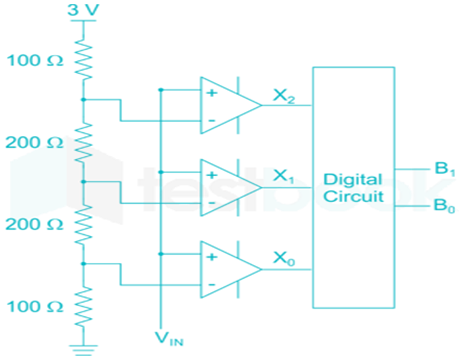
Tutorial Questions

1. List the differences between i) Analog and Digital Data ii) Serial and Parallel data transfer.
2. Explain the following 3 basic operations in Analog to Digital data conversion with its neat representations. i) Sampling, ii) Holding and iii) Quantization
3. Explain the following communication Protocols, with its frame formats,
   1. UART
   2. I2C
4. With neat diagram, explain the working of 4-bit Flash ADC.
5. With neat circuit diagram, explain the working of R2R ladder type DAC.
6. A two-bit flash ADC is shown in figure bellow. The input voltage varies from 0<Vin<5 Volts. Find the digital Output for a given input voltage Vin=2V. Mention the outputs of each stages in the circuit.



1. What are the three basic operations in analog-to-digital data conversion? Explain each operation briefly with the help of a diagram.
2. What is the SPI communication protocol, and how is it used in embedded systems? What are the advantages and disadvantages of using SPI over other protocols?
3. What are port pins and GPIOs in an Arduino board? How are they used in embedded systems design?
4. How does the I2C communication protocol work in embedded systems, and what are its key features?
5. How do you program the port pins and GPIOs in an Arduino board using the Arduino IDE? Can you provide an example code?
6. Write schematic diagram of interfacing Arduino to control led using push button. Write a programs to power on the LED when the button is pressed, and power off the LED when the button is not pressed.
7. With neat diagram, explain the working of 3-bit Flash ADC
8. A two-bit flash ADC is shown in figure.7. b. The input voltage varies from 0<Vin<5 Volts. Find the digital Output for a given input voltage Vin=3V. Mention the outputs of each stages in the circuit.
9. With neat circuit diagram, explain the working of Successive Approximation ADC Type
10. How do you measure and display the room temperature using an LM35 temperature sensor and an Arduino Uno R3 board? Provide an interfacing diagram.
11. How do you generate a PWM signal with a 75% duty cycle on pin number 3 using an Arduino board? Also, explain the principle of DC motor speed control using PWM technique.
12. Why are motor drivers necessary for interfacing motors with an Arduino board, and how does an H-Bridge motor driver circuit work?
13. Explain the working principles of DC and stepper motors using a neat diagram?
14. Write a program to rotate the DC motor in clock wise direction with 100rpm and anti-clockwise with 200rpm using Arduino and L298 H bridge IC.
15. Interfacing examples of mono color LED and rgb LED with arduino.

https://www.tinkercad.com/things/7Bww5k6RkB3-copy-of-arduino-projects-with-leds/editel?tenant=circuits

1. Interfacing switch to control led on and off with Arduino.

https://www.tinkercad.com/things/b0qGee4KNgA-copy-of-arduino-project-7-button-tactile-switch-and-led/editel?tenant=circuits

1. Different projects with Arduino using tinkercad.

https://ediylabs.com/contests/TCDC2021/projectlist

1. Interfacing TMP36 Temperature and humidity sensor to sense the temperature/humidity and LED’s to indicate its range.

https://www.tinkercad.com/things/72e73A9hVlm-copy-of-tmp36-temperature-sensor-with-arduino/editel?tenant=circuits

1. Theory to Interfacing LCD with Arduino.

https://www.elprocus.com/interface-lcd-liquid-crystal-display-using-arduino/#:~:text=LCD%20Interfacing%20with%20the%20Arduino%20Module&text=The%20LCD%20of%20R%2FW,to%20DB7%20of%20the%20LCD.

1. Interfacing LED with Arduino using tinkercad.

https://www.tinkercad.com/things/aSMJ27SsEVK-copy-of-arduino-with-lcd-interface/editel?tenant=circuits

1. Interfacing GAS sensor with Arduino using tinker cad

https://www.tinkercad.com/things/3AuT2IFiGSn-copy-of-activity11-arduino-gas-sensor/editel?tenant=circuits

1. Interfacing Stepper motor with Arduino using tinker cad

https://www.tinkercad.com/things/5q0jl0fxp8w-copy-of-stepper-motor-interface-with-arduino/editel?tenant=circuits

1. Note: Questions will be prepared by combining any of the above components to perform a particular application with Arduino microcontroller board.

SOLUTIONS

**1)List the differences between i) Analog and Digital Data ii) Serial and Parallel data transfer.**

ANS— RSST PCE

i) Representation:

Analog: Analog data is continuous and represented as a continuous waveform. It can take on an infinite number of values within a range.

Digital: Digital data is discrete and represented as a series of discrete values or digits. It can only take on a finite set of discrete values.

Signal Nature:

Analog: Analog signals are susceptible to noise and interference since they are continuous and can vary infinitely.

Digital: Digital signals are less susceptible to noise and interference because they use discrete values, and errors can be detected and corrected.

Storage:

Analog: Analog data is challenging to store and reproduce accurately, as it requires precise mechanisms to maintain fidelity.

Digital: Digital data is easy to store and reproduce accurately, making it more robust for long-term preservation.

Transmission:

Analog: Analog signals degrade over long distances, requiring amplification and filtering to maintain signal quality.

Digital: Digital signals can be transmitted over long distances without significant degradation, and errors can be corrected using error-checking techniques.

Processing:

Analog: Analog data is challenging to process using standard computing devices because it involves continuous values.

Digital: Digital data is well-suited for processing by computers, as they operate on discrete values and can perform operations with high precision.

Conversion:

Analog: To convert analog data to digital, an analog-to-digital converter (ADC) is required, which involves some loss of information.

Digital: Digital data can be converted to analog using a digital-to-analog converter (DAC), with minimal loss of information.

Examples:

Analog: Examples of analog data include analog audio signals (e.g., vinyl records, cassette tapes), analog thermometers, and analog watches.

Digital: Examples of digital data include digital audio files (e.g., MP3), digital temperature readings (e.g., digital thermometers), and digital clocks.

ii) DND SCS E

Data Transmission:

Serial: In serial data transfer, bits are sent one at a time sequentially over a single communication channel (e.g., a single wire or optical fiber).

Parallel: In parallel data transfer, multiple bits are sent simultaneously over multiple communication channels (e.g., multiple wires or traces).

Number of Wires/Channels:

Serial: Uses fewer wires or channels, typically just one or a few, which simplifies cabling and reduces cost.

Parallel: Requires more wires or channels, with the number of wires equal to the number of bits being transferred simultaneously, which can lead to increased complexity and cost.

Data Rate (Speed):

Serial: Generally has a lower data rate compared to parallel, as it sends one bit at a time.

Parallel: Typically offers higher data rates because it can transfer multiple bits simultaneously.

Synchronization:

Serial: Requires additional mechanisms for synchronization, such as start and stop bits or clock signals, to ensure the receiver interprets data correctly.

Parallel: Data is inherently synchronized, as all bits are transferred simultaneously.

Clocking:

Serial: May require a separate clock signal for timing, especially in asynchronous serial communication.

Parallel: Often relies on a shared clock signal to synchronize the transfer of multiple bits.

Signal Integrity:

Serial: Generally provides better signal integrity and is less susceptible to skew or timing issues because it transmits one bit at a time.

Parallel: More susceptible to signal integrity problems, such as skew and interference, due to the simultaneous transmission of multiple bits.

Applications:

Serial: Commonly used for long-distance communication, including data transmission over networks (e.g., Ethernet) and serial peripheral connections (e.g., USB, SATA).

Parallel: Used for short-distance, high-speed data transfers within computer systems, such as memory buses and some legacy interfaces (e.g., parallel ports).

**2)Explain the following 3 basic operations in Analog to Digital data conversion with its neat representations. i) Sampling, ii) Holding and iii) Quantization**

ANS—

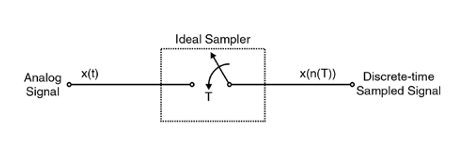
Analog-to-Digital (A/D) conversion is a process that involves converting continuous analog signals into discrete digital values. These operations together allow us to convert a continuous analog signal into a discrete digital representation for further processing and storage.

Sampling:

Sampling is the first step in A/D conversion and involves capturing the value of the analog signal at specific time intervals.

The continuous analog signal is sampled at regular intervals, known as the sampling rate or frequency (usually measured in Hertz, Hz).

Each sampled value represents the amplitude of the analog signal at a particular moment in time.

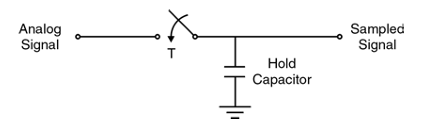


Holding (or Sample-and-Hold):

After sampling, the sampled value must be held constant for a brief period until it can be quantized into a digital value.

The purpose of holding is to ensure that the sampled value remains constant while it is processed by the subsequent quantization step.

This operation is typically performed by a sample-and-hold circuit that keeps the voltage level constant until quantization.



Quantization:

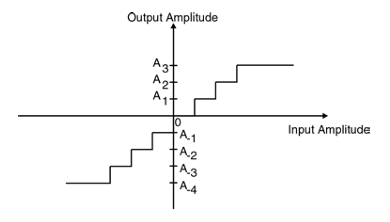
Quantization is the process of mapping the continuous range of analog values to a finite set of discrete digital values.

Each sampled and held value is rounded or approximated to the nearest quantization level, resulting in a digital representation.

Analog quantization size

Q =(Vmax-Vmin)/2N

Q is the Resolution



**3) Explain the following communication Protocols, with its frame formats,**

* 1. **UART**
  2. **I2C**

ANS—

SPI (Serial Peripheral Interface) is a synchronous serial communication protocol used for connecting microcontrollers, sensors, and other digital devices in embedded systems.

1. UART (Universal Asynchronous Receiver/Transmitter):

UART is a widely used serial communication protocol that facilitates the asynchronous transmission of data between two devices. It's commonly used for point-to-point communication between devices like microcontrollers, sensors, and other embedded systems.

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART.

UART Frame Format:

The UART frame format typically consists of the following components:

Start Bit: It's always a logic low (0) and indicates the beginning of the data frame. It serves as a synchronization signal.

Data Bits: These represent the actual data to be transmitted (usually 8 bits, but it can be 5, 6, 7, or 8 bits depending on the configuration).

Parity Bit (optional): The parity bit is used for error checking. It can be odd, even, or disabled. Its value is determined by the number of set bits in the data bits.

Stop Bit(s): One or more stop bits (usually 1 or 2) indicate the end of the data frame and provide time for the receiver to prepare for the next frame.

b. I2C (Inter-Integrated Circuit):

I2C is a synchronous, multi-master, multi-slave communication protocol often used for interconnecting multiple digital integrated circuits on a PCB (Printed Circuit Board). It uses a master-slave architecture, where one or more master devices communicate with one or more slave devices.

I2C Frame Format:

The I2C frame format is quite different from UART, as it is a serial, two-wire bus protocol. It consists of the following components:

Start Condition: The communication begins with the master device sending a start condition (S) on the bus. It indicates the start of a communication transaction.

7-bit or 10-bit Slave Address: The master sends the 7-bit or 10-bit address of the slave device it wants to communicate with. This address specifies the target device on the bus.

Read/Write Bit (R/W): This bit indicates whether the master wants to read from (R) or write to (W) the slave device.

Data: Data bytes are sent between the master and slave, with acknowledgment (ACK) or no acknowledgment (NACK) after each byte.

Stop Condition: The communication ends with the master sending a stop condition (P) on the bus, indicating the end of the transaction.

**4) With neat diagram, explain the working of 4-bit Flash ADC.**

ANS—

A 4-bit Flash Analog-to-Digital Converter (ADC) is a type of ADC that directly converts an analog input voltage into a digital output in a very fast and parallel manner. It's called a "Flash" ADC because it operates by comparing the input voltage to a set of reference voltages using a series of voltage comparators.

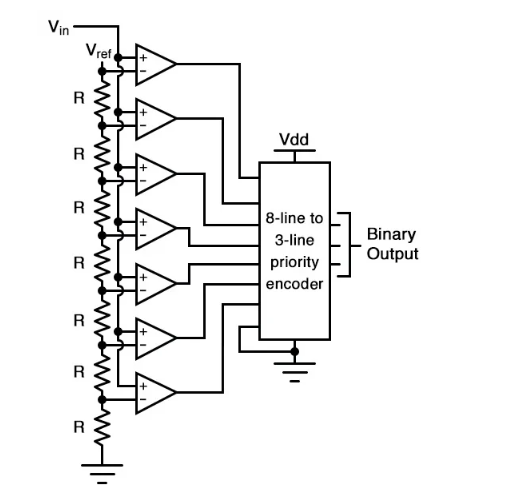
Working Principle:

A 4-bit Flash ADC has 16 possible output codes, which correspond to the 16 possible voltage ranges that the input can fall into. Each bit of the output represents a different voltage range. The most significant bit (MSB) is the highest voltage range, and the least significant bit (LSB) is the lowest voltage range.

Explanation of the Working:

* The input analog voltage is compared simultaneously with 16 different reference voltages using 4-bit digital comparators. Each comparator corresponds to one of the 16 possible output codes (0000 to 1111).
* Each comparator generates a digital output (either HIGH or LOW) based on whether the input voltage is greater or less than the reference voltage it is comparing against.
* The outputs of these comparators are then fed into an encoding logic circuit that determines the 4-bit binary code representing the range in which the input voltage falls. This binary code is the digital output of the ADC.
* The encoding logic generates the 4-bit digital output based on which comparators produced a HIGH output. For example, if the input voltage falls between the references corresponding to comparators D2 and D3 (in the diagram), the output code might be 1100, indicating that the input voltage is closer to the reference voltage represented by D3.
* The output code represents a digital representation of the input voltage, allowing you to process it digitally in a microcontroller or other digital circuits.

DIAGRAM OF 3 BIT FLASH ADC



 For N-bit flash ADC  urn:x-wiley:1751858X:media:cds2bf00430:cds2bf00430-math-0001 resistors are required to generate a reference voltage for the  urn:x-wiley:1751858X:media:cds2bf00430:cds2bf00430-math-0002 comparators. The output of the comparator is a bit-string called thermometer code. A simple urn:x-wiley:1751858X:media:cds2bf00430:cds2bf00430-math-0003 encoder will convert thermometer code to binary code.

**5)With neat circuit diagram, explain the working of R2R ladder type DAC.**

ANS-

The R2R ladder DAC is a popular type of digital-to-analog converter that converts a digital input (binary code) into an analog voltage output. It consists of a ladder-like network of resistors, with two different resistor values: R and 2R.

* When a binary input code is applied to the DAC, each bit (0 or 1) selects whether to include the corresponding resistor in the ladder network in the signal path.
* For a '1' bit, the 2R resistor is included in the signal path, while for a '0' bit, it's the R resistor. The voltage drop across each resistor in the ladder network depends on the bit value: V = I \* R, where I is the current flowing through the resistor and R is the resistance value.
* As you move from the LSB to the MSB, the voltage drop across the resistors in the ladder network doubles with each bit (binary weighting), resulting in a weighted sum of voltages at the summing amplifier's input.
* The summing amplifier adds up all these weighted voltages to produce an analog output voltage that represents the digital input code.



3 BIT R-2R LADDER DAC CIRCUIT

**6) What are port pins and GPIOs in an Arduino board? How are they used in embedded systems design?**

ANS—

In an Arduino board, port pins and GPIOs (General-Purpose Input/Output pins) are essential elements used for interfacing with external components and sensors in embedded systems design.

Port Pins:

Port pins are the physical pins on the Arduino microcontroller that can be used for various purposes, including digital input/output, analog input, and communication (e.g., UART, SPI, I2C).

Arduino boards typically have a set of digital pins and analog pins.

Digital pins can be used as either inputs or outputs, allowing you to read digital signals (high or low) or send digital signals to control external devices.

Analog pins are used for analog-to-digital conversion (ADC), allowing you to read analog signals from sensors such as temperature sensors, light sensors, and potentiometers.

GPIOs (General-Purpose Input/Output Pins):

GPIOs are a subset of the port pins, specifically referring to pins that can be configured as either inputs or outputs in a general-purpose manner.

GPIO pins can be used to interact with a wide range of digital components and sensors, making them highly versatile for embedded systems.

You can use GPIO pins to read switches, buttons, and sensors, as well as to control LEDs, relays, motors, and other digital devices.

These pins allow you to interface with the physical world and make decisions based on external inputs or control external outputs.

In embedded systems design, port pins and GPIOs are crucial because they provide the means to interact with the external environment and sensors. Here's how they are typically used:

Input Handling:

Port pins and GPIOs can be configured as digital inputs to read the state of external devices or sensors. For example, you can read the state of a button press, detect motion with a PIR sensor, or measure temperature with a temperature sensor.

Output Control:

Port pins and GPIOs can be configured as digital outputs to control external components. For instance, you can turn on and off LEDs, drive a relay to control high-power devices, or control the direction and speed of motors.

Analog Sensing:

Analog pins allow you to interface with analog sensors and convert their continuous voltage values into digital data using the built-in analog-to-digital converter (ADC).

Communication:

Some GPIO pins are multipurpose and can be used for communication protocols like UART, SPI, or I2C, enabling Arduino boards to communicate with other microcontrollers, sensors, or displays.

**7) Write a program to rotate the DC motor in clock wise direction with 100rpm and anti-clockwise with 200rpm using Arduino and L298 H bridge IC.**

ANS—

// Motor control pins

int motorEnableA = 9; // Enable pin for Motor A (ENA)

int motorInput1A = 8; // Input 1 pin for Motor A

int motorInput2A = 7; // Input 2 pin for Motor A

void setup() {

// Set motor control pins as outputs

pinMode(motorEnableA, OUTPUT);

pinMode(motorInput1A, OUTPUT);

pinMode(motorInput2A, OUTPUT);

// Initialize serial communication for debugging

Serial.begin(9600);

}

void loop() {

// Rotate the motor clockwise at 100 RPM

rotateClockwise(100);

delay(3000); // Delay for 3 seconds

// Rotate the motor counterclockwise at 200 RPM

rotateCounterclockwise(200);

delay(3000); // Delay for 3 seconds

}

// Function to rotate the motor clockwise at a specified RPM

void rotateClockwise(int rpm) {

// Calculate the delay between motor steps based on RPM

int delayTime = 60000 / (rpm \* 64); // 64 steps per revolution

// Set motor control pins for clockwise rotation

digitalWrite(motorInput1A, HIGH);

digitalWrite(motorInput2A, LOW);

// Enable the motor

analogWrite(motorEnableA, 255); // 255 is maximum speed

// Rotate the motor for a specific duration

delay(delayTime);

// Stop the motor

analogWrite(motorEnableA, 0);

}

// Function to rotate the motor counterclockwise at a specified RPM

void rotateCounterclockwise(int rpm) {

// Calculate the delay between motor steps based on RPM

int delayTime = 60000 / (rpm \* 64); // 64 steps per revolution

// Set motor control pins for counterclockwise rotation

digitalWrite(motorInput1A, LOW);

digitalWrite(motorInput2A, HIGH);

// Enable the motor

analogWrite(motorEnableA, 255); // 255 is maximum speed

// Rotate the motor for a specific duration

delay(delayTime);

// Stop the motor

analogWrite(motorEnableA, 0);

}