Q1)Define Sensor and explain types of sensors with examples.

➔Sensor Definition

A sensor is a device that detects a physical quantity and converts it into a signal that can be measured or recorded. This signal can be electrical, mechanical, or another form of energy, depending on the type of sensor.

Types of Sensors

Sensors can be classified based on the physical quantity they measure. Here are some common types:

Temperature Sensors:

 * Thermocouples: Measure temperature by generating a voltage based on the temperature difference between two dissimilar metals.

 * Resistance Temperature Detectors (RTDs): Measure temperature by measuring changes in electrical resistance.

 * Thermistors: Similar to RTDs, but use a semiconductor material instead of a metal.

 * Infrared (IR) Sensors: Detect temperature by measuring the amount of infrared radiation emitted by an object.

Light Sensors:

 * Photodiodes: Convert light into an electrical current.

 * Photoresistors: Change their resistance in response to light.

 * Phototransistors: Similar to photodiodes but use a transistor structure.

Pressure Sensors:

 * Strain Gauges: Measure pressure by detecting changes in the resistance of a wire or foil when it is stretched or compressed.

 * Piezoelectric Sensors: Generate a voltage when subjected to mechanical stress.

 * Capacitive Sensors: Measure pressure by detecting changes in capacitance.

Motion Sensors:

 * Accelerometers: Measure acceleration, including vibration and shock.

 * Gyroscopes: Measure angular velocity.

* Magnetometers: Measure magnetic fields.

Chemical Sensors:

* Gas Sensors: Detect the presence of specific gases.

* pH Sensors: Measure the acidity or alkalinity of a solution.

* Humidity Sensors: Measure the amount of water vapor in the air.

Other Sensors:

* Force Sensors: Measure force or pressure.

* Flow Sensors: Measure the flow rate of liquids or gases.

* Level Sensors: Measure the level of liquids or solids in a container.

Examples of Sensor Applications:

* Smartphones: Use a variety of sensors, including accelerometers, gyroscopes, magnetometers, and light sensors, for features like navigation, fitness tracking, and camera autofocus.

* Industrial Automation: Sensors are used to monitor and control processes in factories and manufacturing plants.

* Healthcare: Sensors are used in medical devices like heart rate monitors, blood pressure monitors, and glucose meters.

* Environmental Monitoring: Sensors are used to monitor air quality, water quality, and weather conditions.

The choice of sensor depends on the specific application and the physical quantity to be measured.

Q2)Write a short note on :

i) 12C bus protocol
ii) CAN bus protocol
iii) UART
iv) USRT

➔I) 12C Bus Protocol

12C (pronounced "I-squared-C") is a serial communication bus that uses a master-slave architecture. It is widely used in embedded systems due to its simplicity, low cost, and flexibility.

 * Master-Slave Architecture: There is one master device that initiates communication and controls the bus, while other devices act as slaves and respond to the master's requests.

 * Multi-Master Capable: While the basic architecture is master-slave, 12C can also support multiple masters.

 * Open-Drain Output: Devices on the bus share a common open-drain output, which means they must pull the bus low to transmit data.

 * Clock Stretching: Slaves can stretch the clock signal to indicate that they are busy or require additional time to process a request.

II) CAN Bus Protocol

CAN (Controller Area Network) is a high-speed serial communication bus that is commonly used in automotive and industrial applications. It is known for its robustness, reliability, and ability to handle real-time communication.

 * Multi-Master Architecture: Multiple devices can act as masters and initiate communication on the bus.

 * Priority-Based Arbitration: The bus uses a priority-based arbitration mechanism to resolve conflicts when multiple devices attempt to transmit simultaneously.

 * Error Detection and Correction: CAN includes features for error detection and correction, such as CRC (Cyclic Redundancy Check) and automatic retransmission.

 * Broadcast and Unicast: CAN supports both broadcast and unicast communication, allowing messages to be sent to all devices on the bus or to specific devices.

III) UART

UART (Universal Asynchronous Receiver-Transmitter) is a simple serial communication protocol that is widely used in microcontrollers and other embedded systems.

 * Asynchronous Transmission: Data is transmitted in asynchronous mode, meaning there is no clock signal shared between the transmitter and receiver.

 * Start and Stop Bits: Each data bit is preceded by a start bit and followed by a stop bit, which help to synchronize the transmitter and receiver.

 * Parity Bit: An optional parity bit can be used for error detection.

* Full-Duplex Communication: UART supports full-duplex communication, allowing data to be transmitted in both directions simultaneously.

IV) USRT

USRT (Universal Synchronous/Asynchronous Receiver-Transmitter) is a more advanced version of UART that supports both synchronous and asynchronous communication.

* Synchronous Mode: In synchronous mode, a clock signal is shared between the transmitter and receiver, allowing for higher data transfer rates.
* Frame Format: The frame format for synchronous communication includes a start bit, data bits, a parity bit, and a stop bit, similar to asynchronous communication.
* Clock Recovery: USRTs typically include circuitry for clock recovery, which allows the receiver to reconstruct the clock signal from the received data.

Note: While "USRT" is sometimes used as an abbreviation for "Universal Synchronous/Asynchronous Receiver-Transmitter," it is not a widely recognized term. The more common term is UART.

Q3) Draw and explain interfacing of LED with ARDIUNO with program to blink it.

➔Interfacing LED with Arduino and Blinking Code

Circuit Diagram:

* LED: Connect the longer leg (anode) to the Arduino's digital output pin (e.g., pin 13). Connect the shorter leg (cathode) to ground (GND).
* Resistor: Place a resistor (e.g., 220 ohms) in series with the LED to limit current and prevent it from burning out.

Arduino Code:

```
const int ledPin = 13; // Pin connected to the LED


void setup() {

  pinMode(ledPin, OUTPUT); // Set the LED pin as an output

}
```

```
void loop() {
  digitalWrite(ledPin, HIGH); // Turn the LED on
  delay(1000); // Wait for 1000 milliseconds (1 second)

  digitalWrite(ledPin, LOW); // Turn the LED off
  delay(1000); // Wait for 1000 milliseconds (1 second)
}
```

Explanation:

 * Include necessary libraries: If you're using additional libraries, include them at the beginning of your code.

 * Define the LED pin: Declare a constant variable ledPin to store the pin number connected to the LED.

 * Setup function:

   * pinMode(ledPin, OUTPUT): Sets the specified pin as an output pin.

 * Loop function:

   * digitalWrite(ledPin, HIGH): Turns the LED on by setting the output pin to HIGH (5 volts).

   * delay(1000): Pauses the program for 1000 milliseconds (1 second).

   * digitalWrite(ledPin, LOW): Turns the LED off by setting the output pin to LOW (0 volts).

   * delay(1000): Pauses the program for another 1 second.

This code will continuously turn the LED on and off, creating a blinking effect. You can adjust the delay times to change the blinking speed

Q4) Draw and explain interfacing of DC motor with ARDIUNO with program of speed control.

➔Interfacing a DC Motor with Arduino for Speed Control

Circuit Diagram:

* DC Motor: Connect the positive terminal (anode) of the motor to the collector or drain of the transistor. Connect the negative terminal (cathode) to ground (GND).
* Transistor: Use a transistor (e.g., MOSFET or BJT) to control the current flowing to the motor. The base or gate of the transistor should be connected to a digital output pin on the Arduino.
* Resistor: Place a resistor (e.g., 220 ohms) in series with the base or gate of the transistor to limit the base current and prevent the transistor from saturating.

Arduino Code:

```
const int motorPin = 9; // Pin connected to the transistor base or gate
const int speed = 150; // Adjust the speed value (0-255)

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

void loop() {
  analogWrite(motorPin, speed); // Set the motor speed
}
```

Explanation:

* Include necessary libraries: If you're using additional libraries, include them at the beginning of your code.
* Define the motor pin: Declare a constant variable motorPin to store the pin number connected to the transistor base or gate.
* Define the speed: Declare a constant variable speed to store the desired motor speed.
* Setup function:
  * pinMode(motorPin, OUTPUT): Sets the specified pin as an output pin.
* Loop function:

* analogWrite(motorPin, speed): Uses pulse-width modulation (PWM) to control the motor's speed. The speed value determines the duty cycle of the PWM signal, which affects the average voltage applied to the motor. A higher value results in a higher motor speed.

Note: The specific transistor and resistor values may vary depending on the motor's specifications and power requirements. It's important to choose components that can handle the motor's current and voltage.

By adjusting the speed variable, you can control the rotational speed of the DC motor. A higher value will result in a faster rotation, while a lower value will result in a slower rotation.