devices communicate through various wired and wireless protocols, enabling seamless data transfer between endpoints and cloud-based systems.
- The choice of communication interface depends on the application's requirements in terms of power consumption, range, and data rate.
- Programming these devices is essential for interfacing with sensors, controlling actuators, and processing real-time data.
- Python is widely used for programming single-board computers like Raspberry Pi, whereas microcontrollers often utilize languages such as C or C++.
- Proper integration of hardware and software allows IoT devices to perform automated tasks efficiently and contribute to a connected ecosystem.

Introduction to Arduino and Installation :

- Arduino is an open-source hardware and software company, project, and user community.
- It designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices.
- Its products are licensed under the **GNU Lesser General Public License (LGPL)** or the **GNU General Public License (GPL)**, allowing anyone to manufacture Arduino boards and distribute software.
- Arduino boards are available in **preassembled form** or as **do-it-yourself (DIY) kits**.

Arduino Board Features:

- Uses a variety of **microprocessors and controllers**.
- Equipped with **digital and analog input/output (I/O) pins** for interfacing with expansion boards ('**shields**') and breadboards (**for prototyping**).
- Includes **serial communication interfaces**, such as **Universal Serial Bus (USB)** on some models.
- USB is also used for **loading programs from personal computers**.

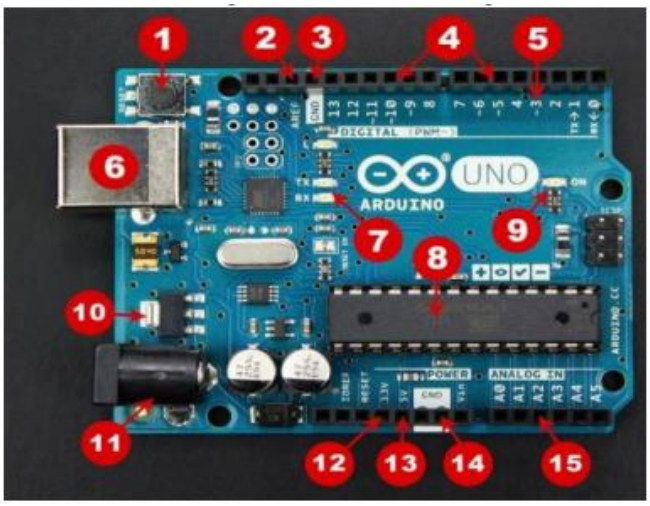
Programming and Development:

- The microcontrollers can be programmed using **C and C++** programming languages.
- Supports traditional **compiler toolchains** for development.
- Provides an **integrated development environment (IDE)** based on the **Processing language project**.

Arduino Overview

- **Arduino** is an open-source hardware and software platform.
- It is used for designing and developing digital devices using **microcontrollers**.
- Arduino products are licensed under **GNU Lesser General Public License (LGPL)** and **GNU General Public License (GPL)**.
- Arduino boards are available in **preassembled form** or as **DIY kits**.

Arduino Board Components and Functions



1. **Reset Button** – Restarts any loaded code on the Arduino board.
2. **AREF (Analog Reference)** – Used to set an external reference voltage.
3. **Ground Pins** – Multiple ground pins are available, all functioning the same.
4. **Digital Input/Output Pins (0-13)** – Used for digital input and output.
5. **PWM (~) Pins** – Simulate analog output.

6. **USB Connection** – Used for powering the board and uploading sketches.
7. **TX/RX LEDs** – Indicate data transmission and reception.
8. **ATmega Microcontroller** – The main processing unit where programs are stored.
9. **Power LED Indicator** – Lights up when the board is powered.
10. **Voltage Regulator** – Controls voltage input to prevent damage.
11. **DC Power Barrel Jack** – Allows power supply connection to the board.
12. **3.3V Pin** – Provides 3.3V power to projects.
13. **5V Pin** – Provides 5V power to projects.
14. **Analog Pins** – Read signals from analog sensors and convert them to digital.

Different Types of Arduino Boards and Their Specifications

Board Name	Operating Voltage	Clock Speed	Digital I/O	Analog Inputs	PWM	UART	Programming Interface
Arduino Uno R3	5V	16MHz	14	6	6	1	USB via ATMega16U2
Arduino Uno R3 SMD	5V	16MHz	14	6	6	1	USB via ATMega16U2
Red Board	5V	16MHz	14	6	6	1	USB via FTDI
Arduino Pro (3.3V/8MHz)	3.3V	8MHz	14	6	6	1	FTDI-Compatible Header
Arduino Pro (5V/16MHz)	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino Mini 05	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro Mini (3.3V/8MHz)	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro Mini (5V/16MHz)	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Ethernet	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino Fio	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header
LilyPad Arduino 328 Main Board	3.3V	8MHz	14	6	6	1	FTDI-Compatible Header
LilyPad Arduino Simple Board	3.3V	8MHz	9	4	5	0	FTDI-Compatible Header

Arduino IDE – Installation and Setup

STEP 1 : Gather Requirements:

- Obtain an Arduino board and a USB cable.
- If using Arduino Uno, Duemilanove, Nano, Mega 2560, or Diecimila, use a **standard USB cable (A plug to B plug)**.

- If using Arduino Nano, use an **A to Mini-B cable**.

STEP 2: Download Arduino IDE:

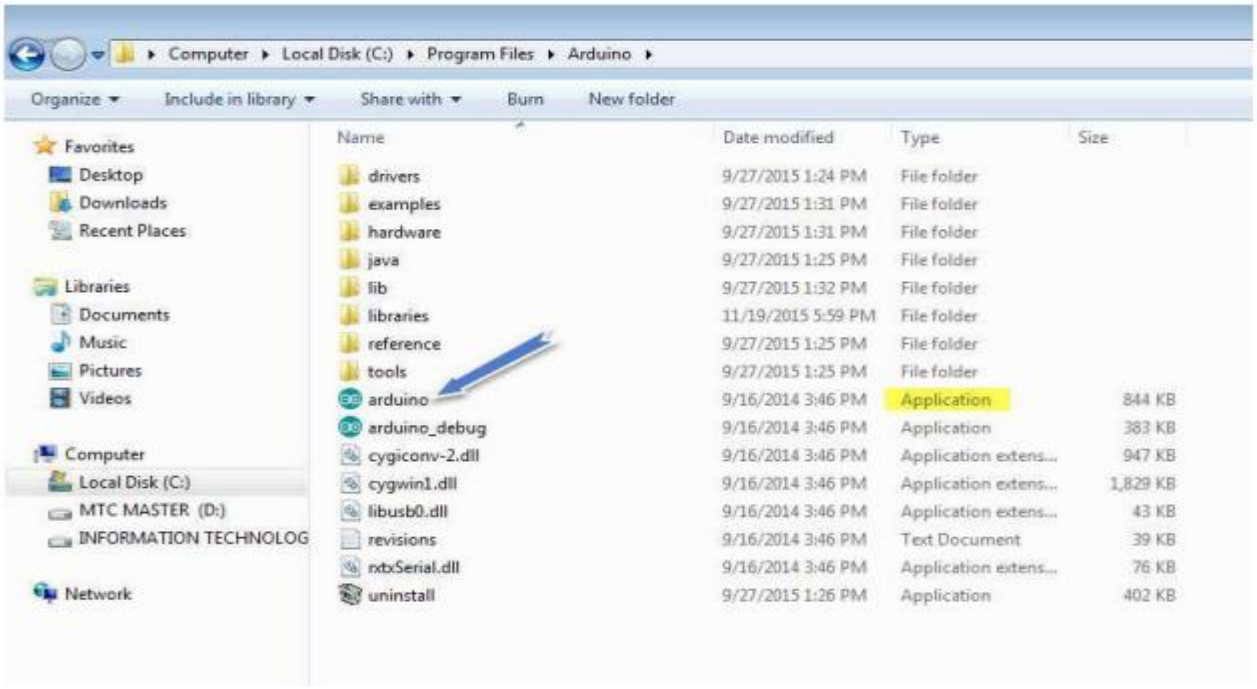
- Visit the **Arduino Official Website** and download the **IDE version** compatible with your operating system (**Windows, macOS, or Linux**).
- Unzip/install the software.

STEP 3 : Power Up Your Board:

- Connect the Arduino board to the computer using a **USB cable**.
- The **green power LED (PWR)** should light up.

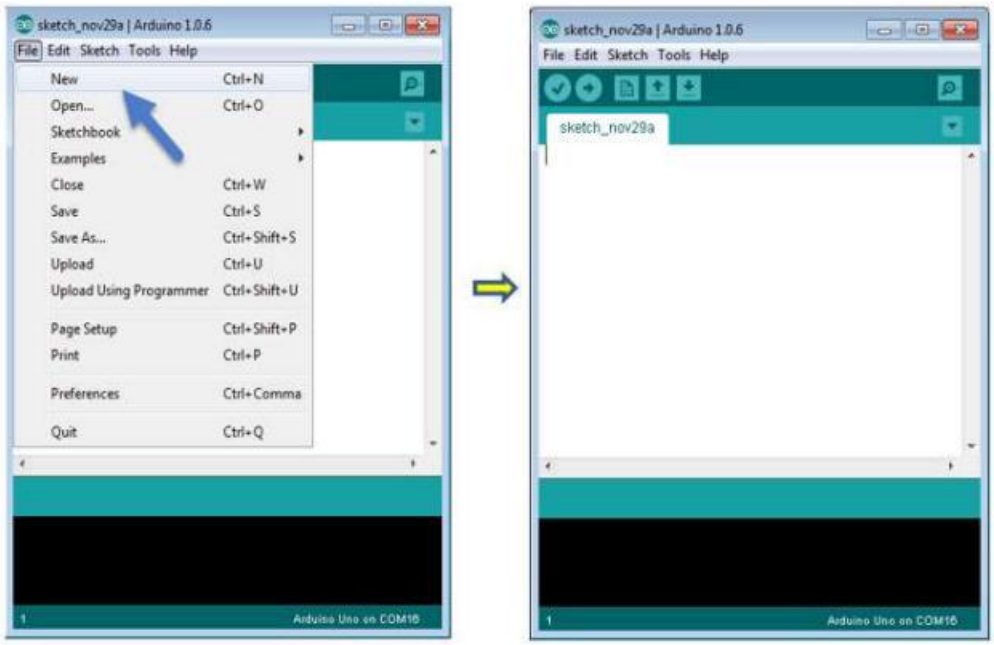
STEP 4 : Launch Arduino IDE:

- Locate the **application.exe** file inside the Arduino IDE folder and **double-click** to open it.

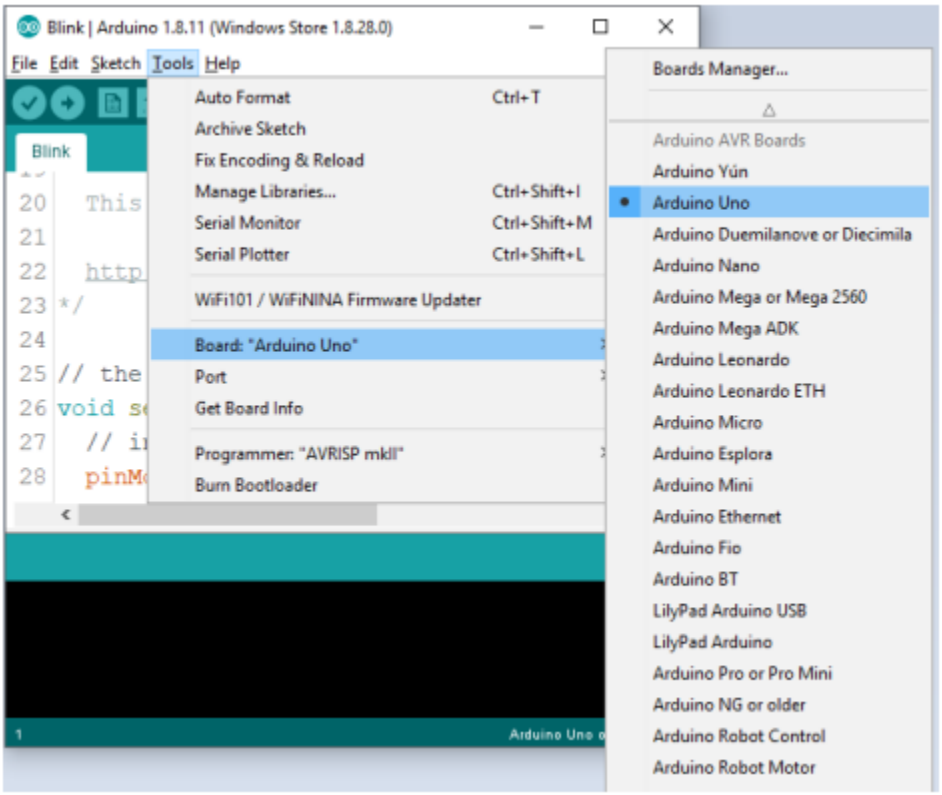


STEP 5 : Open Your First Project:

- Create a **new project** (`File -> New`)



- Open an **example project** (`File -> Examples -> Basics -> Blink`).



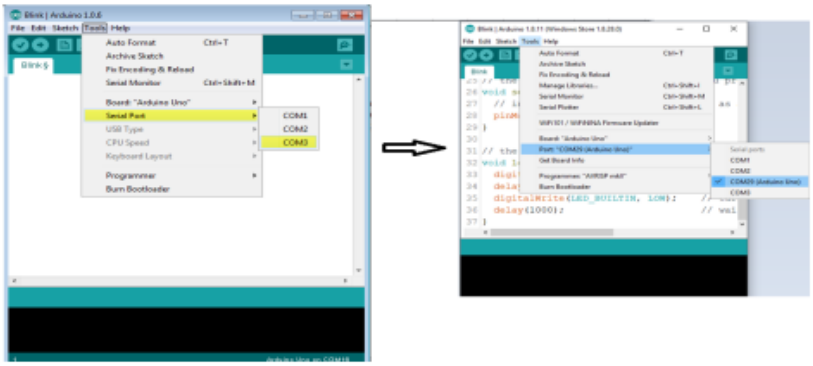
- The Blink program toggles an LED on and off.

STEP 6 : Select Your Arduino Board:

- Go to `Tools -> Board` and select the **correct board model**.

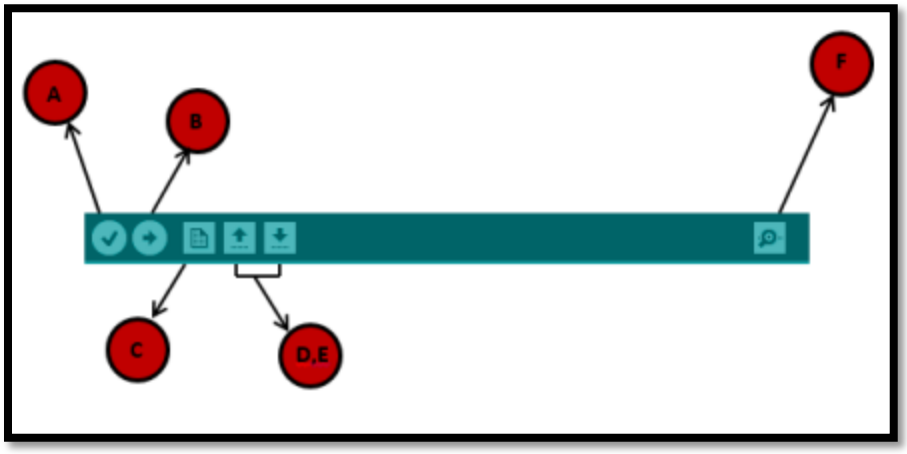
STEP 7 : Select Your Serial Port:

- Navigate to `Tools -> Serial Port` and choose the **Arduino board's port** (COM3 or higher).



STEP 8 : Upload the Program to the Board:

- Understand the **Arduino IDE Toolbar Functions:**



  - **A** – Checks for compilation errors.
  - **B** – Uploads the program to the board.
  - **C** – Creates a new sketch.
  - **D** – Opens an example sketch.
  - **E** – Saves the sketch.
  - **F** – Opens the Serial Monitor.
- Click the **Upload** button and wait for the **RX and TX LEDs** to flash.
- If successful, **"Done Uploading"** will appear in the status bar.
- For Arduino Mini, NG, or other boards, **press the reset button** before clicking **Upload**.

Arduino – Program Structure

- The **Arduino program structure** consists of key components and terminologies used in Arduino programming.
- The **Arduino software** is **open-source**, with:
  - **Java environment source code** released under **GPL**.
  - **C/C++ microcontroller libraries** under **LGPL**.

Sketch

- **Sketch** refers to an **Arduino program**.

Program Structure

- Arduino programs have **three main parts**:
  1. **Structure**
  2. **Values (Variables and Constants)**
  3. **Functions**

- The **structure** consists of two main functions:
    1. **setup() function**
    2. **loop() function**

setup() Function

- **Purpose**:
    - Called when a sketch starts.
    - Used to **initialize variables, set pin modes, and start libraries**.
    - Runs **only once** after power-up or reset.

Example:

```
void setup()

{
}
```

loop() Function

- **Purpose**:
    - Runs continuously after `setup()`.
    - Allows the program to change and respond in real-time.
    - Used to actively **control the Arduino board**.

Example:

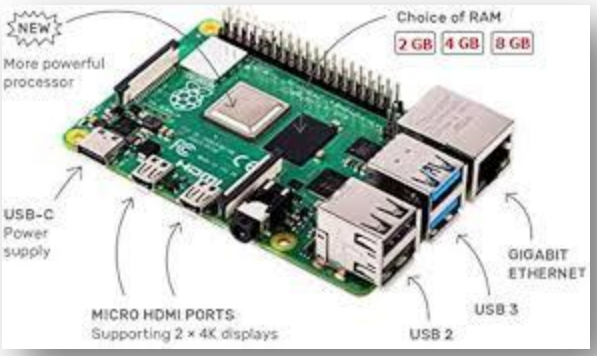
```
void loop()

{
}
```

Introduction to Raspberry Pi and Installation :

- **Raspberry Pi** is a **low-cost, credit-card-sized** single-board computer developed by the **Raspberry Pi Foundation**.
- It is used for **education, DIY projects, robotics, IoT applications, and industrial automation**.
- Runs on **Linux-based operating systems**, primarily **Raspberry Pi OS (formerly Raspbian)**.
- Supports programming languages like **Python, C, C++, Java, and Scratch**.

Raspberry Pi Board Components and Functions

1. **Processor (CPU & GPU)** – The main computing unit responsible for running applications.
2. **RAM (Memory)** – Provides temporary storage for running programs and processes.
3. **GPIO (General Purpose Input/Output) Pins** – Used to connect sensors, LEDs, and other external hardware.
4. **HDMI Port** – Connects the Raspberry Pi to a display or monitor.
5. **USB Ports** – Used for connecting peripherals like a keyboard, mouse, and storage devices.
6. **Ethernet Port** – Enables wired internet connectivity (except in some models).
7. **Wi-Fi & Bluetooth** – Wireless communication support (available in most models).



8. **Power Supply (USB-C or Micro-USB Port)** – Used to power the board.
9. **MicroSD Card Slot** – Stores the operating system and other files.
10. **Audio Jack** – Provides analog sound output.
11. **Camera Interface (CSI Port)** – Connects the Raspberry Pi Camera Module.
12. **Display Interface (DSI Port)** – Connects Raspberry Pi-compatible LCD displays.

Different Types of Raspberry Pi Boards and Their Specifications

Model	CPU	RAM	USB Ports	GPIO Pins	Ethernet	Wi-Fi & Bluetooth	Power Input
Raspberry Pi 4 Model B	Quad-core Cortex-A72	2GB/4GB/8GB	4 (USB 3.0 & 2.0)	40	Yes	Yes	USB-C
Raspberry Pi 3 Model B+	Quad-core Cortex-A53	1GB	4 (USB 2.0)	40	Yes	Yes	Micro-USB
Raspberry Pi 3 Model A+	Quad-core Cortex-A53	512MB	1 (USB 2.0)	40	No	Yes	Micro-USB
Raspberry Pi Zero W	Single-core ARM11	512MB	1 (USB OTG)	40	No	Yes	Micro-USB
Raspberry Pi Pico	Dual-core Cortex-M0+	264KB	No USB Host	26	No	No	Micro-USB

Raspberry Pi OS – Installation and Setup

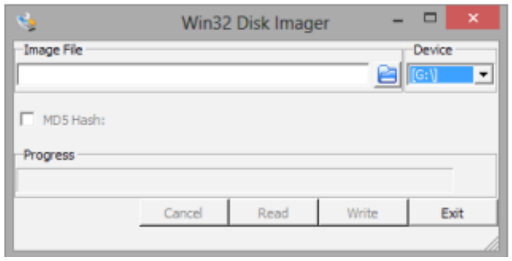
Essential Items and Recommendations :

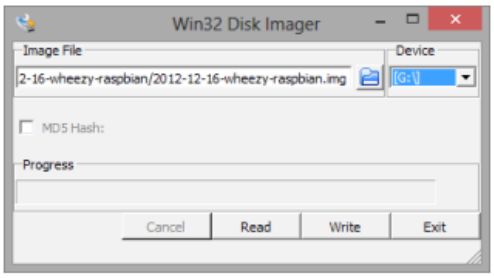
1. **SD Card**
   - Minimum **4GB**, **Class 4** or higher recommended.
   - Use **branded SD cards** for reliability.
2. **Display Connection**
   - **HDMI to HDMI** (for HD TVs and monitors).
   - **HDMI to DVI** (for monitors with DVI input).
   - **RCA Video Lead** (for analog displays, if HDMI is not used).
3. **Keyboard and Mouse**
   - Any **USB keyboard and mouse** will work.
   - **Wireless devices may need a powered USB hub**.
4. **Network Cable (Optional)**
   - Ethernet cable for easier software updates and network access.
5. **Power Adapter**
   - **Micro USB power supply (at least 700mA at 5V)**.
   - Check for a **stable power source** to avoid issues.
6. **Audio Output (Optional)**
   - HDMI provides **digital audio**.
   - RCA connection provides **analog stereo audio**.

Preparing the SD Card for Raspberry Pi

The SD card contains the Raspberry Pi's operating system (the OS is the software that makes it work, like Windows on a PC or OSX on a Mac)

1. **Download the Raspberry Pi OS (Raspbian)**
   - Get the latest version from the **official Raspberry Pi website**.
   - Example link:
     `http://downloads.raspberrypi.org/images/raspbian/...`
2. **Extract the OS Image**
   - Right-click the downloaded **.zip file** and select **"Extract all"**.
   - You will get a **.img file** after extraction.
3. **Download and Install Win32DiskImager** (For Windows Users)
   - Download: `https://launchpad.net/win32-image-writer/+download`
   - Unzip and open the **Win32DiskImager.exe** file.
4. **Write Raspbian OS to the SD Card**
   - Insert your **SD card** into your computer.
   - Open **Win32DiskImager** and select the **.img file**.
   - Click **Write** and wait for completion.





Booting the Raspberry Pi for the First Time

1. **Insert the SD card** into your Raspberry Pi.
2. **Connect power, keyboard, mouse, and display**.
3. **Turn on the Raspberry Pi** – It will boot into the **Raspi-config menu**.
4. **Initial Setup in Raspi-config:**
    - Set **timezone** and **locale** if needed.
    - Select **expand_rootfs** to use the full SD card storage.
    - Confirm reboot.
5. **Login to Raspberry Pi**
    - Default username: `pi`
    - Default password: `raspberry`
6. **Start the Desktop Interface**
    - Type `startx` and press Enter.
    - The Raspberry Pi **GUI (Graphical User Interface)** will appear.

Next Steps

- **Explore the Raspberry Pi desktop environment**.
- Visit **www.raspberrypi.org** for updates and project ideas.
- Join discussions at the **official Raspberry Pi forums**.

Raspberry Pi – Program Structure

- The **Raspberry Pi** supports various programming languages, including **Python, C, C++, Java, and Scratch**.
- **Python** is the most commonly used language due to its ease of use and built-in support for **GPIO programming**.

Raspberry Pi Basic Program Structure :

1. **Import Required Libraries**

- Libraries extend the functionality of the Raspberry Pi (e.g., GPIO control, I2C, SPI, etc.).

Example (Python – Import GPIO Library):

```
import RPi.GPIO as GPIO
import time
```

2. **Setup and Initialization**
   - Define pin numbering mode (**BCM or BOARD**).
   - Set input/output modes for GPIO pins.

Example (Setup GPIO Mode and Pin Configuration):

```
GPIO.setmode(GPIO.BCM)  # Use Broadcom pin-numbering scheme
GPIO.setup(18, GPIO.OUT)  # Set pin 18 as an output
```

3. **Main Program Loop**
   - Contains the logic of the program.
   - Runs continuously until manually stopped or interrupted.

Example (Blinking an LED on GPIO Pin 18):

```
while True:
    GPIO.output(18, GPIO.HIGH)  # Turn LED on
    time.sleep(1)  # Wait 1 second
    GPIO.output(18, GPIO.LOW)   # Turn LED off
    time.sleep(1)  # Wait 1 second
```

4. **Cleanup and Exit**
   - Ensures GPIO pins are reset before exiting.
   - Prevents unexpected behavior in subsequent runs.

Example (Cleanup on Program Exit):

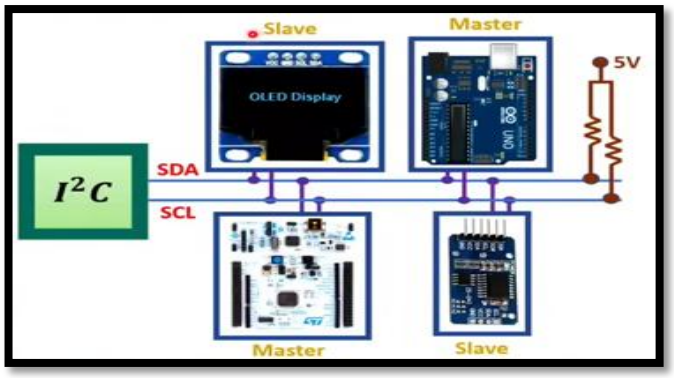
```
GPIO.cleanup()
```

Interfaces (serial, SPI, I2C) :

Basics of I2C Protocol / TWI {Two wire Interface}

- I2C is Inter Integrated Circuit Bus protocol.
- Only Two lines are required for the communication using 12C protocol.
- Here main advantage of 12C protocol is interfacing simplicity and with the use of Two lines we can interface 128 devices.
- It is simple, low bandwidth protocol used for short range communication.
- Most I2C devices operates up to speed of 400Kbps.

- 12C lines
    * Serial Data SDA
    * Serial Clock SCL



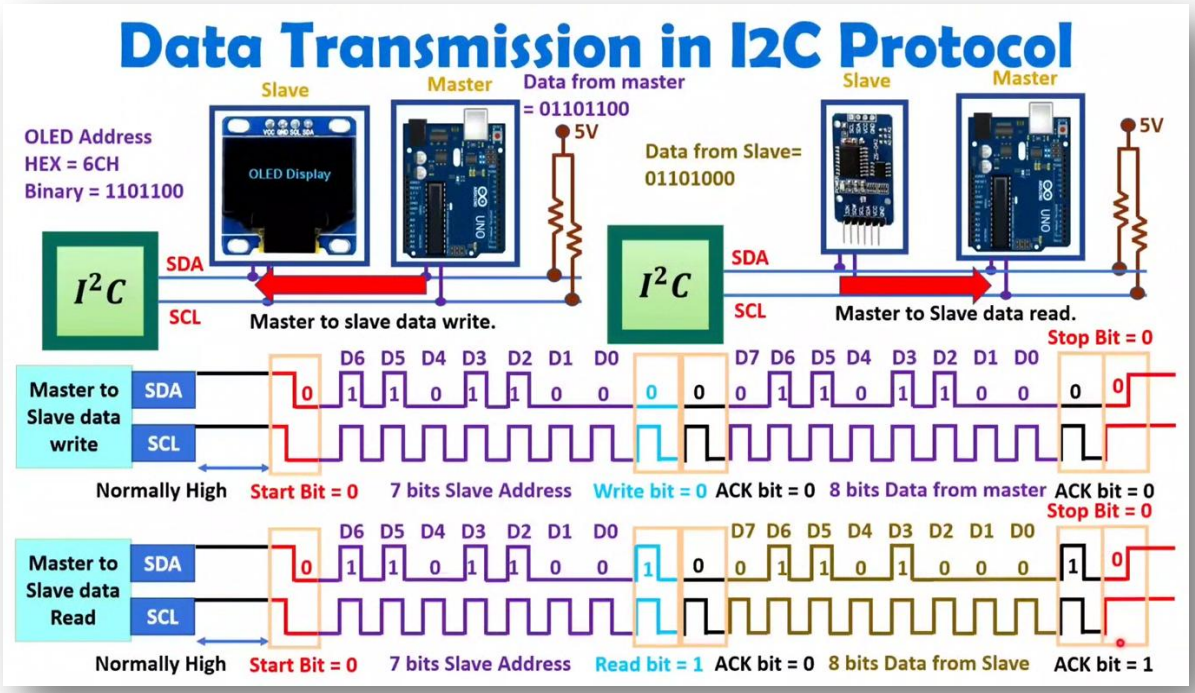
- Both of this lines are pulled up at 5V, explains that no communication is happening.
- Addressing is done by 7bits to communicates with 128 devices at max.
- Here number of devices is also limited by the total allowed bus capacitance of 400pF.
- Master is initiating data transfer and controls the clock normally and the device being addressed is slave.

Clock Stretching in 12C protocol

- An I2C Slave can hold of the master in the middle of transaction using clock stretching.
- Slave pull clock low until it's ready to continue.
- 12C support multiple master as well but most system are designed with one master and rest slaves.

Data Transmission in I2C Protocol :

In the I2C (Inter-Integrated Circuit) protocol, data communication between a master and slave



device occurs in two primary modes: writing data from the master to the slave and reading data from the slave to the master. Here's a detailed explanation of both processes:

Master to Slave Data Write

1. **Start Condition**: The master device initiates communication by sending a start condition. This is done by pulling the SDA line low while the SCL line is high, signaling all devices on the bus to prepare for communication.

2. **Slave Addressing** : The master then sends the 7-bit address of the target slave device followed by a read/write bit set to '0' (indicating a write operation). The addressed slave acknowledges by pulling the SDA line low.

3. **Data Transmission** : The master sends data in bytes (8 bits). After each byte, the slave must acknowledge receipt by sending an ACK signal (pulling SDA low).

4. **Multiple Bytes** : The master can send multiple bytes consecutively while the slave continues to acknowledge each byte. This allows for efficient batch data transmission.

5. **Stop Condition** : Once all data has been sent, the master terminates the communication by sending a stop condition. This is done by releasing the SDA line to high while the SCL line is high, indicating to all devices that the communication session has ended.

Master to Slave Data Read

1. **Start Condition :** Similar to the write operation, the master initiates the read process with a start condition.

2. **Slave Addressing** : The master sends the slave address, followed by a read/write bit set to '1' (indicating a read operation). The addressed slave acknowledges by pulling the SDA line low.

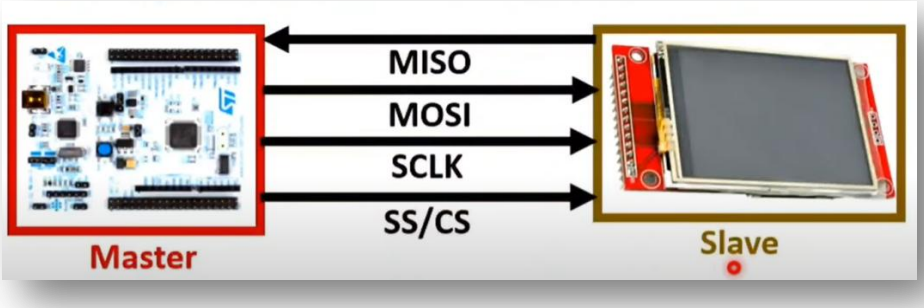
3. **Data Request** : The master can request one or more bytes of data from the slave. The slave prepares to send the requested data on the SDA line.

4. **Data Transmission** : The slave transmits data bytes, one at a time, to the master. After each byte, the master issues an ACK signal if it wishes to continue reading additional bytes (pulling SDA low). To stop reading, the master sends a NACK (not acknowledge) signal by leaving the SDA high after the last byte.

5. **Stop Condition** : After the data has been read, the master sends a stop condition, signaling the end of the communication.

Basics of SPI - Serial Peripheral Interface Bus

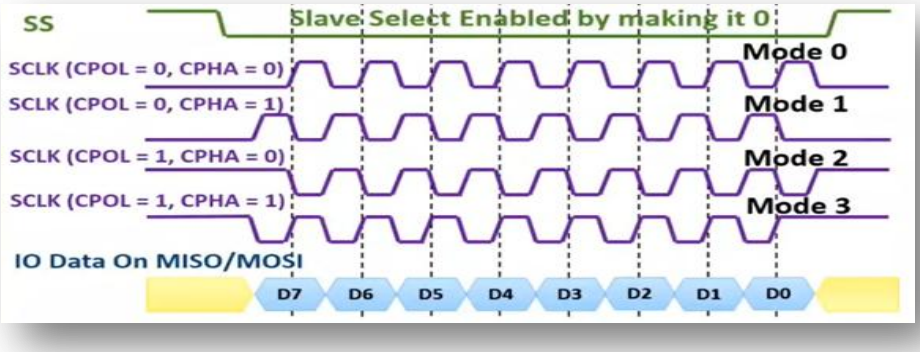
- SPI supports Half / Full Duplex serial communication.
- It is synchronous communication protocol. {As we provide clock along with data.}
- It is four wire communication protocol.
- In most cases one master multiple slave protocol, but it also supports multiple master configuration.
- It is on board short range communication protocol for embedded system.



- It supports maximum speed up to 10Mbps.
- Master provides clock for synchronization.
- SPI supports 8 & 16 bits data frame format.
- SS/CS {Slave Select / Chip Select} - Master selects the slave.
- SCLK (Serial Clock} - Line to send clock signal.
- MOSI {Master Out Slave In) - Line to send data from Master.
- MISO {Master In Slave Out) - Line to receive data from Slave.

Modes of SPI Protocol

- Based on Clock Polarity CPOL and Clock Phase CPHA, there are four operating modes.



Advantages of SPI Protocol

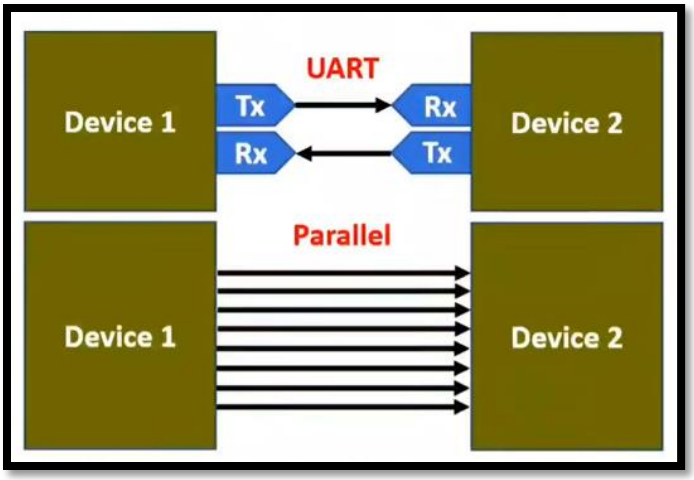
- No start & Stop bits required.
- SIP is full Duplex protocol.
- Signals are unidirectional allowing easy isolation.

Disadvantages of SPI Protocol

- Four wires required.
- Only One Master.
- Multiple SS required for multiple slave.
- No protocol for error detection.
- No acknowledgment for confirmation.

Basics of UART - Universal Asynchronous Receiver/Transmitter

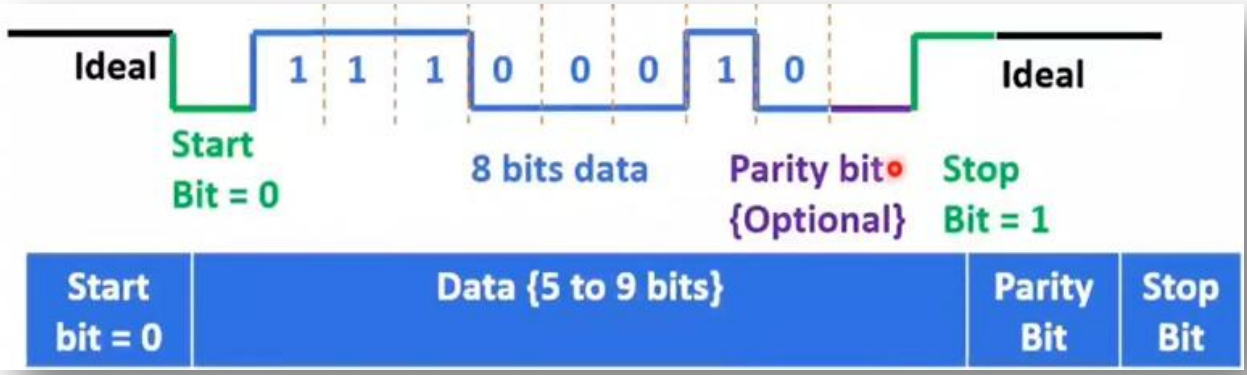
- UART is used for serial communication.
- It is two wire communication protocol:
- One wire is for transmission
- Second wire is for reception
- Data Format & Transmission speeds are configurable.
- Here, clock is not given for synchronisation.
- UART transmission speed is slower compared to parallel communication but it needs less bus interface for communication.



Configuration of UART before transmission

- Both devices should be configured with same transmission speed. {Baud rate}
- Fix the Data length. (5 to 9 bits}
- It requires start bit {1 bit) and stop bit (1 or 2 bits).
- NRZ encoding is used for data communication.

Data Format of UART



Advantages of UART Protocol

- Less physical interfacing.
- Simple configuration.
- Data size is configurable.
- Speed is configurable.
- Full Duplex mode can be configure by two wire.
- Error identification mechanism {One bit error can be identified}

Disadvantages of UART Protocol

- Low speed communication.
- Start & Stop bit required.
- Asynchronous communication.
- Redundant bits are present. {Start bit + Stop bits + parity bit}

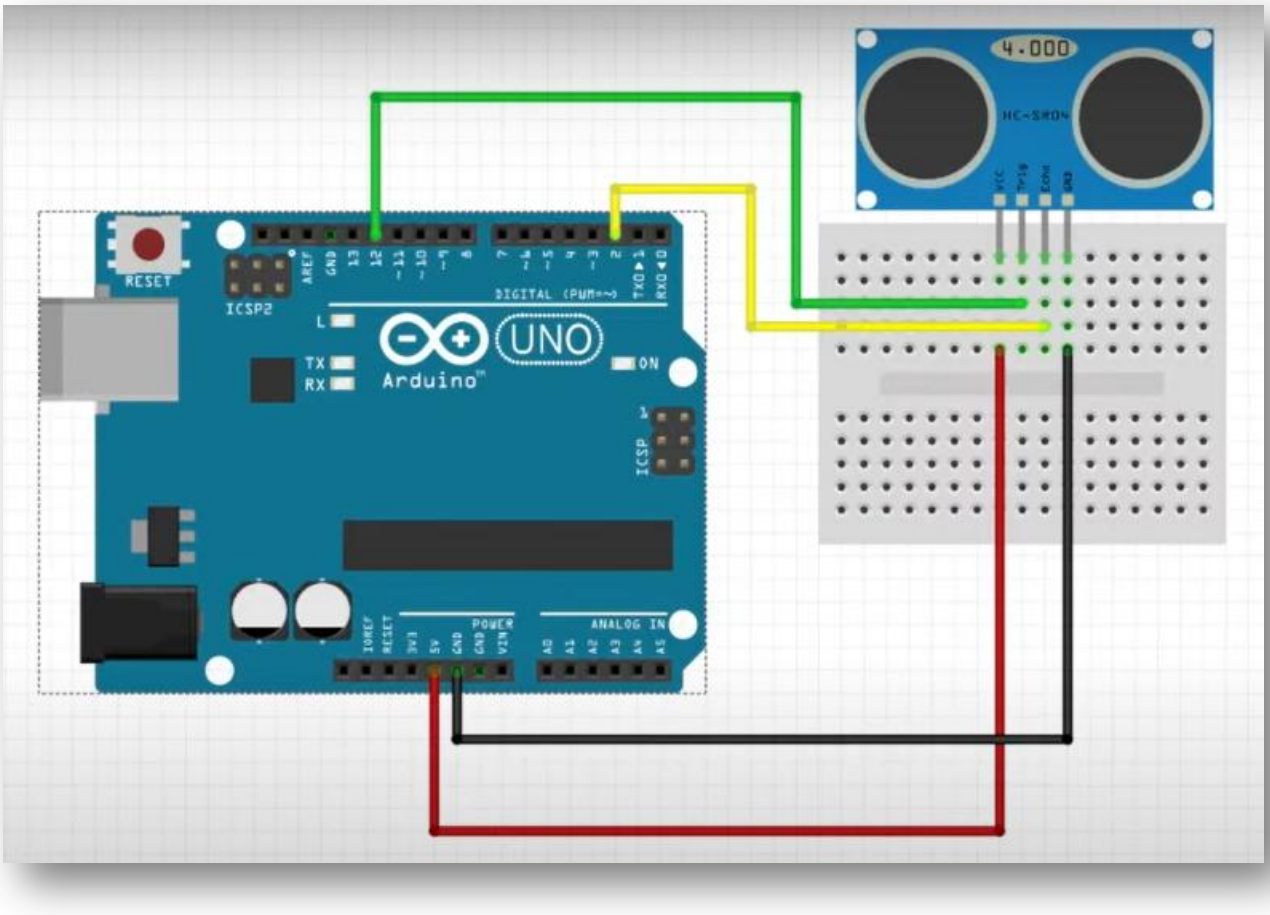
Arduino - Interfacing with Serial / UART :

Interfacing an Ultrasonic Sensor (HC-SR04) with Arduino via UART (Serial Communication)

The **HC-SR04 ultrasonic sensor** is commonly used to measure distances by emitting ultrasonic waves and measuring the time it takes for them to bounce back. When using **UART (Serial Communication)**, we can send the distance data to an external system like a **PC** via the **Serial Monitor** or **another microcontroller**.

Components Required:

- **Arduino board** (e.g., Arduino Uno)
- **HC-SR04 Ultrasonic Sensor**
- **Jumper wires**
- **Breadboard**



Wiring the HC-SR04 Ultrasonic Sensor to Arduino:

- **VCC → 5V**
- **GND → GND**
- **Trig → Digital Pin 12**
- **Echo → Digital Pin 2**

The **Trig** pin sends the pulse, and the **Echo** pin receives the reflected pulse, which is then used to calculate the distance.

Arduino Code: Interfacing HC-SR04 with UART

This example shows how to use the **HC-SR04 ultrasonic sensor** to measure distance and send the data via **Serial / UART** to the **Serial Monitor**.

```
const int trigPin = 12;  // Pin connected to the Trigger pin of the HC-SR04
const int echoPin = 2; // Pin connected to the Echo pin of the HC-SR04
long duration;
int distance;

void setup() {
  Serial.begin(9600);       // Start serial communication at 9600 baud
  pinMode(trigPin, OUTPUT); // Set the Trig pin as an output
  pinMode(echoPin, INPUT);  // Set the Echo pin as an input
}

void loop() {
  // Send a pulse to trigger the sensor
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);  // Wait for a short time to stabilize
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10); // Send a 10-microsecond pulse
  digitalWrite(trigPin, LOW);

  // Measure the time it takes for the pulse to return
  duration = pulseIn(echoPin, HIGH);  // Measure the duration of the pulse

  // Calculate the distance in centimeters
  distance = duration * 0.034 / 2;
// Speed of sound is 0.034 cm/us, divide by 2 to account for travel time

  // Print the distance to the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);  // Print the distance value
  Serial.println(" cm");   // Print "cm" to denote units

  delay(1000);  // Wait 1 second before the next measurement
}
```

Protocol	Experiment	Where the Protocol is Used?
UART (Serial)	**Ultrasonic Sensor (HC-SR04)**	- The **sensor transmits distance data** to the **Raspberry Pi/Arduino** via **UART (TX/RX)**. - The microcontroller sends a **trigger pulse** and receives **distance data** in the form of a **serial response**. - The distance measurement is calculated and displayed via the Serial Monitor.

Example Serial Monitor Output:

```
Distance: 25 cm
Distance: 30 cm
Distance: 45 cm
...
```

Conclusion:

By using **UART/Serial communication**, the Arduino can send real-time distance data from the **HC-SR04 ultrasonic sensor** to the **Serial Monitor**.

Arduino – Interfacing with I2C - LCD Module (16x2)

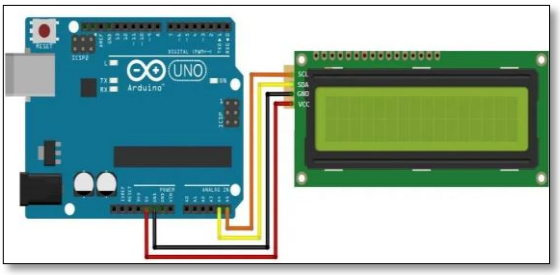
The **16x2 LCD Module** is a commonly used display module to show alphanumeric data. By using **I2C communication**, we can connect the LCD to the **Arduino** with just **two wires** for data transfer (SDA and SCL), making it much easier to interface than directly connecting each of the 16 pins on a standard LCD.

Components Required:

- **Arduino board** (e.g., Arduino Uno)
- **16x2 I2C LCD Module**
- **Jumper wires**
- **Breadboard (optional)**

Wiring the 16x2 LCD (I2C) to Arduino:

LCD Pin	Arduino Pin
GND	**GND**
VCC	**5V**
SDA (Data)	**A4 (Pin 27 on Uno)**
SCL (Clock)	**A5 (Pin 28 on Uno)**



Arduino Code: Interfacing with I2C 16x2 LCD

To use the **I2C LCD module**, we need to include the **Wire library** (for I2C communication) and the **LiquidCrystal_I2C library** (to control the LCD module).

Arduino Code:

```
#include <Wire.h>                // Include the Wire library for I2C communication
#include <LiquidCrystal_I2C.h>    // Include the LCD library for I2C LCD

// Create an LCD object, specifying the I2C address (0x27 is common) and the number
of columns and rows
LiquidCrystal_I2C lcd(0x27, 16, 2);  // I2C address 0x27, 16 columns, 2 rows

void setup() {
  lcd.begin(16, 2);  // Initialize the LCD with 16 columns and 2 rows
  lcd.backlight();   // Turn on the backlight

  // Print text to the LCD
  lcd.setCursor(0, 0);           // Set cursor to the first row, first column
  lcd.print("Hello, Arduino!");  // Print the message

  lcd.setCursor(0, 1);           // Set cursor to the second row
  lcd.print("I2C LCD Example");  // Print another message
}

void loop() {
  // Nothing to do in the loop as the text is static
}
```

Protocol	Experiment	Where the Protocol is Used?
I2C	**16x2 LCD Module (I2C)**	- The **Raspberry Pi/Arduino** communicates with the LCD module using **I2C protocol**. - **SDA (Data Line)** and **SCL (Clock Line)** are used to send and receive data. - I2C allows sending **text commands** to display characters on the LCD screen.

OUTPUT :

The LCD should now display:

```
Hello, Arduino!
I2C LCD Example
```

Conclusion:

This example demonstrates how to use an **I2C 16x2 LCD module** with Arduino to display text with minimal wiring and code. Using **I2C** communication reduces the number of connections needed to control the display, which makes interfacing easy.

Arduino – Interfacing with SPI ADC (Analog to Digital Converter)

- An **Analog-to-Digital Converter (ADC)** is used to convert an **analog signal** (such as a voltage from a sensor) into a **digital value** that can be processed by a microcontroller like **Arduino**.
- In this example, we will interface the **MCP3008**, a **10-bit** ADC, with the **Arduino** to read analog values and convert them into digital data.
- We will also use **SPI communication** to connect the ADC to the Arduino.

Components Required:

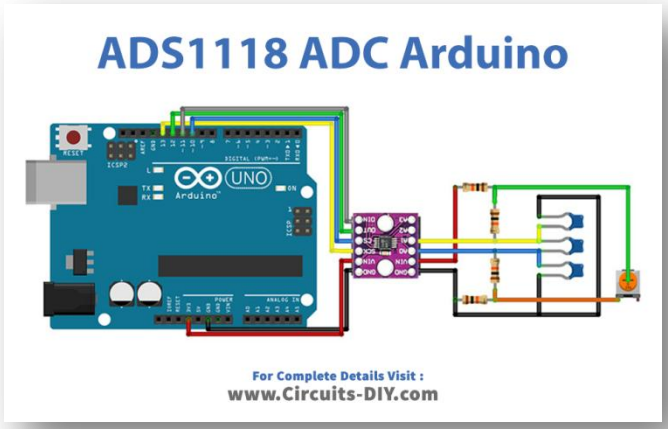
- **Arduino board** (e.g., Arduino Uno)
- **MCP3008 ADC module**
- **Jumper wires**
- **Breadboard (optional)**

MCP3008 ADC Overview:

- The **MCP3008** is a **10-bit** ADC that communicates with Arduino over **SPI**.
- It has **8 channels** (A0 to A7) for analog inputs, meaning you can read 8 different analog sensors with just one ADC module.

Wiring the MCP3008 to Arduino (SPI Communication):

MCP3008 Pin	Arduino Pin
VDD	**5V**
VREF	**5V**
AGND	**GND**
DGND	**GND**
DIN (MOSI)	**Pin 11** (MOSI)
DOUT (MISO)	**Pin 12** (MISO)
CLK	**Pin 13** (SCK)
CS/SHDN	**Pin 10** (SS)



Arduino Code: Reading Analog Data from MCP3008 Using SPI

The following code will allow us to read analog values from the **MCP3008** using **SPI communication**.

Arduino Code:

```
#include <SPI.h>

const int CS_PIN = 10;  // Chip Select pin for MCP3008

void setup() {
  Serial.begin(9600);       // Start the Serial Monitor
  pinMode(CS_PIN, OUTPUT);  // Set CS pin as output
  SPI.begin();              // Initialize SPI communication
}

void loop() {
  int value = readADC(0);   // Read the analog value from channel 0
  Serial.println(value);    // Print the value to Serial Monitor
  delay(1000);              // Wait 1 second before the next reading
}

int readADC(int channel) {
  digitalWrite(CS_PIN, LOW);                // Enable the ADC
  SPI.transfer(1);                          // Start bit
  SPI.transfer((channel << 4) | 0x80);      // Send channel address
  int value = (SPI.transfer(0) & 0x03) << 8 | SPI.transfer(0);
// Combine high and low byte
  digitalWrite(CS_PIN, HIGH);               // Disable the ADC
  return value;                             // Return the 10-bit value
}
```

Protocol	Experiment	Where the Protocol is Used?
SPI	**Analog to Digital Converter (ADC - MCP3008)**	- The **ADC (MCP3008)** communicates with the **Raspberry Pi/Arduino** via **SPI protocol**. - **MOSI (Data Output), MISO (Data Input), SCK (Clock), and CS (Chip Select)** are used for communication. - The ADC reads **analog sensor data** (e.g., temperature, potentiometer) and converts it to **digital values** for processing.

Example Serial Monitor Output:

```
512
345
678
...
```

Conclusion:

This example demonstrates how to interface an **SPI-based ADC** (like the **MCP3008**) with an **Arduino** to read analog values.

Interfacing an Ultrasonic Sensor (HC-SR04) with Raspberry Pi via UART (Serial Communication)

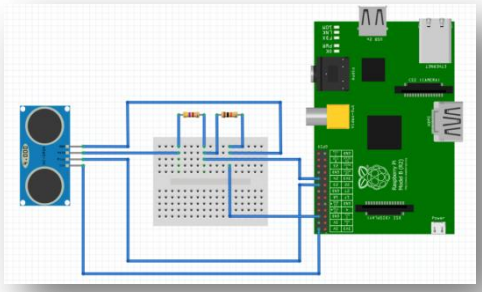
- The **HC-SR04 Ultrasonic Sensor** is widely used to measure distance by emitting ultrasonic waves and measuring the time it takes for them to bounce back.
- You can interface this sensor with a **Raspberry Pi** using **UART (Serial Communication)**.
- This can be useful when you want to communicate distance data over a serial connection or use the data for other applications.

Components Required:

- **Raspberry Pi** (any model with GPIO pins)
- **HC-SR04 Ultrasonic Sensor**
- **Jumper wires**
- **Breadboard (optional)**

Wiring the HC-SR04 to Raspberry Pi:

HC-SR04 Pin	Raspberry Pi Pin
VCC	5V
GND	GND
Trig	GPIO 23 (Pin 16)
Echo	GPIO 24 (Pin 18)



Important Note:

- **Echo Pin** operates at **3.3V** logic for the Raspberry Pi, but the **HC-SR04** returns **5V** from the Echo Pin, which can damage the Raspberry Pi's GPIO. Use a **voltage divider** or a **level shifter** to bring the voltage down to **3.3V** for the Echo pin.

Using a Voltage Divider for Echo Pin:

To avoid damaging the Raspberry Pi, use a **voltage divider** with two resistors to convert the **5V signal from Echo** to **3.3V**. Here's a simple **resistor** configuration:

- **Resistor 1 (R1)**: 1kΩ
- **Resistor 2 (R2)**: 2kΩ

Connect them as follows:

- **Echo Pin → R1 (1kΩ)** → GPIO Pin 24 (Raspberry Pi)
- **R2 (2kΩ) → GND**

Python Code: Reading Distance from HC-SR04 using UART on Raspberry Pi

We'll use the **GPIO** library for Python to control the **Trig** pin and read the **Echo** pin. The **time library** will be used to measure the pulse duration, which is proportional to the distance.

Python Code Example:

```
import RPi.GPIO as GPIO
import time

# Set GPIO mode
GPIO.setmode(GPIO.BCM)

# Define pins
TRIG = 23
ECHO = 24

# Set up the pins
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Send a pulse to start the measurement
GPIO.output(TRIG, GPIO.LOW)
time.sleep(2)

while True:
    GPIO.output(TRIG, GPIO.HIGH)
    time.sleep(0.00001)  # 10us pulse
    GPIO.output(TRIG, GPIO.LOW)

    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()

    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()

    # Calculate distance
    distance = round((pulse_end - pulse_start) * 17150, 2)
    print(f"Distance: {distance} cm")

    time.sleep(1)  # Wait for 1 second

# Cleanup GPIO pins when finished
GPIO.cleanup()
```

Protocol	Experiment	Where the Protocol is Used?
UART (Serial)	**Ultrasonic Sensor (HC-SR04)**	- The **sensor transmits distance data** to the **Raspberry Pi/Arduino** via **UART (TX/RX)**. - The microcontroller sends a **trigger pulse** and receives **distance data** in the form of a **serial response**. - The distance measurement is calculated and displayed via the Serial Monitor.

Example Output:

```
Distance: 25.34 cm
Distance: 30.12 cm
Distance: 35.45 cm
...
```

Conclusion:

In this example, we interfaced the **HC-SR04 Ultrasonic Sensor** with the **Raspberry Pi** using **GPIO pins** to trigger the sensor and read the echo pulse. The distance is then calculated and printed on the terminal. The Python code allows the Raspberry Pi to act as the **master device**, sending out signals and receiving distance data via the **ECHO pin**.

Interfacing I2C 16x2 LCD Module with Raspberry Pi

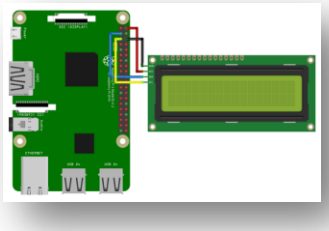
- The **16x2 LCD Module** is commonly used for displaying text-based information.
- Using **I2C communication** reduces the number of wiring connections needed to interface the LCD with the **Raspberry Pi**, making the setup much simpler.
- In this guide, we'll interface a **16x2 LCD** with **I2C** communication on the **Raspberry Pi**.

Components Required:

- **Raspberry Pi** (any model with GPIO pins)
- **16x2 I2C LCD Module**
- **Jumper wires**
- **Breadboard (optional)**

Wiring the 16x2 I2C LCD to Raspberry Pi:

LCD Pin	Raspberry Pi Pin
VCC	**5V**
GND	**GND**
SDA (Data)	**GPIO 2 (Pin 3)**
SCL (Clock)	**GPIO 3 (Pin 5)**



Python Code: Display Text on I2C LCD Module

We will use the **`smbus`** library to communicate with the LCD via **I2C**.

```
import smbus
import time

# Set up I2C bus
bus = smbus.SMBus(1)  # 1 for Raspberry Pi

# LCD I2C address (most common is 0x27)
address = 0x27

# Define LCD commands
LCD_CMD = 0x80  # Command register
LCD_DATA = 0x40  # Data register

# Initialize the LCD
def lcd_init():
    bus.write_byte_data(address, LCD_CMD, 0x33)
    bus.write_byte_data(address, LCD_CMD, 0x32)
    bus.write_byte_data(address, LCD_CMD, 0x06)
    bus.write_byte_data(address, LCD_CMD, 0x0C)
    bus.write_byte_data(address, LCD_CMD, 0x01)
    time.sleep(0.05)

# Write data to the LCD
def lcd_write(value, mode):
    bus.write_byte_data(address, mode, value)
    time.sleep(0.001)

# Display string on the LCD
def lcd_display_string(message):
    for char in message:
        lcd_write(ord(char), LCD_DATA)

# Main program
lcd_init()
lcd_display_string("Hello, Raspberry Pi!")
```

Protocol	Experiment	Where the Protocol is Used?
I2C	**16x2 LCD Module (I2C)**	- The **Raspberry Pi/Arduino** communicates with the LCD module using **I2C protocol**. - **SDA (Data Line)** and **SCL (Clock Line)** are used to send and receive data. - I2C allows sending **text commands** to display characters on the LCD screen.

OUTPUT :

You should now see **"Hello, Raspberry Pi!"** displayed on the LCD.

Conclusion:

This example shows how to interface a **16x2 I2C LCD** with the **Raspberry Pi** and display text on it using **Python**. The use of **I2C** makes the wiring and communication simple, requiring only two data pins for communication instead of multiple pins used by a parallel interface LCD.

Interfacing SPI ADC (Analog to Digital Converter) with Raspberry Pi

- An **Analog-to-Digital Converter (ADC)** converts **analog signals** into **digital values** that a **Raspberry Pi** can process.
- The **SPI (Serial Peripheral Interface)** protocol is commonly used to interface with external ADC modules, allowing the Raspberry Pi to read analog values from sensors or other devices.
- In this example, we will interface an **MCP3008** ADC with the **Raspberry Pi** using **SPI communication** to read analog data.

Components Required:

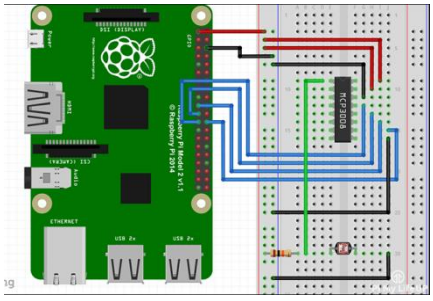
- **Raspberry Pi** (any model with SPI support)
- **MCP3008 ADC Module** (10-bit ADC with 8 channels)
- **Jumper wires**
- **Breadboard (optional)**

MCP3008 ADC Overview:

- The **MCP3008** is an **8-channel, 10-bit ADC** that can be used to convert up to 8 analog inputs to digital values.
- The **MCP3008** communicates with the Raspberry Pi via the **SPI protocol**.

Wiring the MCP3008 to Raspberry Pi (SPI Communication):

MCP3008 Pin	Raspberry Pi Pin
VDD (Pin 16)	3.3V (Pin 1)
VREF (Pin 15)	3.3V (Pin 1)
AGND (Pin 14)	GND (Pin 6)
CLK (Pin 13)	GPIO 11 (Pin 23/SCLK)
DOUT (Pin 12)	GPIO 9 (Pin 21/MISO)
DIN (Pin 11)	GPIO 10 (Pin 19/MOSI)
CS (Pin 10)	GPIO 8 (Pin 24/CE0)
DGND (Pin 9)	GND (Pin 6)



Python Code: Reading Analog Data from MCP3008 Using SPI

We'll use the **`spidev`** library to communicate with the MCP3008 via SPI

```
import spidev
import time

# Create SPI object and set the bus and device
spi = spidev.SpiDev()
spi.open(0, 0)  # Bus 0, Device 0 (on Raspberry Pi)

# Set SPI speed and mode
spi.max_speed_hz = 50000  # 50kHz
spi.mode = 0b00           # SPI mode 0

# Function to read ADC data from MCP3008
def read_adc(channel):
    if channel < 0 or channel > 7:
        return -1  # Channel out of range (0-7)

    # Send the start bit, channel address, and padding bits
    command = [1, (8 + channel) << 4, 0]
    # Send and receive the data over SPI
    response = spi.xfer2(command)

    # Convert the response into a 10-bit value
    result = ((response[1] & 3) << 8) + response[2]
    return result

# Main program
try:
    while True:
        adc_value = read_adc(0)  # Read from channel 0
        voltage = (adc_value / 1023.0) * 3.3  # Convert ADC value to voltage
(0-3.3V range)
        print(f"ADC Value: {adc_value}, Voltage: {voltage:.2f}V")
        time.sleep(1)  # Wait for 1 second before next reading

except KeyboardInterrupt:
    spi.close()  # Close SPI connection on Ctrl+C
```

Protocol	Experiment	Where the Protocol is Used?
SPI	**Analog to Digital Converter (ADC - MCP3008)**	- The **ADC (MCP3008)** communicates with the **Raspberry Pi/Arduino** via **SPI protocol**. - **MOSI (Data Output), MISO (Data Input), SCK (Clock), and CS (Chip Select)** are used for communication. - The ADC reads **analog sensor data** (e.g., temperature, potentiometer) and converts it to **digital values** for processing.

Example Output:

```
ADC Value: 512, Voltage: 1.65V
ADC Value: 256, Voltage: 0.83V
ADC Value: 1023, Voltage: 3.30V
...
```

Conclusion:

This example demonstrates how to interface the **MCP3008 SPI ADC** with the **Raspberry Pi** to read analog signals, convert them to digital values, and display the corresponding voltages. The use of **SPI** allows for efficient communication with the ADC, and the **spidev** library simplifies the process.

Programming with Python on Raspberry Pi – Interfacing External Gadgets

Overview

- **Raspberry Pi** can interface with various external gadgets like **LEDs, sensors, buzzers, motors, and displays**.
- Uses the **RPi.GPIO** library to control **GPIO (General Purpose Input/Output) pins**.
- Can read data from input devices (**sensors, buttons**) and control output devices (**LEDs, motors, buzzers**).

---

1. Setting Up Raspberry Pi for GPIO Programming

- Ensure **Python** and **RPi.GPIO** library are installed.
- Run the following command to install the library (if not already installed):
- `sudo apt-get update`
- `sudo apt-get install python3-rpi.gpio`

---

2. Controlling an Output Device (LED Blinking Example)

Wiring:

- **GPIO Pin 18 → LED → Resistor (330Ω) → Ground (GND)**

Python Code:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)  # Use Broadcom GPIO numbering
GPIO.setup(18, GPIO.OUT)  # Set GPIO 18 as output

try:
    while True:
        GPIO.output(18, GPIO.HIGH)  # Turn LED on
        time.sleep(1)  # Wait 1 second
        GPIO.output(18, GPIO.LOW)  # Turn LED off
        time.sleep(1)  # Wait 1 second
except KeyboardInterrupt:
    GPIO.cleanup()  # Reset GPIO on exit
```

 Explanation:

- **GPIO 18** is set as an output pin.
- The program blinks an LED with a **1-second delay**.
- **CTRL+C** stops execution and resets GPIO pins.

---

3. Reading Input from a Button

Wiring:

- **GPIO Pin 17 → Push Button → Ground (GND)**

Python Code:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Set GPIO 17 as input with pull-up resistor

try:
    while True:
        button_state = GPIO.input(17)  # Read button state
        if button_state == GPIO.LOW:
            print("Button Pressed!")
        time.sleep(0.2)  # Debounce delay
except KeyboardInterrupt:
    GPIO.cleanup()
```

 Explanation:

- **GPIO 17** is set as an **input pin** with a **pull-up resistor** to prevent floating states.
- If the button is pressed, it prints **"Button Pressed!"**.

---

4. Controlling an External Buzzer Based on Input

Wiring:

- **GPIO 18 → Buzzer → GND**
- **GPIO 17 → Button → GND**

Python Code:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)  # Buzzer Output
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)  #
Button Input

try:
    while True:
        if GPIO.input(17) == GPIO.LOW:
            GPIO.output(18, GPIO.HIGH)  # Turn on
Buzzer
            print("Buzzer ON")
        else:
            GPIO.output(18, GPIO.LOW)  # Turn off
Buzzer
        time.sleep(0.2)
except KeyboardInterrupt:
    GPIO.cleanup()
```

 Explanation:

- If **button is pressed**, the **buzzer turns ON**.
- If **button is released**, the **buzzer turns OFF**.

---

5. Reading Data from an Ultrasonic Sensor (HC-SR04) for Distance Measurement

Wiring:

- **VCC → 5V**
- **GND → GND**
- **Trig → GPIO 23**
- **Echo → GPIO 24**

Python Code:

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins
TRIG = 23
ECHO = 24

GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

def measure_distance():
    GPIO.output(TRIG, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(TRIG, GPIO.LOW)

    while GPIO.input(ECHO) == 0:
        start_time = time.time()

    while GPIO.input(ECHO) == 1:
        stop_time = time.time()

    time_elapsed = stop_time - start_time
    distance = (time_elapsed * 34300) / 2  # Convert to cm
    return round(distance, 2)

try:
    while True:
        dist = measure_distance()
        print(f"Distance: {dist} cm")
        time.sleep(1)
except KeyboardInterrupt:
    GPIO.cleanup()
```

 Explanation:

- Sends an **ultrasonic pulse** and calculates the **time taken for the echo** to return.
- Converts time to **distance (in cm)** using **speed of sound formula**.

---

6. Controlling a Servo Motor

Wiring:

- **VCC → 5V**
- **GND → GND**
- **Signal → GPIO 25**

Python Code:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)

servo = GPIO.PWM(25, 50)  # Set GPIO 25 as PWM with
50Hz frequency
servo.start(0)

try:
    while True:
        servo.ChangeDutyCycle(2)  # 0-degree position
        time.sleep(1)
        servo.ChangeDutyCycle(7)  # 90-degree position
        time.sleep(1)
        servo.ChangeDutyCycle(12)  # 180-degree
position
        time.sleep(1)
except KeyboardInterrupt:
    servo.stop()
    GPIO.cleanup()
```

 Explanation:

- Uses **PWM (Pulse Width Modulation)** to rotate the **servo motor to 0°, 90°, and 180°**.
- Adjusts **duty cycle** to change the servo angle.

Summary of Interfacing External Gadgets with Raspberry Pi

Device	GPIO Mode	Example Application
LED	Output	Blinking indicator
Button	Input	User input detection
Buzzer	Output	Alert system
Ultrasonic Sensor (HC-SR04)	Input/Output	Distance measurement
Servo Motor	PWM Output	Robotic arm movement