❖ IOT Question Bank Solution

Unit-1 Introduction to Internet of Things

1) Define the Internet of Things.

- The Internet of Things (IoT) refers to a network of physical objects that are connected to the Internet so that they can exchange data and information in order to improve productivity, efficiency, services, and more.
- IoT is a sensor network of billions of smart devices that connect people, systems and other applications to collect and share data.
- The IoT includes a wide range of devices, such as smart appliances, wearable devices, industrial sensors, and smart home systems, among others

2) List the application of IoT.

1. Home
2. Cities
3. Environment
4. Energy
5. Retail
6. Logistics
7. Agriculture
8. Industry
9. Health & LifeStyle

3) Discuss the characteristics of IoT.

1) **Connectivity**: IoT devices are connected to the internet or other networks, enabling them to communicate with each other and with other systems.
2) **Sensors and Actuators:** IoT devices are equipped with sensors to collect data from their environment and actuators to take actions based on that data.
3) **Data processing:** IoT devices can process data locally or send it to a remote server for processing.
4) **Interoperability:** IoT devices can work with other devices and systems, regardless of their manufacturer or technology.
5) **Scalability:** IoT systems can scale up or down easily, depending on the needs of the application.
6) **Security:** IoT systems require robust security mechanisms to protect against unauthorized access and data breaches.

7) **Autonomous operation:** Some IoT devices can operate autonomously without human intervention, using artificial intelligence and machine learning algorithms to make decisions and take actions based on data.

4) What are the “things” in IoT?

- The "things" in the Internet of Things (IoT) refer to the wide range of physical objects or devices that are embedded with sensors, software, and connectivity capabilities, enabling them to collect, exchange, and act upon data.
- The “Thing” in a network can be monitored/measure.
  For example, a temperature sensor could be a thing.
- Things are capable of exchanging data with other connected devices in the system
- The data could be stored in a centralized server (or cloud), processed there and a control action could be initiated
- Some of the famous “Things” are temperature sensors, pressure sensors, humidity sensors, etc.
- The data from these sensors are collected and sent it to the cloud or stored it in local server for data analysis.
- Based on the data analysis, the control action would be taken.

5) Explain IoT Stack in detail.

1. **Physical layer :-**
   - This layer is considered about Physical components, which mainly includes sensors and actuators.
   - which are core components and it is responsible for data collection and action execution.
   - Eg :- Temperature Sensor, Humidity Sensor etc
   - Function of layer :
     - Action execution (via Actuators)
     - Sensing data (via Sensors)

2. **Processing and control layer :-**
   - This layer is responsible for the data action layer.
   - Microcontroller/Processor and operating system play vital roles at this layer.
   - Various development kits can be used for this purpose such as Arduino, Node MCU (based on ESP32 or ESP8266), ARM, PIC etc.
   - Data collected from the sensors is processed in this layer.

3. **Hardware Interface layer :-**
   - The 3rd layer in the stack is the Hardware Interface Layer. It connects the components.
   - Hardware components and communication standards such as RS232, CAN, SPI, SCI, I2C, etc. occupy this layer.

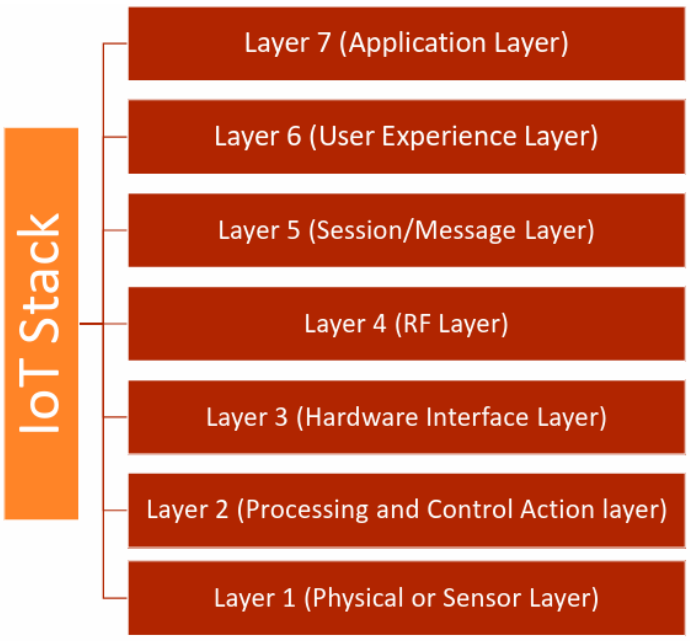
- All these components ensure flawless communication

4. **RF Layer :-**
   - It plays a major role in the communication channel– whether it is short range or long range.
   - Protocols used for communication and transport of data based on RF are listed in this layer.
   - Some famous and common protocols are Wi-Fi, NFC, Bluetooth, Zig-bee, etc.
   - RF layer does communication of data using radio frequency based Electromagnetic (EM) waves

5. **Session/Message Layer :**
   - This layer deals with various messaging protocols such as MQTT, CoAP, HTTP, FTP (or Secured FTP), SSH etc.
   - It defines how messages are broadcasted to the cloud.

6. **User experience layer :**
   - This layer deals with providing the best experience to the end users of IoT products.
   - To fulfill this, this layer takes care of rich UI designs with lots of features.
   - Various languages and tools are developed for the design of GUI interface softwares.

7. **Application layer :**
   - This layer utilizes the rest of the six layers in order to develop the desired application



6) List the IoT Layers.

1. Perception Layer
2. Network Layer
3. Middleware Layer
4. Application Layer
5. Business Layer

7) With neat diagrammatic representation, explain the IoT stack with appropriate examples for each layer.

1. **Perception Layer:** This layer is responsible for collecting data from the physical environment using sensors and actuators. Examples of devices in this layer include:
    a. Temperature sensors: Collect temperature data for smart thermostats in homes.
    b. GPS modules: Provide location data for tracking devices or navigation systems.

2. **Network Layer:** The network layer facilitates communication between IoT devices and enables data transmission over various communication protocols. Examples include:
    a. Wi-Fi: Allows smart home devices like smart speakers to connect to the internet.
    b. Bluetooth: Enables communication between wearable fitness trackers and smartphones.

3. **Middleware Layer:** This layer acts as an intermediary between the perception layer and the application layer, providing services such as data processing and device management. Examples of middleware components include:
    a. IoT platforms: Offer services for device management, data analytics, and application development, such as AWS IoT, Google Cloud IoT, or Microsoft Azure IoT Hub.

4. **Application Layer**: The application layer encompasses the software applications and services that leverage IoT data to provide value-added functionalities and insights. Examples include:
    a. Smart home automation apps: Control smart lights, thermostats, and security cameras from a smartphone app.
    b. Healthcare management platforms: Track patient vitals remotely and provide alerts for medical emergencies.

5. **Business Layer:** This layer focuses on the business logic, rules, and policies governing IoT deployments. Examples of functionalities in this layer include:

a. Data analytics and decision-making algorithms: Analyze IoT data to optimize resource allocation, predict equipment failures, or improve operational efficiency.
b. Security management: Implement security measures to protect IoT devices and data from cyber threats and ensure compliance with regulations such as GDPR or HIPAA.

8) Elaborate Enabling Technology in IoT.

Enabling technologies are technologies that help other technologies improve their capabilities. In the Internet of Things (IoT), enabling technologies help IoT systems function. Some examples of enabling technologies in IoT include:

- Communication technologies: These technologies allow IoT devices to connect to each other and to the internet. Examples of communication technologies include Bluetooth, Wi-Fi, and cellular networks.
- Wireless sensor networks: These networks allow IoT devices to collect data and send it to other devices.
- Big data analytics: This technology helps IoT systems to analyze large amounts of data.
- Machine learning: This technology helps IoT systems to learn from data and make decisions.
- Artificial intelligence: This technology helps IoT systems to understand and respond to their environment.

9) What is the role of different protocols in IoT?

- IoT protocols are standards that dictate how data is transmitted between electronic devices, facilitating communication between the Internet and edge devices.
- These protocols enable device-to-device, device-to-gateway, or device-to-cloud/data center communication, or combinations thereof.
- Factors like geographic and special location, power consumption needs, presence of physical barriers, and cost influence the choice of protocol in an IoT deployment.
- IoT protocols fall into two main categories:
    - **IoT data protocols (User Interface layer / Application layers):** Focus on information exchange between devices and applications.
    - **Network protocols for IoT (Datalink layers (RF layer) / Physical layers):** Provide methods for connecting IoT edge devices with each other or the Internet.
- Data protocols handle data exchange, while network protocols facilitate connectivity between IoT devices and other devices or the Internet.

Explain IoT Levels.

Level 1:-

- It is of minimal complexity.
- The application has one sensor (temperature sensor, pressure sensor, etc).
- The data sensed is stored locally and the data analysis is done locally.
- Monitoring / control is done through an application.
- This is used for simple applications.
- Data generated in this level application is not huge.
- For example, a temperature sensor senses the room temperature and the data is stored and analysed locally.

Level 2:-

- This level consists of air conditioner, temperature sensor, Big data (Bigger than level - 1, data analysis done here) , cloud and control & monitoring app.
- This level-2 is complex compare to level-1. Moreover rate of sensing is faster compare to level-1.
- This level has voluminous size of data. Hence cloud storage is used.
- Data analysis is carried out locally. Cloud is used for only storage purpose.
- Based on data analysis, control action is triggered using web app or mobile app.
- Examples: Agriculture applications, room freshening solutions based on odour sensors etc.

Level 3:-

- The data is huge, frequency of sensing done by the sensor is faster and the data is stored on cloud.
- The difference is that the analysis is also carried out on cloud.
- Based on the data analysis, the control action can be triggered through the web application or mobile application.
- Some examples are agriculture applications, room freshening solutions based on odour.

Level 4:-

- This level consists of multiple sensors, data collection and analysis and control & monitoring app.
- At this level-4, multiple sensors are used which are independent of the others.
- The data collected using these sensors are uploaded to the cloud separately.
- The cloud storage is used in this level due to requirement of huge data storage
- The data analysis is performed on the cloud and based on which control action is triggered either using web app or mobile app.

Level 5:-

- This level consists of multiple sensors, coordinator node, data collection and analysis and control & monitoring app.

- This level is similar to level-4 which also has huge data and hence they are sensed using multiple sensors at much faster rate and simultaneously.
- The data collection and data analysis is performed at the cloud level.
- Based on analysis,control action is performed using mobile app or web app.

11) How does the classification of IoT as Level 1 to 5 happen? Explain.

No Idea

12) Enumerate the role of cloud in IoT.

Cloud computing plays an important role in the Internet of Things (IoT) by providing storage and computing power for IoT applications, and by helping to store and analyze data.

IoT devices produce a large amount of data, and cloud storage is a cost-effective way to store and manage it.

his storage allows devices to stay connected and exchange data in real time. Cloud computing also offers other benefits for IoT applications, such as:

- Scalability: Cloud services can handle large amounts of data from many devices and adjust to changing demands.
- Flexibility: Cloud computing offers a range of cloud services and platforms specifically designed for IoT ecosystems.
- Security: Cloud computing offers security benefits.
- Improved decision-making: Enterprises can use cloud platforms to improve decision-making processes.
- Increased operational efficiency: Enterprises can use cloud platforms to maximize operational efficiency.

13) Explain the importance of communication protocols when it comes to IoT.

No Idea

14) Discuss any three IoT network protocol.

Bluetooth:

- Bluetooth is a wireless technology that allows data to be exchanged in small amounts over short distances at a high speed.
- Its ability to transfer data faster comes at the cost of relatively high power consumption
- In 2010, Bluetooth was optimized for short-range IoT communications. Known as BLE, this version of the connectivity protocol boasts greater energy efficiency. However, their data transfer speed is also slower compared to Bluetooth.

ZigBee:

- ZigBee is a robust and scalable IoT protocol utilized for gathering sensor data in both home automation and industrial applications.
- It excels in transferring small amounts of data over moderate distances.
- Operating on a self-healing mesh topology, ZigBee ensures high reliability in communication.
- Additionally, new devices can seamlessly join the network after performing a "handshake" process, which typically takes a mere 30 milliseconds.

Wi-Fi:

- Wi-Fi utilizes Internet Protocol (IP) to connect devices on a Local Area Network (LAN).
- It ensures reliable and secure communication between closely located devices, such as gadgets within one IoT application deployment like smart homes.
- This IoT protocol is relatively cheap and easy to implement.
- It works well with heavy files and can handle vast amounts of data.
- However, this IoT communication protocol is too power-consuming and has inherent coverage limitations.

15) Compare Bluetooth and ZigBee protocol.

Bluetooth	ZigBee
The frequency range supported in Bluetooth vary from 2.4 GHz to 2.483 GHz.	While the frequency range supported in Zigbee mostly 2.4 GHz worldwide.
There are seventy nine RF channels in Bluetooth.	There are sixteen RF channels in zigbee.
There is maximum of 8 cell nodes in Bluetooth.	While there is more than sixty five thousand (65000) cell nodes in zigbee.
Bluetooth requires low bandwidth.	While zigbee also requires low bandwidth but greater than Bluetooth's bandwidth most of time.
Bluetooth batteries may be recharged.	Although ZigBee batteries cannot be recharged, they last longer
A network speed of up to 250 megabits per second.	A network speed of up to 1 megabit per second.

16) List Advantages and Disadvantages of Bluetooth.

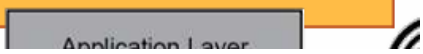
Advantages:	Disadvantages:
Low latency	Uses the crowded 2.4 GHz frequency
Lower implementation costs due to hardware simplicity	Might not be suitable for large files
Easy internet access via a smartphone	A limited number of connected devices
Data encryption by default	

17) List Advantages and Disadvantages of Zigbee.

Advantages:	Disadvantages:
Can accommodate up to 65,000 devices	Uses the common 2.4 GHz frequency, which is prone to interferences
Low power consumption (small devices can operate on one battery for several years)	Requires a custom gateway, which is expensive
Relatively long range of communication	

18) List Advantages and Disadvantages of LoRa protocol.

Advantages:	Disadvantages:
Scalability	Low data transfer rate
Covers large distances	Custom LoRa gateway
Low power consumption	Not suitable for real-time applications
Operates on unlicensed frequency	



19) List Advantages and Disadvantages of Z wave.

Advantages:	Disadvantages:
Low latency	Low data transfer rate
Avoids the crowded 2.4 GHz frequency used by Wi-Fi, Bluetooth, and Zigbee	Premium prices
Low power consumption	
Reasonable coverage	

Unit 2: IoT Physical Devices and Endpoints

20) Compare IoT and M2M.

IOT	M2M
Devices have objects that are responsible for decision making	Some degree of intelligence is observed in this.
The connection is via Network and using various communication types.	The connection is a point to point
Internet protocols are used such as HTTP, FTP, and Telnet.	Traditional protocols and communication technology techniques are used
Internet connection is required for communication	Devices are not dependent on the Internet.
It supports cloud communication	It supports point-to-point communication.
Involves the usage of both Hardware and Software.	Mostly hardware-based technology
Business 2 Business(B2B) and Business 2 Consumer(B2C)	Business 2 Business (B2B)

22) What is Arduino? Explain in detail.

- Arduino is an open-source electronics platform based on easy-to-use hardware and software.
- Software : Arduino Software (IDE), based on Processing tool.
- Arduino programming language (based on Wiring)
- Hardware: Arduino Board can be instructed by sending a set of instructions to the microcontroller on the board.
- Arduino boards are able to read inputs likes light on a sensor, a finger on a button, or a twitter message and turn it into an output likes activating a motor, turning on an LED, publishing something online.
- Arduino is made up of a programmable circuit board and software that runs on your computer, called an Integrated Development Environment (IDE).
- With the IDE, you can upload or write code, work in real-time, and move your code to the cloud.
- Arduino libraries are small packages of code that somebody else wrote for a particular purpose.
- For example, a library can add support for a specific sensor, such as the BME280.

22) Give the pin description with example of Arduino UNO.

1. **Digital Pins (D0 - D13):**

- These pins can be configured as either inputs or outputs.
- Examples:
    - Input: Reading a digital sensor (e.g., pushbutton, switch).
    - Output: Controlling an LED, relay, or motor.

2. **Analog Pins (A0 - A5):**
    - These pins can read analog voltages ranging from 0 to 5 volts.
    - Examples:
        - Reading an analog sensor (e.g., temperature sensor, light sensor).
        - Outputting analog voltages using PWM (Pulse Width Modulation) for dimming an LED or controlling the speed of a motor.
3. **Power Pins:**
    - 5V: Provides a regulated 5-volt power supply.
    - 3.3V: Provides a regulated 3.3-volt power supply.
    - GND (Ground): Ground pins for connecting the ground of external components.
    - Vin: Input voltage to the board when using an external power source (e.g., battery or power adapter).
4. **Communication Pins:**
    - Serial (RX, TX): Pins used for serial communication (e.g., with a computer or other devices).
    - I2C (SDA, SCL): Pins used for I2C communication with compatible sensors and devices.
    - SPI (MISO, MOSI, SCK): Pins used for SPI communication with compatible sensors and devices.
5. **Reset Pin:**
    - RESET: Pin used to reset the microcontroller.

23) Which architecture ATMEL ATmega328 follows? Write down it's architectural feature.

Architecture Type: Modified Harvard architecture.

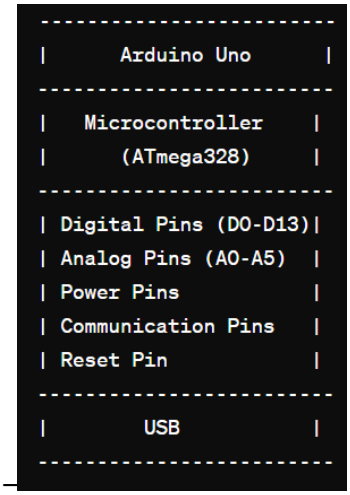
1. **Memory Organization**:
    - Separate memories for program (Flash) and data (SRAM).
    - 32 KB of Flash memory for program storage.
    - 2 KB of SRAM for data storage.
2. **CPU Core**:
    - 8-bit RISC (Reduced Instruction Set Computing) architecture.
    - Operating frequency of up to 20 MHz.
3. **Peripherals**:
    - GPIO (General Purpose Input/Output) pins for digital I/O operations.
    - Analog-to-digital converters (ADC) for converting analog signals to digital values.
    - Timers/counters for generating timing signals and counting external events.

- Pulse-width modulation (PWM) channels for generating analog-like output signals.

4. **Low-Power Modes**:
    - Supports low-power modes for energy-efficient operation.

24) What is developement board? Draw Block diagram of Arduino UNO and define each block.

- A development board is a hardware platform designed to facilitate the prototyping and development of electronic projects.
- It typically includes a microcontroller or microprocessor, input/output interfaces, power management circuitry, and often, additional components such as sensors, actuators, and communication modules.
  Here's a block diagram of the Arduino Uno development board:



- Microcontroller (ATmega328): The heart of the Arduino Uno, responsible for executing code and controlling external components.
- Digital Pins (D0-D13): General-purpose input/output pins for connecting to external devices.
- Analog Pins (A0-A5): Analog input pins for reading analog signals from sensors.
- Power Pins: Pins for providing power to external components, including 5V, 3.3V, and GND.
- Communication Pins: Pins for serial communication (RX, TX), I2C (SDA, SCL), and SPI (MISO, MOSI, SCK).
- Reset Pin: Used to reset the microcontroller.
- USB: Interface for programming the Arduino Uno and serial communication with a computer.

25) Write down all components of Arduino board

1. **Reset Button–** This will restart any code that is loaded to the Arduino board
2. **AREF–** Stands for “Analog Reference” and is used to set an external reference voltage

3. **Ground Pin–** There are a few ground pins on the Arduino and they all work the same
4. **Digital Input/Output–** Pins 0-13 can be used for digital input or output.
5. **PWM–** The pins marked with the (~) symbol can simulate analog output.
6. **USB Connection–** Used for powering up your Arduino and uploading sketches
7. **TX/RX–** Transmit and receive data indication LEDs
8. **ATmega Microcontroller–** This is the brains and is where the programs are stored.
9. **Power LED Indicator–** This LED lights up anytime the board is plugged in a power source
10. **Voltage Regulator–** This controls the amount of voltage going into the Arduino board
11. **DC Power Barrel Jack–** This is used for powering your Arduino with a power supply
12. **3.3V Pin–** This pin supplies 3.3 volts of power to your projects
13. **5V Pin–** This pin supplies 5 volts of power to your projects
14. **Ground Pins–** There are a few ground pins on the Arduino and they all work the same
15. **Analog Pins–** These pins can read the signal from an analog sensor and convert it to digital

26) Explain the code structure of basic Arduino Program.

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run
repeatedly:

}
```

setup() function

- Called only once, when Board starts.
- To initialize variables, pin modes, start using libraries, etc.
- It is called only when the Arduino is powered on or reset.

loop() function

- The loop functions runs continuously till the device is powered off.
- The main logic of the code goes here.
- Similar to while (1) for micro-controller programming.
- Code in the loop() section of the sketch is used to actively control the Arduino board.

Commenting

- Anyline that starts with two slashes (//) will not be read by the compiler

27) Define void setup() and void loop() function

setup() function

- Called only once, when Board starts.
- To initialize variables, pin modes, start using libraries, etc.
- It is called only when the Arduino is powered on or reset.

loop() function

- The loop functions runs continuously till the device is powered off.
- The main logic of the code goes here.
- Similar to while (1) for micro-controller programming.
- Code in the loop() section of the sketch is used to actively control the Arduino board.

Example:

```
void setup() {
  pinMode(ledPin, OUTPUT);
}

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

28) Write down an Arduino functions for Serial communication in detail.

1. **Serial.begin(baudRate)**:
   - This function initializes the serial communication with the specified baud rate.
   - Parameters:
     - `baudRate`: The baud rate (bits per second) at which data is transmitted and received. Common baud rates include 9600, 115200, etc.

Example:

cpp Copy code

```
void setup() {
  Serial.begin(9600); // Initialize serial communication at 9600 baud rate
}
```

2. **Serial.available()**:
   - This function returns the number of bytes (characters) available to read from the serial input buffer.
   - Returns:
     - The number of bytes available to read.

   Example:

```cpp
void loop() {
  if (Serial.available() > 0) {
    // Read data from serial input buffer
    char receivedChar = Serial.read();
    // Process the received data
  }
}
```

3. **Serial.read()**:
   - This function reads the next byte (character) from the serial input buffer.
   - Returns:
     - The next byte of data received via serial communication.

   Example:

```cpp
void loop() {
  if (Serial.available() > 0) {
    char receivedChar = Serial.read(); // Read a character from serial input buffer
    // Process the received character
  }
}
```

4. **Serial.write(data)**:
   - This function writes data to the serial output buffer for transmission.
   - Parameters:
     - `data`: The data to be transmitted, which can be a byte, character, or a string (array of characters).

   Example:

```cpp
void loop() {
  int sensorValue = analogRead(A0); // Read analog sensor value
  Serial.write(sensorValue); // Transmit sensor value over serial
  delay(1000); // Delay before sending the next reading
}
```

5. **Serial.print(data)** and **Serial.println(data)**:
   - These functions are used to print data to the serial port for debugging or monitoring purposes.
   - Parameters:
     - `data`: The data to be printed, which can be a byte, character, integer, float, or a string.

   Example:

```cpp
void loop() {
  float temperature = 25.5; // Sample temperature value
  Serial.print("Temperature: "); // Print label
  Serial.println(temperature); // Print temperature value followed by a newline
  delay(1000); // Delay before printing the next reading
}
```

29) What is the function of PWM pins?

- PWM (Pulse Width Modulation) pins on Arduino simulate analog output by adjusting the width of pulses in a square wave signal.
- Commonly used for con–trolling LED brightness, motor speed, and other tasks requiring analog-like control.
- Achieved by rapidly switching the output pin between HIGH and LOW states at different duty cycles.
- PWM pins produce an average voltage level proportional to the duty cycle, allowing for precise control over devices.
- Essential for tasks needing adjustable voltage or power levels without true analog output.

30) Explain functionalities of all the A0 to A5 pins.

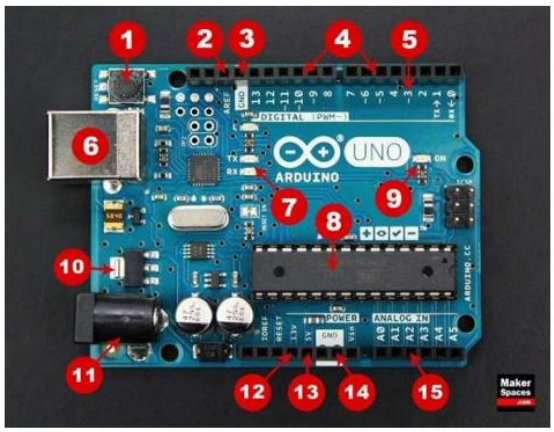
- Analog Input Pins: A0 to A5 are analog input pins, capable of reading analog voltages ranging from 0 to 5 volts.
- Resolution: These pins have a 10-bit analog-to-digital converter (ADC), allowing them to detect voltage levels with a resolution of 1024 (2^10).
- Input Range: The input voltage range is typically 0 to 5 volts, but some Arduino boards may support different voltage ranges (e.g., 0 to 3.3 volts).
- Usage: A0 to A5 are commonly used to interface with analog sensors such as temperature sensors, light sensors, potentiometers, and other devices that generate analog signals.
- Conversion: Analog signals are converted into digital values (0 to 1023) by the ADC, which can then be processed by the microcontroller

31) Explain functionalities of all the digital pins.

1. **Digital Input (D0 to D13)**: Used to read binary signals from external devices like switches or sensors.

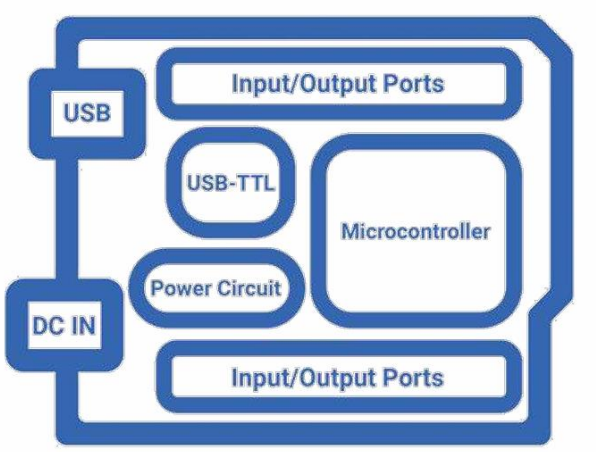
2. **Digital Output(D0 to D13)**: Controls external devices by producing binary signals, such as turning LEDs on/off.
3. **PWM (Pulse Width Modulation)(D3, D5, D6, D9, D10, D11):** Generate analog-like output levels for tasks like LED dimming or motor speed control.
4. **Communication Interfaces**: Enable serial communication with other devices using UART, SPI, or I2C protocols.
5. **Special Function Pins**: Also known as RX and TX pins, used for serial communication with other devices or computers.

32) Explain the layout of Arduino Uno with its components. (IMP)



1. **Reset Button–** This will restart any code that is loaded to the Arduino board
2. **AREF–** Stands for “Analog Reference” and is used to set an external reference voltage
3. **Ground Pin–** There are a few ground pins on the Arduino and they all work the same
4. **Digital Input/Output–** Pins 0-13 can be used for digital input or output.
5. **PWM–** The pins marked with the (~) symbol can simulate analog output.
6. **USB Connection–** Used for powering up your Arduino and uploading sketches
7. **TX/RX–** Transmit and receive data indication LEDs
8. **ATmega Microcontroller–** This is the brains and is where the programs are stored.
9. **Power LED Indicator–** This LED lights up anytime the board is plugged in a power source
10. **Voltage Regulator–** This controls the amount of voltage going into the Arduino board
11. **DC Power Barrel Jack–** This is used for powering your Arduino with a power supply
12. **3.3V Pin–** This pin supplies 3.3 volts of power to your projects
13. **5V Pin–** This pin supplies 5 volts of power to your projects
14. **Ground Pins–** There are a few ground pins on the Arduino and they all work the same
15. **Analog Pins–** These pins can read the signal from an analog sensor and convert it to digital

☞ **Draw Arduino block diagram**



33) Give the technical specifications of Arduino UNO. (IMP)

1. **Microcontroller**: ATmega328P
    - Architecture: 8-bit AVR
    - Operating Voltage: 5V
    - Input Voltage (Recommended): 7-12V
    - Input Voltage (Limits): 6-20V
2. **Digital I/O Pins**: 14 (of which 6 provide PWM output)
    - PWM Output Pins: D3, D5, D6, D9, D10, D11
3. **Analog Input Pins**: 6
    - Analog Input Range: 0-5V
    - Analog-to-Digital Converter (ADC) Resolution: 10-bit
4. **Flash Memory**: 32KB (0.5KB used for bootloader)
5. **SRAM**: 2KB
6. **EEPROM**: 1KB
7. **Clock Speed**: 16MHz
8. **Communication**:
    - UART: 1
    - SPI: 1
    - I2C: 1
9. **USB Interface**: ATmega16U2
10. **Power Supply**:
    - USB: 5V
    - External DC: 7-12V
11. **Dimensions**: 68.6mm x 53.4mm
12. **Weight**: 25g

34) What are the Input and Output Pins of Arduino UNO?

The Arduino Uno has a total of 14 digital input/output (I/O) pins, labeled from D0 to D13. Among these pins, some serve specific functions:

1. **Digital Output Pins** (with PWM support):
    - D3
    - D5

- D6
- D9
- D10
- D11

2. **Digital Input Pins**:
    - D0 to D13

35) Explain void setup() and void loop() functions. (IMP)

Question: 27

36) Describe Serial.begin(), Serial.print() and Serial.println() functions.

Question: 28

37) Describe the following functions with syntax and example (IMP)

38) digitalWrite():

This function is used to set the state of a digital pin to either HIGH (5 volts) or LOW (0 volts).

- Syntax: **digitalWrite(pin, value);**
    - **pin**: The number of the digital pin you want to control.
    - **value**: Either **HIGH** or **LOW**, representing the desired state of the pin

Example:

```
int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

39) digitalRead():

This function reads the state of a digital pin, returning either HIGH or LOW.

- Syntax: **value = digitalRead(pin);**
    - **pin**: The number of the digital pin you want to read from.
    - **value**: The state of the pin, which will be either **HIGH** or **LOW**.

Example

```
int buttonPin = 2; // Define the digital pin connected to the pushbutton
void setup() {
  pinMode(buttonPin, INPUT); // Set the pushbutton pin as an input
```

```
}
void loop() {
  int buttonState = digitalRead(buttonPin); // Read the state of the
   pushbutton
  if (buttonState == HIGH) {
    // Button is pressed
  } else {
    // Button is not pressed
  }
}
```

40) analogWrite(pin, value):

- **Syntax**: **analogWrite(pin, value);**
- **Description**: Writes an analog value (PWM) to a pin.
- **Parameters**:
    - **pin**: The digital pin number (D3, D5, D6, D9, D10, D11) which supports PWM.
    - **value**: The duty cycle to apply, ranging from 0 (always off) to 255 (always on).

Example

```
void setup() {
  pinMode(9, OUTPUT); // Set pin D9 as output
}
void loop() {
  analogWrite(9, 128); // Set LED brightness to half (50% duty cycle)
  delay(1000); // Wait for 1 second
}
```

41) analogRead(pin):

- **Syntax**: **analogRead(pin);**
- **Description**: Reads the analog value (0 to 1023) from the specified analog pin.
- **Parameter**:
    - **pin**: The analog pin number (A0 to A5) to read from.
- **Returns**:
    - Returns an integer value representing the analog voltage level, ranging from 0 (0 volts) to 1023 (5 volts).

Example

```
void setup() {
  Serial.begin(9600); // Initialize serial communication
}
void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue); // Print the value to the serial monitor
  delay(1000); // Wait for 1 second
}
```

42) delay(ms):

- **Syntax**: **delay(ms);**
- **Description**: Pauses the execution of the program for the specified number of milliseconds.
- **Parameter**:
    - **ms**: The duration of the delay in milliseconds (1 second = 1000 milliseconds).

Example

```
// Blink an LED every second using delay()
  void setup() {
    pinMode(13, OUTPUT); // Set pin D13 as output
  }
  void loop() {
    digitalWrite(13, HIGH); // Turn on the LED
    delay(1000); // Wait for 1 second
    digitalWrite(13, LOW); // Turn off the LED
    delay(1000); // Wait for 1 second
  }
```

43) millis():

- **Syntax**: **unsigned long millis();**
- **Description**: Returns the number of milliseconds since the Arduino board started running the current program.
- **Returns**:
    - An unsigned long integer representing the number of milliseconds since the program started.

Example

```
// Measure the elapsed time since the program started
unsigned long startTime;

void setup() {
  Serial.begin(9600); // Initialize serial communication
  startTime = millis(); // Record the start time
}
void loop() {
  unsigned long elapsedTime = millis() - startTime; // Calculate elapsed time
  Serial.println(elapsedTime); // Print elapsed time to serial monitor
  delay(1000); // Update every second
}
```

44) map(value, fromLow, fromHigh, toLow, toHigh):

- **Syntax**: **long map(value, fromLow, fromHigh, toLow, toHigh);**
- **Description**: Re-maps a value from one range to another. Useful for converting sensor readings or adjusting output values.
- **Parameters**:
    - **value**: The value to map.
    - **fromLow**: The lower bound of the input range.
    - **fromHigh**: The upper bound of the input range.
    - **toLow**: The lower bound of the output range.
    - **toHigh**: The upper bound of the output range.
- **Returns**:
    - A long integer representing the mapped value.

Example

```
// Map a sensor value to a range of LED brightness
void setup() {
  pinMode(A0, INPUT); // Set analog pin A0 as input
  pinMode(9, OUTPUT); // Set pin D9 as output for LED
}
void loop() {
  int sensorValue = analogRead(A0); // Read sensor value
  int brightness = map(sensorValue, 0, 1023, 0, 255); // Map value to LED brightness range
  analogWrite(9, brightness); // Set LED brightness
  delay(100); // Delay for stability
}
```

45) Explain Arduino function for the mathematical operations like square,square root, power, minimum and maximum.

1. **Square (sq())**:
    - **Syntax**: **sq(x)**
    - **Description**: Calculates the square of a number.
    - **Parameter**:
        - **x**: The number for which you want to find the square.
    - **Returns**:
        - The square of the input number.
    - **Example**: `int result = sq(5);`
2. **Square Root (sqrt())**:
    - **Syntax**: **sqrt(x)**
    - **Description**: Calculates the square root of a number.
    - **Parameter**:
        - **x**: The number for which you want to find the square root.
    - **Returns**:
        - The square root of the input number.
    - **Example**: `float result = sqrt(25);`

3. **Power (pow())**:
    - **Syntax**: **pow(base, exponent)**
    - **Description**: Calculates the value of a number raised to the power of another number.
    - **Parameters**:
        - **base**: The base number.
        - **exponent**: The exponent (the power to which the base is raised).
    - **Returns**:
        - The result of raising the base to the exponent.
    - **Example**: `float result = pow(2, 3);`
4. **Minimum (min())**:
    - **Syntax**: **min(a, b)**
    - **Description**: Returns the smaller of two numbers.
    - **Parameters**:
        - **a**: The first number.
        - **b**: The second number.
    - **Returns**:
        - The smaller of the two input numbers.
    - **Example**: `int result = min(10, 5);`
5. **Maximum (max())**:
    - **Syntax**: **max(a, b)**
    - **Description**: Returns the larger of two numbers.
    - **Parameters**:
        - **a**: The first number.
        - **b**: The second number.
    - **Returns**:
        - The larger of the two input numbers.
    - **Example**: `int result = max(10, 5);`

46) Describe Arduino functions to read and write digital values.

Question: 38,39

47) Describe Arduino functions to read and write Analog values.

Question: 40,41

48) Explain pulseIn() function.

- **Syntax**: **pulseIn(pin, value)**
- **Description**: Measures the duration of a pulse on the specified pin.

- Reads a pulse HIGH or LOW from specified pin and wait for the time to go the pulse from HIGH to LOW or LOW to HIGH.
- Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.
- Works onpulses from 10 microseconds to 3 minutes in length
- **Parameters**:
    - **pin**: The number of the digital pin to monitor the pulse.
    - **value**: The state of the pulse to measure, either HIGH or LOW.
- **Returns**:
    - The duration of the pulse in microseconds (µs).

Example

```
unsigned long duration = pulseIn(2, HIGH);
Serial.println(duration);
```

49) Explain map() function.

Question :44

50) Describe dealy() and micros().

Question :42

51) Write program to blink built-in LED

```
void setup() {
   pinMode(LED_BUILTIN, OUTPUT);
}

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

52) Write a code to control brightness of LED using potentiometer

```
const int potPin = A0; // Analog pin connected to the potentiometer
const int ledPin = 9;  // Digital pin connected to the LED

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

void loop() {
  // Read the analog value from the potentiometer (0 to 1023)
  int potValue = analogRead(potPin);
```

```
    // Map the potentiometer value (0 to 1023) to the PWM range (0 to 255)
    int brightness = map(potValue, 0, 1023, 0, 255);

    // Set the brightness of the LED using PWM
    analogWrite(ledPin, brightness);

    // Optional: Print the brightness value to the serial monitor
    Serial.println(brightness);

    // Optional: Add a small delay to prevent flickering
    delay(10);
  }
```

53) Write down a code to print your name on serial monitor with 9600 baud rate.

```
  void setup() {
    Serial.begin(9600); // Initialize serial communication at 9600 baud rate
  }

  void loop()
  {
    // Print your name to the serial monitor
    Serial.println("Deep Bhuva");
    delay(1000); // Print once every second
  }
```

54) Write a code to measure time duration of HIGH or LOW pulse on the specified pin using pulseIn() function.

```
const int inputPin = 2; // The digital pin connected to the input signal
unsigned long durationHigh, durationLow; // Variables to store pulse durations

void setup() {
  Serial.begin(9600); // Initialize serial communication
  pinMode(inputPin, INPUT); // Set the input pin as an input
}

void loop() {
  // Measure the duration of a HIGH pulse
  durationHigh = pulseIn(inputPin, HIGH);
  Serial.print("Duration of HIGH pulse: ");
  Serial.print(durationHigh);
  Serial.println(" microseconds");

  // Measure the duration of a LOW pulse
  durationLow = pulseIn(inputPin, LOW);
  Serial.print("Duration of LOW pulse: ");
  Serial.print(durationLow);
```

```
  Serial.println(" microseconds");

  delay(1000); // Wait for 1 second before measuring again
}
```

55) What is Raspberry Pi? Explain in detail.

- Raspberry Pi is a single-board computers, developed by Raspberry Pi Foundation in association with Broadcom and perhaps the most inspiring computer available today.
- Because of its low cost and open design, the model became far more popular than anticipated
- It is widely used to make gaming devices, fitness gadgets, weather stations, and much more
- In 2012, the company launched the Raspberry Pi and the current generations of regular Raspberry Pi boards are Zero, 1, 2, 3, and 4.
- Raspberry Pi needs an operating system to work.
- Raspberry Pi OS (previously called Raspbian) is an official supported operating system.
- Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi.
- Download and install Raspberry Pi Imager to a computer with an SD card reader.
- Put the SDcard into card reader and Run Raspberry Pi Imager

Architecture, Layout and Interface of Raspberry Pi

1. Processor
2. HDMIPort
3. USBPort
4. Ethernet Port
5. SDCardSlot
6. CameraConnector
7. Composite Output
8. AudioOutput Video
9. MicroUSBPower
10. GPIOPins
11. Status LEDs

56) What is GPIO pins of Raspberry Pi.

- GPIO (General Purpose Input/Output) pins on Raspberry Pi are physical pins located on the board that can be used for digital input or output operations. A40-pin GPIO header is found on all current Raspberry Pi boards.
- These pins allow the Raspberry Pi to interact with external electronic components such as sensors, LEDs, motors, and more.

- GIOP header contains outputs for 3.3V, 5V, Ground, and lots of General Purpose Input/Output (GPIO) pins!
- GPIO pins are one of the key features that make Raspberry Pi a versatile platform for learning about electronics and physical computing.

57) Give the technical specifications of Raspberry Pi.

1. **Raspberry Pi 4 Model B**:
   - Quad-core 64-bit ARM Cortex-A72 CPU
   - Options for 2GB, 4GB, or 8GB LPDDR4 RAM
   - MicroSD card slot for storage
   - Dual-band Wi-Fi, Gigabit Ethernet, Bluetooth 5.0
   - USB 3.0 and USB 2.0 ports, micro HDMI ports
   - 40-pin GPIO header
   - Supports up to 4K video output
2. **Raspberry Pi 3 Model B+**:
   - Quad-core 64-bit ARM Cortex-A53 CPU
   - 1GB LPDDR2 RAM
   - MicroSD card slot for storage
   - Wi-Fi, Bluetooth 4.2
   - USB 2.0 ports, HDMI port
   - 40-pin GPIO header
3. **Raspberry Pi Zero W**:
   - Single-core 32-bit ARM CPU
   - 512MB LPDDR2 RAM
   - MicroSD card slot for storage
   - Wi-Fi, Bluetooth 4.1
   - Mini HDMI port, micro USB ports
   - GPIO header

58) Write a Python Code for Blinking LED. (Raspberry Pi)

```
import time
import RPi.GPIO as GPIO
led_pin = 12

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

while True:
 GPIO.output(led_pin, GPIO.HIGH)

time.sleep(1)

GPIO.output(led_pin, GPIO.LOW)
time.sleep(1)
```

Unit 3 : Sensors, Microcontrollers and Interfacing

59) What is a Microcontroller?

- A microcontroller is a compact integrated circuit (IC) that includes a processor core, memory, and various input/output peripherals on a single chip.
- It is designed to execute specific tasks within embedded systems, where it serves as the brain of the system, controlling its operation and interfacing with external components.
  Key characteristics of microcontrollers include:

- Microcontrollers are optimized for specific applications, with 8-bit, 16-bit, or 32-bit architectures for processing tasks within embedded systems.
- They feature built-in memory components like RAM for data storage and ROM or Flash memory for program instructions and data storage.
- Microcontrollers come with various input/output peripherals, including digital and analog I/O pins, communication interfaces (UART, SPI, I2C), timers/counters, ADCs, and PWM controllers, facilitating interaction with external devices.
- Designed for low-power operation, microcontrollers are suitable for battery-powered and energy-efficient applications.
- Many microcontrollers support real-time processing, enabling them to respond to external events and execute tasks within specified time constraints.

60) Explain asynchronous Serial Communication with UART.

- Universal Asynchronous Receiver-Transmitter (UART)
- Universal Asynchronous Receiver-Transmitter (UART), a serial communication protocol that can be used to send data between an Arduino board and other devices
- It allows an asynchronous serial communication in which the data format and transmission speed are configurable

1. **Configuration**: In a point-to-point UART communication setup, there are two devices involved: a transmitting device and a receiving device. The transmitting device sends data, while the receiving device receives and processes it.
2. **Data Transmission**: The transmitter collects data from a source, formats it into serial bits, and sends it via a transmit (TX) pin. The receiver receives this data via a receive (RX) pin and processes it accordingly.
3. **Asynchronous Communication**: UART operates asynchronously, meaning it does not rely on a shared clock signal between the transmitter and receiver. Instead, it uses predefined baud rates to determine the timing of data bits.
4. **Baud Rate**: The baud rate, expressed in bits per second (bps), determines the speed of data transmission. Both the transmitting and receiving devices must agree on the same baud rate for successful communication.

5. **Flow Control**: In point-to-point UART communication, flow control methods may be employed to manage data transmission between devices, ensuring that the receiver can handle incoming data at its own pace. Hardware flow control uses additional wires for signaling, while software flow control utilizes special characters over data lines.

61) Write a code for UART communication between two Arduino UNO board via software serial port.

Transmitter Arduino Code:

```
#include <SoftwareSerial.h>

// Define software serial pins
#define TX_PIN 2
#define RX_PIN 3

// Create a SoftwareSerial object
SoftwareSerial mySerial(TX_PIN, RX_PIN);

void setup() {
  // Initialize serial communication at 9600 bps
  Serial.begin(9600);

  // Initialize software serial communication at 9600 bps
  mySerial.begin(9600);
}

void loop() {
  // Send data over software serial
  mySerial.println("Hello from Transmitter Arduino!");

  // Wait for a short delay
  delay(1000);
}
```

Receiver Arduino Code:

```
#include <SoftwareSerial.h>

// Define software serial pins
#define TX_PIN 2
#define RX_PIN 3

// Create a SoftwareSerial object
SoftwareSerial mySerial(TX_PIN, RX_PIN);

void setup() {
  // Initialize serial communication at 9600 bps
  Serial.begin(9600);

  // Initialize software serial communication at 9600 bps
  mySerial.begin(9600);
}

void loop() {
  // Check if data is available to read
  if (mySerial.available() > 0) {
    // Read data from software serial
    String receivedData = mySerial.readString();

    // Print received data to serial monitor
    Serial.println("Received data: " + receivedData);
  }
}
```

62) Explain Synchronous Serial communication with SPI.

- Synchronous Serial communication is a type of communication where the data is transmitted in a synchronized manner, meaning that the transmitting and receiving devices are both using the same clock signal.
- This ensures that the data is transmitted and received correctly, as the devices are both aware of when the data is being sent and received.
- SPI (Serial Peripheral Interface) is a synchronous serial communication protocol that is used for short-distance communication, primarily in embedded systems.
- SPI is a full-duplex protocol, meaning that data can be transmitted and received simultaneously.
- SPI is also a master-slave protocol, meaning that there is one master device that controls the communication and one or more slave devices that respond to the master device.

SPI uses four wires for communication:

- Serial Clock (SCK):
  The clock signal that synchronizes the data transmission.
- Serial Data Out (SDO):
  The data line that the master device uses to transmit data to the slave devices.
- Serial Data In (SDI):
  The data line that the slave devices use to transmit data to the master device.
- Chip Select (CS):
  The signal line that the master device uses to select which slave device it wants to communicate with.

63) Explain I2C protocol in-detail.

- The I2C (Inter-Integrated Circuit) protocol, also known as TWI (Two-Wire Interface), is a synchronous serial communication protocol used for communication between integrated circuits, especially within embedded systems and electronic devices.

1. **Master-Slave Architecture**:
   - I2C communication typically involves one master device and one or more slave devices.
   - The master device initiates communication and controls the timing of data transmission.
   - Slave devices respond to commands from the master and send data back as required.
2. **Two-Wire Interface**:
   - I2C uses only two wires for communication:
     - SDA (Serial Data): Bi-directional data line for transmitting and receiving data.
     - SCL (Serial Clock): Clock line controlled by the master device, used to synchronize data transmission.
3. **Addressing**:
   - Each slave device on the I2C bus has a unique 7-bit or 10-bit address.
   - The master device initiates communication by sending the address of the slave it wants to communicate with.
   - Multiple slave devices can share the same bus, with each having a different address.
4. **Synchronous Operation**:
   - I2C operates synchronously, meaning that data is transmitted based on a shared clock signal (SCL) between the master and slave devices.
   - Data is transmitted in a series of clock cycles, with each bit of data being transmitted or received on the rising or falling edge of the clock signal.
5. **Speed Modes**:
   - I2C supports different speed modes, including standard mode (up to 100 kbps), fast mode (up to 400 kbps), and high-speed mode (up to 3.4 Mbps).

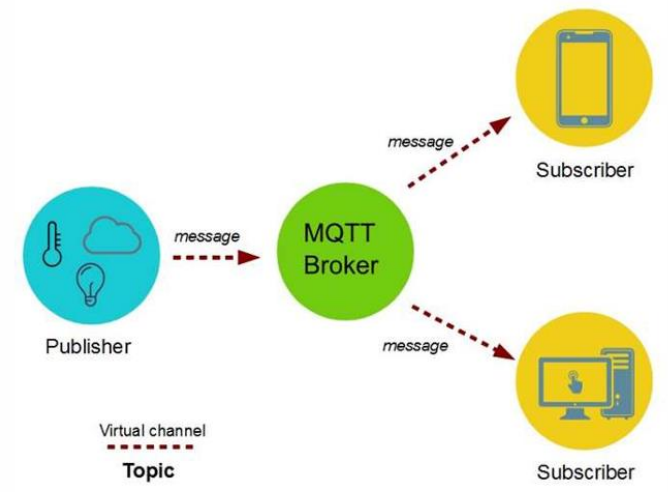
64) Explain MQTT protocol with respect to IoT infrastructure.

- It's Reliable, Lightweight, Cost-effective protocol that transports messages between devices.
- Suited for the transport of telemetry data (sensor and actuator data) ♣
- MQTT is a machine-to-machine connectivity protocol that runs over TCP/IP.
- MQTT is based on a publish- subscribe structure.
- Publish : A Publisher sends messages according to Topics, to specified Brokers.
- Broker : A Broker acts as a switchboard, accepting messages from publishers on specified topics, and sending them to subscribers to those Topics.
- Subscribe : A Subscriber receives messages from connected Brokers and specified Topics.
- MQTT is an Event based IoT middleware (one to many).
- It can be supported by some of the smallest measuring and monitoring devices (ex. Arduino).

MQTT Architecture:-

Architecture of MQTT:

1. Publisher
2. Broker
3. Subscriber



MQTT Communication:

- If no matches, the message is discarded
- If one /more matches, the message is delivered to each matching subscriber/consumer
- Asynchronous communication model with messages (events)
- Publisher and Subscriber need not be online at the same time

MQTT Topic:

- Is hierarchical with each "sub topic" separated by a /

A house publishes information about itself on:

- <country>/<state>/<town>/<postcode>/<house>/energyConsumption
- <country>/<state>/<town>/<postcode>/<house>/solarEnerg
- <country>/<state>/<town>/<postcode>/<house>/alarmState

And subscribes for control commands:

- <country>/<state>/<town>/<postcode>/<house>/thermostat/setTemp

65) Define Sensor and list the different types of sensors.

- **Sensors**: Sensors are the ways of getting information into your device, finding out things about your surroundings.
- A sensor is a device that detects changes and events in a physical environment.
- It may convert physical parameters like humidity, pressure, temperature, heat, motion, etc., into electrical signals.
- Electrical signal can be converted into a human-readable display and sent across a network for additional processing.
- **Types** : Active sensors and Passive sensors.
- Active sensors necessitate a power supply, whereas passive sensors don't require a power supply

  list of different types of sensors:

  1. Temperature Sensors
  2. Pressure Sensors
  3. Motion Sensors
  4. Light Sensors
  5. Humidity Sensors
  6. Gas Sensors
  7. Force Sensors
  8. Color Sensors
  9. Sound Sensors
  10. pH Sensors
  11. Vibration Sensors
  12. Moisture Sensors
  13. Flame Sensors
  14. Optical Sensors

66) Explain types of Sensor with example.

1. **Temperature Sensors**: Measure the temperature of their surroundings, such as the LM35 sensor used in weather stations.
2. **Pressure Sensors**: Detect changes in pressure, like the BMP280 sensor used in altimeters.
3. **Motion Sensors**: Detect motion or movement, such as the PIR (Passive Infrared) sensor used in security systems.

4. **Light Sensors**: Measure light intensity, like the LDR (Light Dependent Resistor) used in streetlights.
5. **Humidity Sensors**: Measure the humidity level in the air, such as the DHT22 sensor used in climate control systems.
6. **Gas Sensors**: Detect the presence and concentration of gases, like the MQ-2 sensor used in gas leakage detectors.
7. **Force Sensors**: Measure force or pressure applied to them, such as the FSR (Force-Sensitive Resistor) used in touch-sensitive devices.
8. **Color Sensors**: Identify and detect colors, like the TCS3200 sensor used in color recognition systems.
9. **Sound Sensors**: Detect sound levels and variations, such as the sound detection module used in noise monitoring systems.

67) Mention the importance of sensors in IoT applications with appropriate examples.

- The Internet of Things (IoT) is revolutionizing the way we live and work by connecting everyday objects to the internet.
- This connectivity allows for the collection and analysis of data in real-time, enabling us to make informed decisions and automate processes.

Sensors in IoT:

- In IoT, sensors are used to collect data from various sources and send it to cloud-based platforms for analysis.
- The data collected by sensors are used to monitor and control various systems, including environmental conditions, traffic patterns, and equipment performance.
- For example, sensors in a smart home monitor temperature, humidity, and occupancy, and adjust the heating and cooling systems accordingly.
- Sensors are also used to track the location of assets, such as vehicles or inventory, in real-time.
- This allows businesses to optimize their supply chain and logistics operations by reducing inventory costs, improving delivery times, and increasing efficiency.
- Another use of sensors in IoT is in healthcare. Wearable sensors monitor vital signs such as heart rate, blood pressure, and oxygen levels, allowing doctors to monitor patients remotely and make timely interventions when necessary.

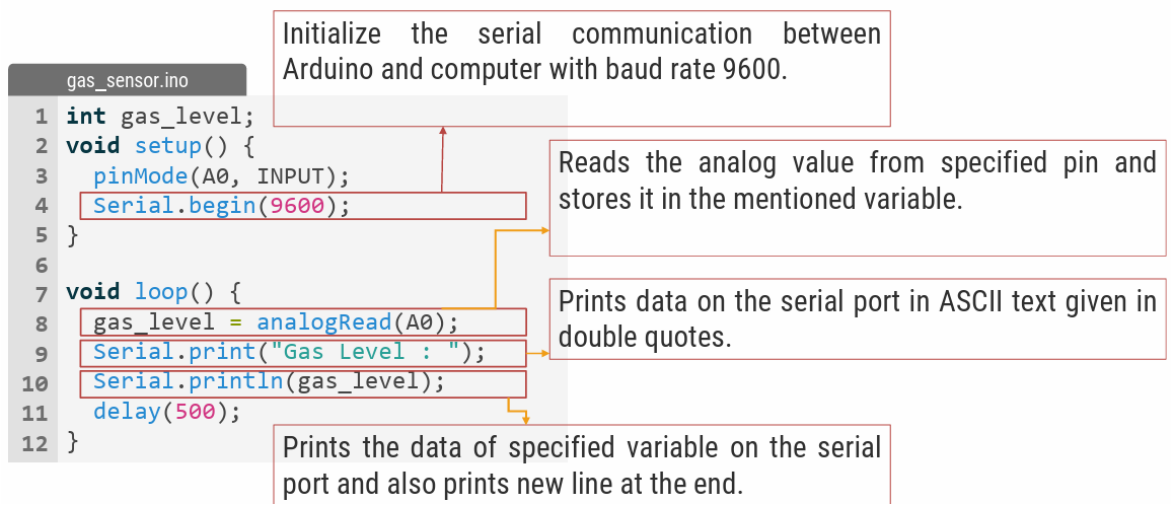
68) Take any reference and explain how a pH sensor can be used in an IoT application.

Water Quality Monitoring:

- In agriculture and environmental monitoring, pH sensors can be deployed in bodies of water such as rivers, lakes, and reservoirs to monitor water quality.

- IoT-enabled pH sensors can continuously measure pH levels and transmit the data wirelessly to a central server or cloud platform for analysis.
- This information can help detect pollution, assess the health of aquatic ecosystems, and ensure water quality compliance with regulatory standards.

69) Write an Arduino code to detect presence of gas using MQ-02/05 Gas Sensor with Arduino Uno.



70) Explain the working principle of HC-SR04 Ultrasonic Sound Sensor.

Here's how the HC-SR04 sensor works:

1. The transmitter pin initiates the transmission of sound waves.
2. The HC-SR04 will automatically send eight 40 kHz sound waves.
3. The waves travel through the air at a speed of 340 m/s (0.034 cm/s).
4. The waves bounce back to the module if there is an object or obstacle on their path.
5. The time it takes for the waves to return is used to calculate the distance.
6. The total distance is divided by 2 because the signal travels from HC-SR04 to object and returns to the module HC-SR-04.

71) Write a code for Arduino to measure light intensity in the environment using LDR Sensor.

```ino
// Define the LDR pin
const int ldrPin = A0;

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

void loop() {
  // Read the analog value from the LDR sensor
  int sensorValue = analogRead(ldrPin);

  // Convert the analog value to voltage (0-5V)
  float voltage = sensorValue * (5.0 / 1023.0);

  // Convert voltage to light intensity percentage
  // Assuming LDR is connected to a voltage divider with a fixed resistor
  // and voltage varies linearly with light intensity
  float lightIntensity = (voltage / 5.0) * 100.0;

  // Print the light intensity value
  Serial.print("Light Intensity: ");
  Serial.print(lightIntensity);
  Serial.println("%");

  // Wait for a short delay before taking another reading
  delay(1000);
}
```

72) Write an Arduino code to detect color of object. (Color Sensor)

Unit 4: Controlling Hardware

73) What is actuators?

- An IoT device is made up of a Physical object (“thing”) + Controller (“brain”) + Sensors + Actuators + Networks (Internet).
- An actuator is a machine component or system that moves or controls the mechanism or the system.
- An actuator is a machine component or system that moves or controls the mechanism or the system.
- Actuators are the output devices like LEDs, motors, lights, and so on, which let your device do something to the outside world.
- A servo motor is an example of an actuator.

74) Define Actuators and explain Relay module with Pin-Out.

- An actuator is a machine component or system that moves or controls the mechanism or the system.
- They are responsible for executing various tasks based on input signals received from sensors or controllers.
- Actuators play a crucial role in controlling and manipulating the physical environment in response to changes detected by sensors.

Relay:

- A relay is an electrically activated switch.
- 5V Relay Module 10A current capacity.
- This small Relay Board works from a 5V signal. It uses a transistor to switch the relay on so can be connected directly to a microcontroller pin.
- When the relay is not activated, the common pin is in contact with the NC pin and when it is activated, the common pin will break away from contact with the NC pin and subsequently makes contact with the NO pin

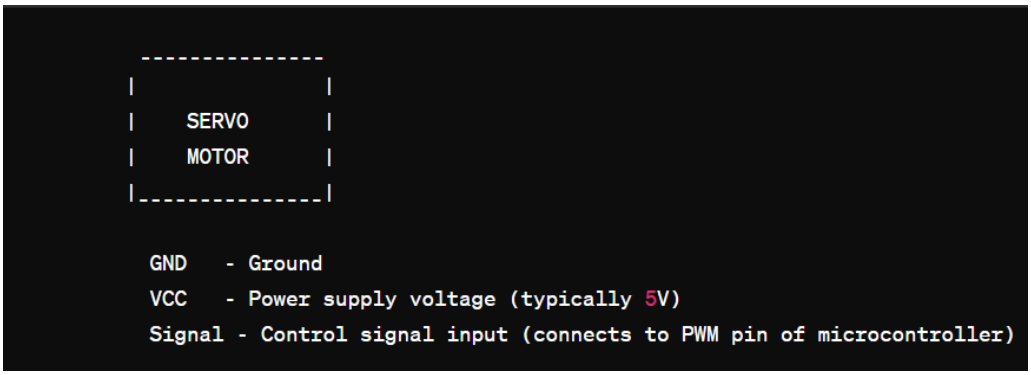
- VCC - Power supply voltage (typically 5V)
- GND - Ground

- IN - Control input (connects to Arduino or microcontroller pin)
- COM - Common terminal (connected to one side of the load)
- NO - Normally open terminal (connected to the other side of the load, when relay is inactive)
- NC - Normally closed terminal (connected to the other side of the load, when relay is active)

75) Describe Servo motor with Pin-Out.

- Servo is a general term for a closed-loop control system.
- A closed-loop system uses the feedback signal to adjust the speed and direction of the motor to achieve the desired result.

  Servo Motor:
- A servo motor is an example of an actuator.
- They are linear or rotary actuators capable of moving to a specified angular or linear position.
- Servo motors can be used in IoT applications to rotate to specific angles such as 90 degrees or 180 degrees.
- A servo motor is a closed-loop servomechanism that utilizes position feedback to control its motion and final position.
- The input to its control is a signal, either analog or digital, representing the commanded position for the output shaft.



76) Demonstrate all the methods of Servo class for Arduino.

attach()	write()	read()	attached()	detach()
• Attach the Servo variable to a pin.	• Writes a value to the servo, controlling the shaft accordingly. • On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation.	• Read the current angle of the servo (the value passed to the last call to write()).	• Check whether the Servo variable is attached to a pin.	• Detach the Servo variable from its pin.
• Syntax ➢ servo.attach(pin)	• Syntax ➢ servo.write(angle)	• Syntax ➢ servo.read()	• Syntax ➢ servo.attached()	• Syntax ➢ servo.detach()

Darshan UNIVERSITY

Servo Methods

attach()	write()	read()	attached()	detach()
• Syntax ➢ servo.attach(pin)	• Syntax ➢ servo.write(angle)	• Syntax ➢ servo.read()	• Syntax ➢ servo.attached()	• Syntax ➢ servo.detach()

#include <Servo.h>
Servo myservo;
void setup()
{
myservo.attach(9);
//attach servo to pin 9
}
void loop()
{
 	
}
	#include <Servo.h>
	Servo myservo;
	void setup() {
	myservo.attach (9);
	myservo.write (90);
	// set servo to mid-point
	}
	void loop() {
	myservo.write (0);
	// set servo to start-point
	}
		#include <Servo.h>
		Servo myservo;
		void setup() {
		myservo.attach (9);
		myservo.write (90);
		}
		void loop() {
		int agl;
		agl = myservo.read();
		// return angle value
		//Here agl = 90;
		}
			#include <Servo.h>
			Servo myservo;
			void setup() {
			myservo.attach (9);
			myservo.write (90);
			// set servo to mid-point
			}
			void loop() {
			bool joined;
			joined = myservo.attached();
			// return true if attached
			//return false if not attached
			}
				#include <Servo.h>
				Servo myservo;
				void setup() {
				myservo.attach (9);
				}
				void loop() {
				
				//code
				
				myservo.detach (9);
				//detach Servo from
				//pin 9
				
				}

77) Write a code to controll the motion of DC motor in forward and Revese direction.

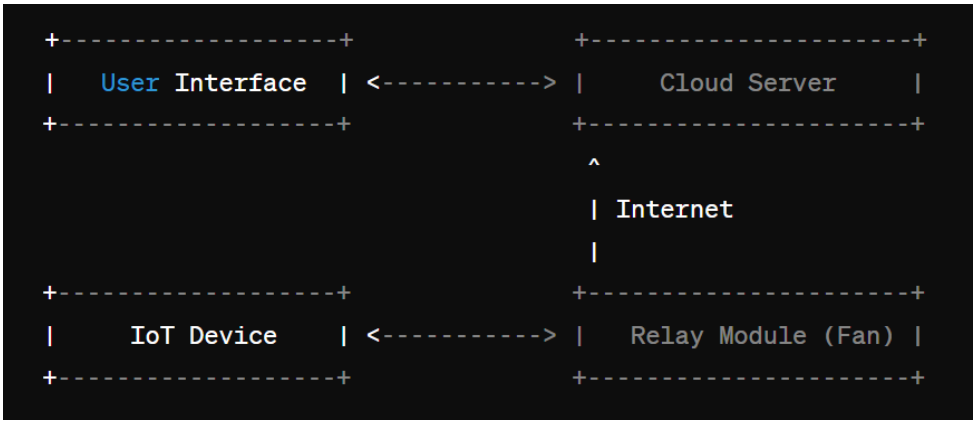
```
// Define the pins connected to the motor driver
const int enablePin = 9;    // PWM pin for speed control
const int in1Pin = 8;       // Input 1 pin for direction control
const int in2Pin = 7;       // Input 2 pin for direction control

void setup() {
  // Set the motor control pins as outputs
  pinMode(enablePin, OUTPUT);
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
}

void loop() {
  // Forward motion
  digitalWrite(in1Pin, HIGH);
  digitalWrite(in2Pin, LOW);
  analogWrite(enablePin, 255); // Full speed

  delay(2000); // Wait for 2 seconds

  // Stop
  analogWrite(enablePin, 0);

  delay(1000); // Wait for 1 second

  // Reverse motion
  digitalWrite(in1Pin, LOW);
  digitalWrite(in2Pin, HIGH);
  analogWrite(enablePin, 255); // Full speed

  delay(2000); // Wait for 2 seconds

  // Stop
  analogWrite(enablePin, 0);

  delay(1000); // Wait for 1 second
}
```

78) Assume you have a fan at your home. You wish to control the On/Off switch via the Internet. Design a system for its control. Draw the system architecture.

- To control the on/off switch of a fan via the Internet, you can design a system using IoT components such as a microcontroller (e.g., Arduino or ESP8266), a relay module to act as a switch, and a Wi-Fi module for Internet connectivity.

1. **User Interface (Mobile App or Web Interface):**
    - Allows the user to remotely control the fan switch.
    - Sends commands (on/off) to the cloud server.
2. **Cloud Server:**
    - Receives commands from the user interface.
    - Processes the commands and forwards them to the IoT device.
3. **IoT Device (NodeMCU/ESP8266 or similar):**
    - Receives commands from the cloud server.
    - Controls the relay module based on the received commands.
4. **Relay Module:**
    - Acts as a switch to turn the fan on/off based on signals from the IoT device.
5. **Fan:**
    - Connected to the relay module, turns on/off based on the relay's state.

System architecture.



79) Demonstrate interfacing of Bluetooth Module with Arduino.

- Interfacing a Bluetooth module with Arduino allows you to wirelessly communicate with other devices such as smartphones, tablets, or computers. Here's a basic demonstration of how to interface a Bluetooth module (HC-05) with Arduino:

Components Needed:

- Arduino board (e.g., Arduino Uno)
- HC-05 Bluetooth module
- Jumper wires

Connections:

- Connect the TX pin of the Bluetooth module to the RX pin (pin 0) of the Arduino.
- Connect the RX pin of the Bluetooth module to the TX pin (pin 1) of the Arduino.

- Connect the VCC pin of the Bluetooth module to the 5V pin of the Arduino.
- Connect the GND pin of the Bluetooth module to the GND pin of the Arduino.

```
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(0, 1); // RX, TX

void setup() {
  Serial.begin(9600);  // Initialize serial communication with the computer
  BTSerial.begin(9600);  // Initialize serial communication with the Bluetooth modul
}

void loop() {
  // Read data from Bluetooth module and send it to serial monitor
  if (BTSerial.available()) {
    char c = BTSerial.read();
    Serial.write(c);
  }

  // Read data from serial monitor and send it to Bluetooth module
  if (Serial.available()) {
    char c = Serial.read();
    BTSerial.write(c);
  }
}
```

80) Explain Zigbee and interfacing circuit of Xbee module with Arduino

- Zigbee is a popular wireless communication protocol used to transfer a small amount of data with very low power.
- It is widely used in applications where data has to be shared among many nodes within personal space and with the advent of the Internet of Things (IoT).
- To establish Zigbee communication, we need a receiver (endpoint). For this, an XBee module connected with Arduino/NodeMCU will do, and this will communicate wirelessly with another XBee module (coordinator) that sends data, will be connected to another Arduino board.

Components Needed:

1. Arduino board (e.g., Arduino Uno)
2. XBee module (e.g., XBee Series 2)
3. XBee Explorer USB or XBee Shield (for programming and interfacing)
4. Jumper wires

Connections:

1. If using XBee Explorer USB:

- Connect the XBee module to the XBee Explorer USB using the appropriate pins.
- Plug the XBee Explorer USB into a USB port on your computer.

2. If using XBee Shield:
   - Plug the XBee module into the XBee Shield.
   - Stack the XBee Shield onto the Arduino board.

```
#include <SoftwareSerial.h>

SoftwareSerial XBeeSerial(2, 3); // RX, TX (pin 2 = RX, pin 3 = TX)

void setup() {
  Serial.begin(9600);  // Initialize serial communication with the computer
  XBeeSerial.begin(9600);  // Initialize serial communication with the XBee module
}

void loop() {
  // Read data from XBee module and send it to serial monitor
  if (XBeeSerial.available()) {
    char c = XBeeSerial.read();
    Serial.write(c);
  }

  // Read data from serial monitor and send it to XBee module
  if (Serial.available()) {
    char c = Serial.read();
    XBeeSerial.write(c);
  }
}
```

Unit 5: Domain Specific Applications of IoT.

81) Explain Cloud Computing.

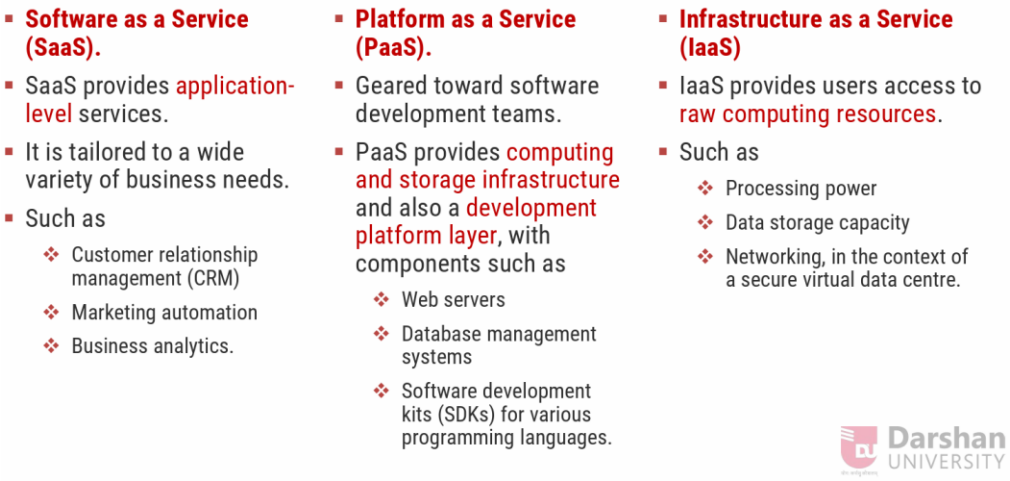
- **Cloud computing :**
  "Cloud computing refers to the delivery of computing services over the internet, allowing users to access and utilize computing resources (such as servers, storage, databases, networking, software, and analytics) on-demand, without the need for direct management of physical infrastructure."
- Cloud computing is the delivery of computing services over the internet, by providing flexible, affordable, effective and efficient resources for development

Benefits of Cloud Computing

- Cost Savings: Cloud Computing allows to decrease operating costs, that includes servers, storage, databases, networking, software, analytics, and intelligence.
- Scalability: Easily scales resources up or down to meet changing demands without incurring downtime or disruption.
- Flexibility: Offers a wide range of services and deployment options to suit diverse business requirements.
- Reliability: Cloud providers offer high availability, redundancy, and disaster recovery capabilities to ensure data and application availability.

82) What are the service models of the Cloud?

- There are three main service models of cloud computing.
  - Infrastructure as a Service (IaaS)
  - Platform as a Service (PaaS)
  - Software as a Service (SaaS).
- Each model represents a different part of the cloud computing stack.



83) Explain Fog Computing.

- Fog and cloud computing are inter-twined.
- In nature, Fog is closer to Earth than clouds
- In the tech world, it's the same; Fog is closer to end-users, bringing cloud capabilities to the ground.
- Fog is an extension of cloud computing that consists of multiple edge nodes directly connected to physical devices.
- The considerable processing power of edge nodes allows them to compute large amounts of data without sending them to distant servers.

Key characteristics of fog computing include:

- Distributed: It uses a network of devices like routers, IoT gadgets, and sensors to share the workload, making processing more efficient.
- Real-Time: It can quickly analyze data as it's generated, making it great for tasks that need instant responses, like monitoring equipment or controlling traffic lights.
- Security: By keeping data closer to where it's generated, fog computing can be more secure and private since it doesn't have to travel far over the internet.

84) What are the benefits of Fog Computing?

1. Lower Latency: Processing data closer to where it's generated reduces delays and improves response times.
2. Improved Reliability: Fog computing can continue operating even if the connection to the cloud is lost, ensuring uninterrupted service.
3. Bandwidth Optimization: Local processing reduces the need to transmit large amounts of data to centralized servers, optimizing bandwidth usage.
4. Enhanced Security and Privacy: Data processed locally reduces the risk of data breaches during transmission over the internet.
5. Scalability and Flexibility: Fog computing supports efficient resource utilization and can accommodate varying workloads.
6. Real-Time Processing: Enables faster decision-making and response to events critical for applications requiring immediate action.
7. Cost-Effectiveness: Reduces infrastructure costs and operational expenses by minimizing data transmission and central processing resources.

85) How is fog computing different from cloud computing?

- Location
  Cloud computing is centralized, with data stored, processed, and accessed from a remote data center. Fog computing is decentralized, with data processed closer to edge devices.
- Data access
  Fog computing stores some data in a local data center, allowing for offline access. Cloud computing does not allow for offline access.
- Data reception

Fog computing receives data from IoT devices in real time. Cloud computing receives and summarizes data in large, centralized server nodes.

- Latency
  Fog computing has low latency because data doesn't have to travel far from the device. Cloud computing has high latency because data has to travel to a centralized server.
- Energy consumption and operational costs
  Fog computing has lower energy consumption and operational costs than cloud computing.

86) Explain IAAS and SAAS services in detail.

Software-as-a-Service (SaaS)

- Complete software application as a service is provided to the user that is run and managed by the service provider.
- It can also be called application as a service as a pay monthly, yearly etc. subscription.
- In SaaS, user don't need to worry about software up-gradation and management.
- A common example of a SaaS application is web-based email where you can send and receive email without having to manage the server that the email program is written.

Example of SaaS Providers:

- Salesforce (Customer Relationship Management)
- Google Workspace (Productivity Suite)
- Microsoft Office 365 (Productivity Suite)

Infrastructure-as-a-Service (IaaS)

- IaaS provides access to computing hardware, networking features, and data storage space.
- Where the consumer is able to deploy and run software, which can include operating systems and applications.
- The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications.
- This service provides the highest level access

Example of IaaS Providers:

- Amazon Web Services (AWS) EC2
- Microsoft Azure Virtual Machines
- Google Compute Engine

87) Define Virtual Pins with respect to the Blynk Cloud.

- **Virtual Pin** is a concept invented by Blynk Inc. to provide exchange of any data between hardware and Blynk mobile app.

- Virtual pins are different than Digital and Analog Input/Output (I/O) pins. They are **physical pins** on your microcontroller board where you connect sensors and actuators.
  For example, if you need to turn On/Off LED connected to Digital pin, you don't have to write any code:
- Just use BlynkBlink code for your hardware.
- In the Blynk app - add **Button Widget** and set it to pin **D8**
- That's it! No additional code is required. Simply press Play in the app.

88) Explain Blynk.virtualWrite() function to sent or write data on the virtual pin or library function to write data on virtual pin

- In Blynk, the **Blynk.virtualWrite()** function is used to send data from the hardware device (such as an Arduino, Raspberry Pi, or ESP8266/ESP32) to a virtual pin in the Blynk cloud platform.
- This function allows users to transmit various types of data, including numerical values, strings, and custom data types, to the Blynk app or web dashboard for visualization, logging, or control purposes.

Syntax:
Blynk.virtualWrite(virtualPin, value);

- virtualPin: The numerical identifier of the virtual pin on the Blynk platform.
- value: The data value to be sent to the virtual pin. This can be a numerical value, string, or custom data type supported by the Blynk platform.

```
#define BLYNK_PRINT Serial    // Defines the output channel for Blynk debug messag
#include <ESP8266WiFi.h>      // Include the ESP8266 WiFi library
#include <BlynkSimpleEsp8266.h>  // Include the Blynk library for ESP8266

// Blynk authentication token
char auth[] = "YourAuthToken";

// WiFi credentials
char ssid[] = "YourWiFiSSID";
char pass[] = "YourWiFiPassword";

void setup() {
  Serial.begin(9600);   // Initialize serial communication
  Blynk.begin(auth, ssid, pass);  // Connect to Blynk cloud server
}

void loop() {
  int sensorValue = analogRead(A0); // Read analog sensor value
  Blynk.virtualWrite(V1, sensorValue); // Send sensor value to virtual pin V1
  delay(1000);  // Delay for 1 second
}
```

89) Explain BLYNK_WRITE() function to receive or read data from the virtual pin or Library function to read data from the virtual pin.

- The **Blynk.virtualWrite()** function is a crucial part of the Blynk IoT platform, allowing users to send data to virtual pins within their hardware projects.
- This function is utilized in conjunction with the Blynk library in Arduino sketches or other compatible microcontroller platforms.

Syntax:
Blynk.virtualWrite(pin, value);

- pin: The virtual pin number to which the data will be written.
- value: The data or value to be written to the virtual pin. This can be of various data types such as integers, floats, strings, etc., depending on the requirement.

```
int sensorValue = analogRead(A0); // Read sensor value
Blynk.virtualWrite(V1, sensorValue); // Send sensor value to virtual pin V1
```

90) Explain common architecture of IoT applications.

No Idea

91) Explain Healthcare Application with IoT.

- The healthcare sector consists of medical and related goods and services
- Healthcare sector provides medical services for maintaining improving health via or
  - Prevention
  - Diagnosis
  - Treatment
  - Recovery or cure of disease, illness, injury, and other physical and mental harms in people.

Requirements for IoT Setup:
Sensors:
- Heartbeat Sensor
- Oxygen Level Sensor
- Glucose Level Sensor

Controller:
- ESP8266 NodeMCU or Arduino Uno with Wi-Fi or Network Shield

For Monitoring and Analysis:
- PC
- Mobile
- Tablet

For Fog Computing:

- Raspberry Pi or PC
- LAN and WAN Connectivity
- Router and Switches
- Wireless Access Point

How it works

- All the sensor nodes should be placed on patient body, either direct or using wearables.
- The sensor nodes should have Wi-Fi connectivity with a controller node of the patient.
- The sensor nodes sends the sensed data to the controller node.
- The controller node collects
  - The data from all the sensors,
  - Apply some business logic on it as per the requirements and
  - Send it to Fog computer.
- All the controller nodes of all the patients should have connected with the Fog computer network.

Fog computer

- Receives all the data from all the controller nodes
- Execute some business logic and
- Do analysis on the collected data, and
- Send the filtered limited data to the cloud computer as well as
- Send the alert messages to the care taker team (nursing staff and doctors).

- The Fog computer should have Internet connectivity for the cloud communication.
- All the local care taker team members can be accessed health status of all the patients from the Fog computer.

92) Explain IoT Application in Retail.

Requirements for IoT Setup:

- Sensors: Heartbeat Sensor, Oxygen Level Sensor, Glucose Level Sensor, Touch Sensors, Gas Sensors, Fire Sensors, IR Sensors.
- Actuators and Tags: Alarm, Water Sprinkler, Beacons, RFID Tags, RFID Reader.
- Controller: ESP32 MCU or Arduino Uno with Wi-Fi or Network Shield.
- Monitoring and Analysis: PC, Mobile, Tablet.
- LAN and WAN Connectivity: Router, Switches, Wireless Access Point.
- Cloud Services

Architecture:

- Sensor Nodes: Placed at various locations in the store, washroom, and parking area.

- Controller Nodes: Collect data from sensor nodes, apply business logic, and send to the local computer.
- RFID Readers: Detect item presence in display shelves for stock management.
- Local Server: Receives data from controller nodes, executes logic, and sends essential data to the cloud.
- Cloud: Stores and analyzes data, accessible by authorized users.
- Beacons: Connect with customers' smartphones, send push notifications for offers and discounts.

How it Works:

- Sensor nodes collect data and send it to controller nodes via Wi-Fi.
- RFID readers monitor item presence and stock levels.
- Touch sensors gather feedback on ambiance and cleanliness.
- Light and temperature sensors optimize energy usage and customer comfort.
- Camera and IR sensors monitor people density and movements.
- Alarm and water sprinkler ensure fire safety.
- Local server processes data and sends essential information to the cloud.
- Cloud stores and analyzes data, accessible by store managers, staff, and data analytics teams.
- Beacons connect with customers' smartphones for in-store navigation and real-time offers.

93) Discuss Driver Assistance Application with IoT.

Overview:

- Vehicle manufacturing companies aim to enhance vehicle comfort and safety for passengers.
- They seek to develop a system that can monitor and alert users suffering from highway hypnosis or drowsiness.
- IoT-based systems are needed to prevent accidents caused by drowsy drivers.

Importance:

- Drowsiness is a major factor in road accidents, leading to physical inactivity and reduced sensitivity to surroundings.
- Monitoring drivers' consciousness, acceleration patterns, and steering angles is crucial.
- Image processing and machine learning can be employed to detect driver deviation and drowsiness.

Requirements for IoT Setup:

- Sensors: Ultrasonic Sensor.
- Cameras.
- On-Board Diagnostic (OBD) Access.
- Controller: Raspberry Pi.

- Monitoring and Analysis: PC, Mobile, Tablet.
- WAN Connectivity.
- Cloud Services.

Architecture:

- Drowsiness Detection System: Utilizes EAR value, facial area, lane detection, speed variation.
- Deviation Detection System: Measures distance from surrounding vehicles, monitors acceleration and steering angles.
- Components include camera units, OBD unit, ultrasonic sensor nodes, and the vehicle computer.

How it Works:

- Camera units capture driver's facial and eye movements, send data to the vehicle computer for analysis.
- EAR value (Eye Aspect Ratio) is calculated to detect drowsiness when the driver closes their eyes.
- Roadside cameras capture front view images for lane, object, and signboard detection.
- OBD records acceleration and steering angle patterns during drowsiness, compares with threshold values.
- Analyzed data is sent to the cloud for further processing and alerting the driver.

94) Explain Collision Impact Detection Using IoT.

Overview:

- Modern vehicles feature collision detection mechanisms, such as airbag deployment, to save lives during accidents.
- However, in severe collisions, airbags may not be sufficient, necessitating emergency assistance.
- An IoT-based system is needed to measure collision severity and send alerts.

Importance:

- Severity data is crucial in managing multiple accidents on highways to save lives.
- Timely emergency assistance can be dispatched based on severity data.
- An IoT system is required to sense necessary parameters of collision impact.

Requirements for IoT Setup:

- Sensors: FSR Sensor, Fire Sensor, GPS Sensor.
- Cameras.
- OBD Access.
- Controller.
- Monitoring and Analysis: PC, Mobile, Tablet.

- WAN Connectivity.
- Cloud Services.

Architecture:

- Components include FSR Sensor nodes, fire sensor nodes, GPS nodes, cameras, OBD unit, vehicle computer, and cloud services.
- Sensors measure collision impact, fire detection, and vehicle speed.
- Cameras capture passenger count and severity impact.

How it Works:

- FSR sensors measure pressure applied to the vehicle during a collision to estimate severity.
- Fire sensors and GPS provide additional data on fire detection and location.
- Cameras capture passenger information and send it to the vehicle computer.
- OBD records vehicle speed, while the vehicle computer collects data from all sensors.
- Analyzed data is sent to the cloud, where alerts are generated for emergency service providers and vehicle owners' family members.
- The system automatically activates when a collision is detected.

95) Explain IoT Based Application for Water Quality Monitoring.

Overview:

- Water quality is crucial for life, but pollution poses a significant challenge worldwide.
- Monitoring water quality helps prevent and control water pollution, avoiding serious health issues.
- Traditional methods of water quality monitoring, such as manual sampling, are expensive and inefficient.
- IoT offers the potential to develop a more efficient water quality monitoring (WQM) system.

Importance:

- Current water quality monitoring methods are costly and inefficient.
- Manual methods involve sampling, laboratory testing, and delayed reporting.
- IoT-based systems can provide real-time monitoring, leading to quicker responses to pollution incidents.

Requirements for IoT Setup:

- Water Sensors Kit: pH Sensor, GPS Sensor, Turbidity Sensor, TDS and Dissolved Oxygen Sensor, Temperature Sensor, Conductivity Sensor.
- Controller: ESP32 MCU or Arduino Uno with Wi-Fi or Network Shield.
- Monitoring and Analysis: PC, Mobile, Tablet.
- WAN Connectivity: GPRS or GSM Module.

- Cloud Services.
- Unmanned Surface Vehicle (USV).

Architecture:

- Components include water quality sensors kit, controller node, USV, GPRS/GSM module, and cloud services.
- Sensors measure various parameters like pH, temperature, conductivity, turbidity, TDS, dissolved oxygen, and location.
- Data is sent to the cloud server for storage, analysis, and access by authorized users.

How it Works:

- Water quality sensors are mounted on an autonomous or unmanned surface vehicle (USV).
- The USV, controlled by an embedded controller node, operates on water bodies.
- The USV has GSM or GPRS internet connectivity to send sensed data to the cloud server.
- The cloud stores data, performs analysis, and provides access to authorized users.
- Users include pollution control board members, data analytics teams, and USV operators.

❖ Solution Prepared By Deep Bhuva
