Institute of E-Mobility
DIYGURU.ORG
An Institute impanelled with NEAT, AICTE, Ministry of Education, Govt. of India. Approved by Automotive Skills Development Council.

impanelled by N·E·A·T प्रौद्योगिकी के लिए राष्ट्रीय शैक्षणिक सहयोग National Educational Alliance for Technology

Ministry of Education Government of India

certified by ASDC AUTOMOTIVE SKILLS DEVELOPMENT COUNCIL

EXPERIMENTAL MANUAL

Electronics & Embedded Hardware Training

MAKE | HACK | INVENT

DIYguru E Mobility

TABLE OF CONTENTS

Experiment 1: LED Blinker: Implementing Control of LEDs Using Arduino GPIO Pins

---

Experiment Title: LED Blinker: Implementing Control of LEDs Using GPIO Pins

Objective: To create various LED blinking patterns using the GPIO pins of an Arduino.

Description: This experiment involves using an Arduino microcontroller to control multiple LEDs through its GPIO pins. By programming the Arduino, we can turn the LEDs on and off in different sequences, creating various visual patterns. This basic yet fundamental exercise helps in understanding how to interface LEDs with a microcontroller, which is a crucial skill in the field of electronics and embedded systems.

Materials Needed:

- Arduino Uno
- LEDs (3-5 different colors)
- Resistors (220 ohms)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: LEDs (Light Emitting Diodes) are semiconductor devices that emit light when an electric current passes through them. The Arduino Uno is a popular microcontroller board that can be programmed to control devices like LEDs through its digital I/O pins. Understanding how to manipulate these pins to create patterns can form the basis for more complex projects, such as indicator lights, displays, and user interfaces.

Pre-Lab Questions:

1. What is a GPIO pin and how does it function on an Arduino?
2. Why are resistors necessary when connecting LEDs to a microcontroller?
3. What are some common applications of LED blinking patterns in real-world electronics?

Procedure:

1. **Setup the Circuit:**

- Place the LEDs and resistors on the breadboard.
- Connect the LEDs to the designated digital pins on the Arduino.
- Ensure the connections follow the proper orientation and polarity of the LEDs.

2. **Program the Arduino:**
    - Write a program to control the LEDs through the digital pins.
    - Use basic functions to turn the LEDs on and off in different sequences.
3. **Upload and Test:**
    - Upload the program to the Arduino.
    - Observe the LED blinking patterns and ensure they match the expected behavior.
4. **Modify Patterns:**
    - Experiment with different blinking patterns by changing the code.
    - Document the changes and their effects on the LED behavior.

Post-Lab Questions:

1. How did changing the delay values in the code affect the LED blinking pattern?
2. What challenges did you encounter during the experiment and how did you overcome them?
3. How can this basic LED control concept be applied to more complex systems?

Experiment 1b: LED Blinker: Implementing Control of LEDs Using ARM GPIO Pins

Objective: To create various LED blinking patterns using the GPIO pins of an STM32 development board.

Description: In this experiment, we will use an STM32 microcontroller to control several LEDs via its GPIO pins. By configuring and programming the STM32, we will develop various LED blinking sequences. This exercise is designed to introduce the fundamentals of GPIO control in STM32 microcontrollers, which are widely used in embedded systems and industrial applications.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- LEDs (3-5 different colors)
- Resistors (220 ohms)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: STM32 microcontrollers are known for their high performance and rich peripheral set. Controlling LEDs through GPIO pins on an STM32 board involves configuring the pins correctly and programming the microcontroller to manipulate these pins. This basic project serves as a stepping stone to more advanced applications in embedded systems, such as creating interactive devices and monitoring systems.

Pre-Lab Questions:

1. What are the differences between Arduino and STM32 in terms of GPIO control?
2. Why is it important to configure the GPIO pins correctly in STM32?
3. What real-world applications can benefit from precise LED control?

Procedure:

1. **Setup the Circuit:**
    - Connect the LEDs and resistors to the breadboard.
    - Attach the LEDs to the appropriate GPIO pins on the STM32 board.

    - Double-check all connections for correct orientation.
2. **Configure the Microcontroller:**
    - Set up the STM32 project environment.
    - Configure the GPIO pins for output mode using the STM32CubeMX or equivalent tool.
3. **Program the STM32 Board:**
    - Write and load the program to control the LEDs.
    - Implement different sequences for the LEDs to blink.
4. **Test and Observe:**
    - Run the program on the STM32 board.
    - Observe the LED patterns and ensure they perform as expected.
5. **Experiment with Patterns:**
    - Modify the program to create new blinking sequences.
    - Note the effects of these changes on the LED behavior.

Post-Lab Questions:

1. How did the configuration of GPIO pins affect the overall LED control?
2. What did you learn about the timing and control of LEDs in an STM32 environment?
3. How can the skills learned in this experiment be applied to more advanced projects?

Safety Note: Ensure all connections are correct and secure before powering the boards. Always double-check the orientation of LEDs and resistors to prevent damage to the components and the microcontroller.

Experiment 2a: PWM LED Dimmer: Adjusting LED Brightness Using Arduino Pulse Width Modulation (PWM)

Experiment Title: PWM LED Dimmer: Adjusting LED Brightness Using Pulse Width Modulation (PWM)

Objective: To adjust the brightness of an LED using Pulse Width Modulation (PWM) with an Arduino.

Description: In this experiment, we will use an Arduino to control the brightness of an LED through PWM. PWM is a technique used to simulate an analog output using digital signals by varying the duty cycle of a digital signal. This experiment will help in understanding how PWM works and how it can be used to control devices like LEDs.

Materials Needed:

- Arduino Uno
- LED (1)
- Resistor (220 ohms)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Pulse Width Modulation (PWM) is a method of controlling the amount of power delivered to an electronic load by varying the duty cycle of a square wave signal. The duty cycle is the fraction of one period in which the signal is active. By adjusting the duty cycle, we can control the brightness of an LED. This technique is widely used in applications such as motor control, light dimming, and signal generation.

Pre-Lab Questions:

1. What is Pulse Width Modulation (PWM) and how does it work?
2. How does changing the duty cycle of a PWM signal affect the brightness of an LED?
3. What are some practical applications of PWM in electronics?

Procedure:

1. **Setup the Circuit:**
    - Connect the anode (longer leg) of the LED to a PWM-capable digital pin on the Arduino (e.g., pin 9).
    - Connect the cathode (shorter leg) of the LED to a resistor, and then connect the other end of the resistor to the ground rail on the breadboard.
    - Connect the ground rail to the GND pin on the Arduino.
2. **Program the Arduino:**
    - Write a program to output a PWM signal on the designated pin.
    - Adjust the duty cycle of the PWM signal to vary the brightness of the LED.
3. **Upload and Test:**
    - Upload the program to the Arduino.
    - Observe the LED brightness changes as the duty cycle is varied.
4. **Experiment with Different Duty Cycles:**
    - Modify the program to gradually increase and decrease the duty cycle, creating a breathing effect.
    - Document the changes and their effects on the LED brightness.

Post-Lab Questions:

1. How did changing the duty cycle affect the LED brightness?
2. What challenges did you encounter during the experiment and how did you overcome them?
3. How can PWM be used in other applications besides LED dimming?

Experiment 2b: PWM LED Dimmer: Adjusting LED Brightness Using ARM based Pulse Width Modulation (PWM)

Objective: To adjust the brightness of an LED using Pulse Width Modulation (PWM) with an STM32 development board.

Description: In this experiment, we will use an STM32 microcontroller to control the brightness of an LED through PWM. PWM allows us to simulate varying levels of power output by changing the duty cycle of a digital signal. This experiment will help in understanding the principles of PWM and its application in controlling LED brightness.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- LED (1)
- Resistor (220 ohms)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Pulse Width Modulation (PWM) is a technique used to control the power delivered to electrical devices by varying the duty cycle of a digital signal. The duty cycle determines the proportion of time the signal is high versus low. By varying the duty cycle, the average power delivered to the LED changes, thereby adjusting its brightness. This method is efficient and widely used in various applications such as motor speed control, signal modulation, and power regulation.

Pre-Lab Questions:

1. What is the principle behind Pulse Width Modulation (PWM)?
2. How does the duty cycle of a PWM signal influence the perceived brightness of an LED?
3. Can you name other devices or systems that utilize PWM for control?

Procedure:

1. **Setup the Circuit:**

- Connect the anode (longer leg) of the LED to a PWM-capable GPIO pin on the STM32 board (e.g., pin PA0).
- Connect the cathode (shorter leg) of the LED to a resistor, and then connect the other end of the resistor to the ground rail on the breadboard.
- Connect the ground rail to a GND pin on the STM32 board.

2. **Configure the Microcontroller:**
   - Set up the STM32 project environment.
   - Configure the GPIO pin for PWM output using STM32CubeMX or an equivalent tool.
3. **Program the STM32 Board:**
   - Write a program to generate a PWM signal on the designated pin.
   - Adjust the duty cycle of the PWM signal to control the LED brightness.
4. **Test and Observe:**
   - Upload the program to the STM32 board.
   - Observe the changes in LED brightness as the duty cycle varies.
5. **Experiment with Different Duty Cycles:**
   - Modify the program to create different duty cycle patterns, such as a gradual increase and decrease in brightness.
   - Document the changes and their effects on the LED brightness.

Post-Lab Questions:

1. How did varying the duty cycle impact the LED brightness?
2. What observations did you make regarding the smoothness of brightness transitions?
3. How can the concepts learned in this experiment be applied to other areas of embedded systems?

Safety Note: Ensure all connections are correct and secure before powering the boards. Double-check the orientation of the LED and resistor to prevent damage to the components and the microcontroller.

Experiment 3a: Serial Communication Setup: Setting Up Arduino UART Communication for Debugging and Data Transfer

---

Experiment Title: Serial Communication Setup: Setting Up UART Communication for Debugging and Data Transfer

Objective: To set up UART (Universal Asynchronous Receiver/Transmitter) communication for debugging and data transfer between an Arduino and a computer.

Description: In this experiment, we will use an Arduino to establish UART communication with a computer. UART is a hardware communication protocol that uses serial communication to send and receive data. This experiment will help in understanding how to set up serial communication, how to send and receive data, and how to use this for debugging purposes.

Materials Needed:

- Arduino Uno
- USB cable for programming and communication
- Computer with Arduino IDE installed

Background Information: UART is a type of asynchronous serial communication protocol where data is transmitted one bit at a time at a fixed baud rate. It is commonly used for communication between microcontrollers and other devices, including computers. The Arduino has a built-in UART interface that can be accessed via the USB connection, allowing for easy setup of serial communication.

Pre-Lab Questions:

1. What is UART and how does it differ from other communication protocols?
2. How does asynchronous communication work in UART?
3. What are the common baud rates used in serial communication?

Procedure:

1. **Connect the Arduino:**
    - Connect the Arduino to the computer using the USB cable.

2. **Setup Serial Communication:**
    - Open the Arduino IDE and create a new sketch.
    - Use the `Serial.begin()` function to initialize serial communication at a specified baud rate (e.g., 9600 bps).
3. **Write the Code:**
    - Write a program to send data from the Arduino to the computer using `Serial.print()` or `Serial.println()`.
    - Write code to read incoming data from the computer using `Serial.read()`.
4. **Upload and Monitor:**
    - Upload the code to the Arduino.
    - Open the Serial Monitor in the Arduino IDE to view the data being sent and received.
5. **Test Data Transfer:**
    - Send data from the computer to the Arduino and observe the response.
    - Use the serial communication for debugging by printing variable values and program states.

Post-Lab Questions:

1. How did you set the baud rate for UART communication and why is it important?
2. What did you observe when sending data from the Arduino to the computer and vice versa?
3. How can serial communication be used for debugging and monitoring in embedded systems?

Experiment 3b: Serial Communication Setup: Setting Up STM UART Communication for Debugging and Data Transfer

Objective: To set up UART (Universal Asynchronous Receiver/Transmitter) communication for debugging and data transfer between an STM32 development board and a computer.

Description: In this experiment, we will use an STM32 microcontroller to establish UART communication with a computer. UART is a widely used protocol for serial communication that enables data transfer between devices. This experiment will cover the setup of UART communication, sending and receiving data, and using it for debugging purposes.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- USB-to-Serial adapter (if necessary)
- Jumper wires
- USB cable for programming and communication
- Computer with STM32CubeIDE or equivalent installed

Background Information: UART is a popular asynchronous serial communication protocol used in embedded systems for data exchange. It operates by transmitting data in a series of bits at a predefined baud rate, ensuring that both the sender and receiver are synchronized. UART is essential for tasks such as debugging, data logging, and inter-device communication.

Pre-Lab Questions:

1. What is UART and what are its key features?
2. How does UART ensure data integrity and synchronization between devices?
3. Why is baud rate selection crucial in UART communication?

Procedure:

1. **Setup the Circuit:**
    - Connect the UART TX (transmit) pin of the STM32 to the RX (receive) pin of the USB-to-Serial adapter.
    - Connect the UART RX pin of the STM32 to the TX pin of the USB-to-Serial adapter.

    - Connect the ground (GND) of the STM32 to the ground of the USB-to-Serial adapter.
    - Connect the USB-to-Serial adapter to the computer.
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the UART peripheral (e.g., USART1) with the desired baud rate (e.g., 9600 bps).
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Write a program to initialize UART communication using the HAL library.
    - Implement code to send data from the STM32 to the computer using `HAL_UART_Transmit()`.
    - Implement code to receive data from the computer using `HAL_UART_Receive()`.
4. **Upload and Monitor:**
    - Compile and upload the code to the STM32 board.
    - Open a serial terminal on the computer (e.g., PuTTY, Tera Term) and connect to the appropriate COM port.
    - Monitor the data being sent and received.
5. **Test Data Transfer:**
    - Send data from the computer to the STM32 and observe the response.
    - Use the serial communication for debugging by printing variable values and program states.

Post-Lab Questions:

1. How did you configure the UART settings in STM32CubeMX and why are they important?
2. What observations did you make when sending and receiving data via UART?
3. How can UART communication be utilized for debugging and monitoring in complex embedded systems?

Safety Note: Ensure all connections are correct and secure before powering the boards. Double-check the TX and RX connections to prevent communication errors.

Experiment 4a: I2C Sensor Integration: Interfacing with Arduino I2C Sensors like MPU6050 and BMP280 to Read and Process Data

---

Experiment Title: I2C Sensor Integration: Interfacing with I2C Sensors like MPU6050 and BMP280 to Read and Process Data

Objective: To interface and communicate with I2C sensors, such as MPU6050 (Accelerometer and Gyroscope) and BMP280 (Barometric Pressure and Temperature Sensor), using an Arduino.

Description: In this experiment, we will use an Arduino to read data from I2C sensors, specifically the MPU6050 and BMP280. I2C (Inter-Integrated Circuit) is a multi-master, multi-slave, packet-switched, single-ended, serial communication bus used for attaching low-speed peripherals to processors and microcontrollers. This experiment helps in understanding how to set up I2C communication and read sensor data for processing.

Materials Needed:

- Arduino Uno
- MPU6050 sensor module
- BMP280 sensor module
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: I2C is a serial communication protocol that uses two lines, SDA (Serial Data) and SCL (Serial Clock), to transfer data between devices. It is commonly used in sensors and other peripherals. MPU6050 is an accelerometer and gyroscope sensor, while BMP280 is a barometric pressure and temperature sensor. Integrating these sensors with an Arduino involves setting up I2C communication and reading data for various applications.

Pre-Lab Questions:

1. What is I2C communication and how does it work?
2. What are the key features of the MPU6050 and BMP280 sensors?
3. How can the data from these sensors be used in real-world applications?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the MPU6050 and BMP280 to the 3.3V/5V and GND pins on the Arduino.
    - Connect the SDA (data) pin of both sensors to the A4 pin on the Arduino.
    - Connect the SCL (clock) pin of both sensors to the A5 pin on the Arduino.
2. **Program the Arduino:**
    - Use the Wire library to initialize I2C communication.
    - Write code to initialize the MPU6050 and BMP280 sensors.
    - Implement functions to read and process data from both sensors.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Open the Serial Monitor to view the data being read from the sensors.
4. **Process Sensor Data:**
    - Implement calculations to interpret the raw data from the MPU6050 (e.g., acceleration, angular velocity).
    - Read and convert data from the BMP280 to obtain temperature and pressure readings.
5. **Experiment with Data:**
    - Modify the code to display sensor data in a more readable format.
    - Document the changes and their effects on data presentation.

Post-Lab Questions:

1. How did you initialize and read data from the I2C sensors?
2. What challenges did you encounter during the experiment and how did you resolve them?
3. How can the data from the MPU6050 and BMP280 be used in practical applications?

Experiment 4b: I2C Sensor Integration: Interfacing with STM I2C Sensors like MPU6050 and BMP280 to Read and Process Data

Objective: To interface and communicate with I2C sensors, such as MPU6050 (Accelerometer and Gyroscope) and BMP280 (Barometric Pressure and Temperature Sensor), using an STM32 development board.

Description: In this experiment, we will use an STM32 microcontroller to read data from I2C sensors, specifically the MPU6050 and BMP280. I2C (Inter-Integrated Circuit) is a widely used serial communication protocol for connecting peripherals to microcontrollers. This experiment will cover the setup of I2C communication, reading sensor data, and processing it for various applications.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- MPU6050 sensor module
- BMP280 sensor module
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: I2C is a popular protocol for interconnecting integrated circuits using two lines: SDA (data) and SCL (clock). The MPU6050 sensor provides accelerometer and gyroscope data, while the BMP280 sensor provides barometric pressure and temperature readings. Using an STM32 microcontroller, we can set up I2C communication to read and process data from these sensors, which can be used in various applications like motion tracking and environmental monitoring.

Pre-Lab Questions:

1. What is I2C communication and what are its advantages?
2. What are the key features and functionalities of the MPU6050 and BMP280 sensors?
3. How can sensor data be applied in real-world scenarios?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the MPU6050 and BMP280 to the 3.3V/5V and GND pins on the STM32 board.
    - Connect the SDA (data) pin of both sensors to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
    - Connect the SCL (clock) pin of both sensors to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the I2C peripheral (e.g., I2C1) with the appropriate settings.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize I2C communication using the HAL library.
    - Write functions to initialize the MPU6050 and BMP280 sensors.
    - Implement functions to read and process data from both sensors.
4. **Upload and Monitor:**
    - Compile and upload the code to the STM32 board.
    - Use a serial terminal to view the data being read from the sensors.
5. **Process Sensor Data:**
    - Implement calculations to interpret the raw data from the MPU6050 (e.g., acceleration, angular velocity).
    - Read and convert data from the BMP280 to obtain temperature and pressure readings.
6. **Experiment with Data:**
    - Modify the code to improve the presentation and accuracy of sensor data.
    - Document the changes and their effects on data output.

Post-Lab Questions:

1. How did you configure the I2C communication for the STM32 microcontroller?
2. What challenges did you face while interfacing with the I2C sensors and how did you overcome them?
3. How can the processed data from the MPU6050 and BMP280 be utilized in practical applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors to prevent damage to the components and the microcontroller.

Experiment 5a: Real-time Sensor Data Display: Displaying Arduino Real-time Data from Various Sensors on the LCD

Experiment Title: Real-time Sensor Data Display: Displaying Real-time Data from Various Sensors on the LCD

Objective: To display real-time data from various sensors on an LCD using an Arduino.

Description: In this experiment, we will use an Arduino to read data from multiple sensors and display this data in real-time on an LCD screen. This involves setting up the sensors, reading their data, and using the LiquidCrystal library to display the information on an LCD. This experiment demonstrates how to create a simple data monitoring system.

Materials Needed:

- Arduino Uno
- LCD (16x2)
- MPU6050 sensor module (Accelerometer and Gyroscope)
- BMP280 sensor module (Barometric Pressure and Temperature Sensor)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Background Information: Displaying real-time data on an LCD is a common requirement in many applications, including weather stations, health monitoring systems, and industrial controls. The LiquidCrystal library in Arduino makes it easy to interface with LCDs, allowing for the display of sensor data, system status, and other information. This project involves integrating multiple sensors, reading their data via I2C, and updating the display in real-time.

Pre-Lab Questions:

1. How does an LCD display work and what are its key components?
2. What are the basics of the LiquidCrystal library in Arduino?
3. How can real-time data from sensors be efficiently displayed on an LCD?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the MPU6050 and BMP280 sensors to the 3.3V/5V and GND pins on the Arduino.
    - Connect the SDA (data) pin of both sensors to the A4 pin on the Arduino.
    - Connect the SCL (clock) pin of both sensors to the A5 pin on the Arduino.
    - Connect the LCD to the Arduino as follows:
        - RS to digital pin 12
        - E to digital pin 11
        - D4 to digital pin 5
        - D5 to digital pin 4
        - D6 to digital pin 3
        - D7 to digital pin 2
    - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Program the Arduino:**
    - Initialize the LiquidCrystal library to control the LCD.
    - Initialize I2C communication to read data from the MPU6050 and BMP280 sensors.
    - Write code to read sensor data and display it on the LCD.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Ensure the LCD displays the sensor data in real-time.
4. **Format and Update Data:**
    - Adjust the code to format the sensor data for clear and concise display on the LCD.
    - Implement periodic updates to refresh the displayed data.

Post-Lab Questions:

1. How did you initialize and configure the LCD display using the LiquidCrystal library?
2. What methods did you use to read and display real-time data from the sensors?
3. How can this project be extended to include additional sensors or display more complex information?

Experiment 5b: Real-time Sensor Data Display: Displaying STM Real-time Data from Various Sensors on the LCD

Objective: To display real-time data from various sensors on an LCD using an STM32 development board.

Description: In this experiment, we will use an STM32 microcontroller to read data from multiple sensors and display it in real-time on an LCD screen. This involves setting up the sensors, reading their data through I2C communication, and using the LCD library to display the information. This experiment demonstrates how to create a simple real-time data monitoring system using STM32.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- LCD (16x2)
- MPU6050 sensor module (Accelerometer and Gyroscope)
- BMP280 sensor module (Barometric Pressure and Temperature Sensor)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Background Information: Displaying real-time sensor data on an LCD is essential in many applications such as environmental monitoring, medical devices, and industrial automation. The STM32 microcontroller can interface with an LCD using parallel communication and read sensor data via I2C. This project involves setting up these interfaces and continuously updating the display with real-time data.

Pre-Lab Questions:

1. How does an LCD display work and what are its key components?
2. What is the process for setting up I2C communication on an STM32 microcontroller?
3. How can real-time data from sensors be efficiently displayed on an LCD?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the MPU6050 and BMP280 sensors to the 3.3V/5V and GND pins on the STM32.
    - Connect the SDA (data) pin of both sensors to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
    - Connect the SCL (clock) pin of both sensors to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
    - Connect the LCD to the STM32 as follows:
        - RS to a GPIO pin (e.g., PC0)
        - E to a GPIO pin (e.g., PC1)
        - D4 to a GPIO pin (e.g., PC2)
        - D5 to a GPIO pin (e.g., PC3)
        - D6 to a GPIO pin (e.g., PC4)
        - D7 to a GPIO pin (e.g., PC5)
    - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the I2C peripheral (e.g., I2C1) and the GPIO pins for the LCD.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize I2C communication and the LCD library.
    - Write functions to read data from the MPU6050 and BMP280 sensors.
    - Write code to display the sensor data on the LCD.
4. **Upload and Monitor:**
    - Compile and upload the code to the STM32 board.
    - Ensure the LCD displays the sensor data in real-time.
5. **Format and Update Data:**
    - Adjust the code to format the sensor data for clear and concise display on the LCD.
    - Implement periodic updates to refresh the displayed data.

Post-Lab Questions:

1. How did you initialize and configure the LCD display using the LCD library on STM32?

2. What methods did you use to read and display real-time data from the sensors?
3. How can this project be extended to include additional sensors or display more complex information?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors and LCD to prevent damage to the components and the microcontroller.

Experiment 6a: Interactive Display: Implementing Arduino Dynamic Data Display and User Interaction via the LCD

Experiment Title: Interactive Display: Implementing Dynamic Data Display and User Interaction via the LCD

Objective: To implement a dynamic data display and enable user interaction using buttons and an LCD with an Arduino.

Description: In this experiment, we will create an interactive display using an Arduino, an LCD, and a set of buttons. The LCD will display dynamic data, and the buttons will allow the user to interact with and control the display. This experiment demonstrates how to combine input and output devices to create interactive systems.

Materials Needed:

- Arduino Uno
- LCD (16x2)
- Push buttons (3-4)
- Resistors (10k ohms)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Background Information: An interactive display system allows users to interact with and control the information shown on the screen. By using buttons as input devices, we can create a system where the user can navigate through menus, adjust settings, or view different data sets. This is fundamental in creating user interfaces for embedded systems.

Pre-Lab Questions:

1. How does an LCD display work and how can it be controlled using an Arduino?
2. What is the role of pull-down resistors in button circuits?
3. How can multiple buttons be used to interact with an Arduino program?

Procedure:

1. **Setup the Circuit:**
    - Connect the LCD to the Arduino as follows:
        - RS to digital pin 12
        - E to digital pin 11
        - D4 to digital pin 5
        - D5 to digital pin 4
        - D6 to digital pin 3
        - D7 to digital pin 2
    - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
    - Connect each push button between a digital pin and ground, and use a pull-down resistor (10k ohms) for each button.
    - Example button connections:
        - Button 1 to digital pin 6
        - Button 2 to digital pin 7
        - Button 3 to digital pin 8
2. **Program the Arduino:**
    - Initialize the LiquidCrystal library to control the LCD.
    - Write code to read the state of the buttons.
    - Implement functions to update the LCD based on button presses.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Ensure the LCD displays dynamic data and responds to button presses.
4. **Implement Interactivity:**
    - Create a menu system or dynamic display that changes based on user input.
    - Document the functionality and user interaction with the system.

Post-Lab Questions:

1. How did you implement the button reading in the Arduino code?
2. What challenges did you encounter while creating the interactive display and how did you overcome them?
3. How can this concept be applied to more complex user interfaces?

Experiment 6b: Interactive Display: Implementing STM Dynamic Data Display and User Interaction via the LCD

Objective: To implement a dynamic data display and enable user interaction using buttons and an LCD with an STM32 development board.

Description: In this experiment, we will create an interactive display using an STM32 microcontroller, an LCD, and a set of buttons. The LCD will display dynamic data, and the buttons will allow the user to interact with and control the display. This experiment demonstrates how to combine input and output devices to create interactive systems.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- LCD (16x2)
- Push buttons (3-4)
- Resistors (10k ohms)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Background Information: An interactive display system allows users to interact with and control the information shown on the screen. By using buttons as input devices, we can create a system where the user can navigate through menus, adjust settings, or view different data sets. This is fundamental in creating user interfaces for embedded systems.

Pre-Lab Questions:

1. How does an LCD display work and how can it be controlled using an STM32 microcontroller?
2. What is the role of pull-down resistors in button circuits?
3. How can multiple buttons be used to interact with an STM32 program?

Procedure:

1. **Setup the Circuit:**

- Connect the LCD to the STM32 as follows:
  - RS to a GPIO pin (e.g., PC0)
  - E to a GPIO pin (e.g., PC1)
  - D4 to a GPIO pin (e.g., PC2)
  - D5 to a GPIO pin (e.g., PC3)
  - D6 to a GPIO pin (e.g., PC4)
  - D7 to a GPIO pin (e.g., PC5)
- Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
- Connect each push button between a digital pin and ground, and use a pull-down resistor (10k ohms) for each button.
- Example button connections:
  - Button 1 to a GPIO pin (e.g., PA0)
  - Button 2 to a GPIO pin (e.g., PA1)
  - Button 3 to a GPIO pin (e.g., PA2)

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the GPIO pins for the buttons and the LCD.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize the LCD library and GPIO inputs for the buttons.
   - Write functions to read the button states.
   - Implement functions to update the LCD based on button presses.
4. **Upload and Monitor:**
   - Compile and upload the code to the STM32 board.
   - Ensure the LCD displays dynamic data and responds to button presses.
5. **Implement Interactivity:**
   - Create a menu system or dynamic display that changes based on user input.
   - Document the functionality and user interaction with the system.

Post-Lab Questions:

1. How did you implement the button reading in the STM32 code?
2. What challenges did you encounter while creating the interactive display and how did you overcome them?

3. How can this concept be applied to more complex user interfaces?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the LCD and buttons to prevent damage to the components and the microcontroller.

Experiment 7a: Binary Counter with LEDs: Implementing an Arduino Binary Counter Using Multiple LEDs for Educational Purposes

---

Experiment Title: Binary Counter with LEDs: Implementing a Binary Counter Using Multiple LEDs for Educational Purposes

Objective: To implement a binary counter using multiple LEDs to visually represent binary numbers.

Description: In this experiment, we will use an Arduino to control multiple LEDs to create a binary counter. Each LED will represent a binary digit (bit), and together they will display numbers in binary format. This experiment demonstrates the concept of binary counting and how digital signals can represent data.

Materials Needed:

- Arduino Uno
- LEDs (4-8)
- Resistors (220 ohms)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Binary counting is fundamental to digital electronics and computer science. Each bit in a binary number can be represented by an LED, where an LED on indicates a binary '1' and an LED off indicates a binary '0'. This project helps in understanding how binary numbers work and how to implement counting in a microcontroller.

Pre-Lab Questions:

1. What is a binary number system and how does it differ from the decimal system?
2. How can an LED be used to represent a binary digit?
3. How does a microcontroller like Arduino handle binary numbers and counting?

Procedure:

1. **Setup the Circuit:**
    - Place the LEDs and resistors on the breadboard.
    - Connect the anode (longer leg) of each LED to different digital pins on the Arduino (e.g., pins 2, 3, 4, 5, 6, 7, 8, 9).
    - Connect the cathode (shorter leg) of each LED to a resistor, and then connect the other end of the resistor to the ground rail on the breadboard.
    - Connect the ground rail to the GND pin on the Arduino.
2. **Program the Arduino:**
    - Write a program to initialize the digital pins as outputs.
    - Implement a counter that increments from 0 and uses bitwise operations to control the LEDs.
    - Use a delay to visually observe the counting.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the LEDs representing the binary count from 0 to the maximum value possible with the given number of LEDs.
4. **Experiment with Counting:**
    - Modify the code to adjust the counting speed.
    - Document the changes and their effects on the LED behavior.

Post-Lab Questions:

1. How did you implement the binary counting logic in Arduino?
2. What challenges did you encounter while programming the LED counter and how did you resolve them?
3. How can this binary counting concept be applied in real-world digital systems?

Experiment 7b: Binary Counter with LEDs: Implementing STM Binary Counter Using Multiple LEDs for Educational Purposes

Objective: To implement a binary counter using multiple LEDs to visually represent binary numbers.

Description: In this experiment, we will use an STM32 microcontroller to control multiple LEDs to create a binary counter. Each LED will represent a binary digit (bit), and together they will display numbers in binary format. This experiment demonstrates the concept of binary counting and how digital signals can represent data.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- LEDs (4-8)
- Resistors (220 ohms)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Binary counting is fundamental to digital electronics and computer science. Each bit in a binary number can be represented by an LED, where an LED on indicates a binary '1' and an LED off indicates a binary '0'. This project helps in understanding how binary numbers work and how to implement counting in a microcontroller.

Pre-Lab Questions:

1. What is a binary number system and how does it differ from the decimal system?
2. How can an LED be used to represent a binary digit?
3. How does a microcontroller like STM32 handle binary numbers and counting?

Procedure:

1. **Setup the Circuit:**
    - Place the LEDs and resistors on the breadboard.
    - Connect the anode (longer leg) of each LED to different GPIO pins on the STM32 board (e.g., pins PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7).

- Connect the cathode (shorter leg) of each LED to a resistor, and then connect the other end of the resistor to the ground rail on the breadboard.
- Connect the ground rail to a GND pin on the STM32 board.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the GPIO pins for output mode.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize the GPIO pins in the main program.
   - Implement a counter that increments from 0 and uses bitwise operations to control the LEDs.
   - Use a delay to visually observe the counting.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Observe the LEDs representing the binary count from 0 to the maximum value possible with the given number of LEDs.
5. **Experiment with Counting:**
   - Modify the code to adjust the counting speed.
   - Document the changes and their effects on the LED behavior.

Post-Lab Questions:

1. How did you implement the binary counting logic in STM32?
2. What challenges did you encounter while programming the LED counter and how did you resolve them?
3. How can this binary counting concept be applied in real-world digital systems?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the LEDs and resistors to prevent damage to the components and the microcontroller.

Experiment 8a: Event-triggered Buzzer: Activating the Arduino Buzzer Based on Specific Events like Button Press or Sensor Triggers

---

Experiment Title: Event-triggered Buzzer: Activating the Buzzer Based on Specific Events like Button Press or Sensor Triggers

Objective: To activate a buzzer based on specific events such as a button press or sensor trigger using an Arduino.

Description: In this experiment, we will use an Arduino to control a buzzer that activates based on certain events. These events can be a button press or a trigger from a sensor. This experiment demonstrates how to use digital inputs and outputs to create an interactive system.

Materials Needed:

- Arduino Uno
- Buzzer (active or passive)
- Push button
- Sensor module (e.g., PIR sensor, temperature sensor)
- Resistors (10k ohms for button, value as needed for sensor)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: A buzzer is a simple device that can produce sound and is commonly used for alarms and notifications. By using digital inputs like buttons and sensors, we can control when the buzzer activates. This project illustrates how to handle input events and generate output signals in an Arduino.

Pre-Lab Questions:

1. What is the difference between an active and a passive buzzer?
2. How can a button press be detected using an Arduino?
3. What types of sensors can be used to trigger events, and how do they work?

Procedure:

1. **Setup the Circuit:**
    - Connect the positive terminal of the buzzer to a digital pin on the Arduino (e.g., pin 8).
    - Connect the negative terminal of the buzzer to the GND pin on the Arduino.
    - Connect one side of the push button to a digital pin on the Arduino (e.g., pin 2).
    - Connect the other side of the push button to the GND pin on the Arduino.
    - Use a pull-up resistor (10k ohms) connected between the button pin and 5V.
    - Connect the sensor to appropriate pins on the Arduino based on the sensor type (e.g., PIR sensor to digital pin 3).
2. **Program the Arduino:**
    - Write a program to initialize the digital pins for the button and the sensor as inputs.
    - Initialize the pin connected to the buzzer as an output.
    - Implement code to check the state of the button and sensor.
    - Activate the buzzer when the button is pressed or the sensor is triggered.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Press the button and observe the buzzer activation.
    - Trigger the sensor and observe the buzzer activation.
4. **Experiment with Events:**
    - Modify the code to handle different types of sensors.
    - Document the changes and their effects on the buzzer behavior.

Post-Lab Questions:

1. How did you detect button presses and sensor triggers in your Arduino code?
2. What challenges did you encounter while programming the event-triggered buzzer and how did you resolve them?
3. How can the concept of event-triggered actions be applied in real-world applications?

Experiment 8b: Event-triggered Buzzer: Activating the STM Buzzer Based on Specific Events like Button Press or Sensor Triggers

Objective: To activate a buzzer based on specific events such as a button press or sensor trigger using an STM32 development board.

Description: In this experiment, we will use an STM32 microcontroller to control a buzzer that activates based on certain events. These events can be a button press or a trigger from a sensor. This experiment demonstrates how to use digital inputs and outputs to create an interactive system.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- Buzzer (active or passive)
- Push button
- Sensor module (e.g., PIR sensor, temperature sensor)
- Resistors (10k ohms for button, value as needed for sensor)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: A buzzer is a simple device that can produce sound and is commonly used for alarms and notifications. By using digital inputs like buttons and sensors, we can control when the buzzer activates. This project illustrates how to handle input events and generate output signals in an STM32 microcontroller.

Pre-Lab Questions:

1. What is the difference between an active and a passive buzzer?
2. How can a button press be detected using an STM32 microcontroller?
3. What types of sensors can be used to trigger events, and how do they work?

Procedure:

1. **Setup the Circuit:**

- Connect the positive terminal of the buzzer to a GPIO pin on the STM32 (e.g., pin PA0).
- Connect the negative terminal of the buzzer to the GND pin on the STM32.
- Connect one side of the push button to a GPIO pin on the STM32 (e.g., pin PA1).
- Connect the other side of the push button to the GND pin on the STM32.
- Use a pull-up resistor (10k ohms) connected between the button pin and 3.3V/5V.
- Connect the sensor to appropriate pins on the STM32 based on the sensor type (e.g., PIR sensor to a GPIO pin).

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the GPIO pins for the button and sensor as inputs.
   - Configure the GPIO pin for the buzzer as an output.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize the GPIO pins in the main program.
   - Implement code to check the state of the button and sensor.
   - Activate the buzzer when the button is pressed or the sensor is triggered.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Press the button and observe the buzzer activation.
   - Trigger the sensor and observe the buzzer activation.
5. **Experiment with Events:**
   - Modify the code to handle different types of sensors.
   - Document the changes and their effects on the buzzer behavior.

Post-Lab Questions:

1. How did you detect button presses and sensor triggers in your STM32 code?
2. What challenges did you encounter while programming the event-triggered buzzer and how did you resolve them?
3. How can the concept of event-triggered actions be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the buzzer, buttons, and sensors to prevent damage to the components and the microcontroller.

Experiment 9a: Tone Generation: Generating Different Tones Using Arduino PWM to Control the Buzzer for Auditory Feedback

---

Experiment Title: Tone Generation: Generating Different Tones Using PWM to Control the Buzzer for Auditory Feedback

Objective: To generate different tones using Pulse Width Modulation (PWM) to control a buzzer and provide auditory feedback with an Arduino.

Description: In this experiment, we will use an Arduino to generate various tones using a buzzer. By controlling the frequency of the PWM signal sent to the buzzer, we can create different sounds. This experiment demonstrates the use of PWM for generating auditory signals.

Materials Needed:

- Arduino Uno
- Buzzer (passive)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: A passive buzzer requires an external signal to produce sound. By using PWM, we can create a square wave signal with varying frequency, which the buzzer converts into sound. This technique is commonly used in alarms, notifications, and musical applications.

Pre-Lab Questions:

1. What is the difference between an active and a passive buzzer?
2. How does PWM control the frequency of a signal?
3. What is the relationship between PWM frequency and the tone produced by a buzzer?

Procedure:

1. **Setup the Circuit:**

    - Connect the positive terminal of the buzzer to a PWM-capable digital pin on the Arduino (e.g., pin 9).
    - Connect the negative terminal of the buzzer to the GND pin on the Arduino.
2. **Program the Arduino:**
    - Use the `tone()` function to generate different frequencies.
    - Write a program to produce a series of tones by changing the frequency of the PWM signal.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the different tones produced by the buzzer.
4. **Experiment with Tones:**
    - Modify the code to produce different sequences of tones.
    - Document the changes and their effects on the sound produced.

Post-Lab Questions:

1. How did you use the `tone()` function to generate different frequencies?
2. What challenges did you encounter while generating different tones and how did you resolve them?
3. How can the concept of tone generation be applied in real-world applications?

Experiment 9b: Tone Generation: Generating Different Tones Using STM PWM to Control the Buzzer for Auditory Feedback

Objective: To generate different tones using Pulse Width Modulation (PWM) to control a buzzer and provide auditory feedback with an STM32 development board.

Description: In this experiment, we will use an STM32 microcontroller to generate various tones using a buzzer. By controlling the frequency of the PWM signal sent to the buzzer, we can create different sounds. This experiment demonstrates the use of PWM for generating auditory signals.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- Buzzer (passive)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: A passive buzzer requires an external signal to produce sound. By using PWM, we can create a square wave signal with varying frequency, which the buzzer converts into sound. This technique is commonly used in alarms, notifications, and musical applications.

Pre-Lab Questions:

1. What is the difference between an active and a passive buzzer?
2. How does PWM control the frequency of a signal?
3. What is the relationship between PWM frequency and the tone produced by a buzzer?

Procedure:

1. **Setup the Circuit:**
    - Connect the positive terminal of the buzzer to a PWM-capable GPIO pin on the STM32 (e.g., pin PA0).
    - Connect the negative terminal of the buzzer to the GND pin on the STM32.
2. **Configure the Microcontroller:**

- Open STM32CubeMX and create a new project.
- Configure the PWM output on a GPIO pin.
- Generate the initialization code and open it in STM32CubeIDE.

3. **Write the Code:**
   - Initialize the PWM signal in the main program.
   - Write code to produce different frequencies by adjusting the PWM parameters.
   - Implement a series of tones by changing the frequency of the PWM signal.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Observe the different tones produced by the buzzer.
5. **Experiment with Tones:**
   - Modify the code to produce different sequences of tones.
   - Document the changes and their effects on the sound produced.

Post-Lab Questions:

1. How did you configure and use PWM to generate different frequencies in STM32?
2. What challenges did you encounter while generating different tones and how did you resolve them?
3. How can the concept of tone generation be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the buzzer to prevent damage to the components and the microcontroller.

Experiment 10a: Voltage Monitoring System: Measuring and Displaying Arduino Voltage Levels on the LCD or via Serial Output

---

Experiment Title: Voltage Monitoring System: Measuring and Displaying Voltage Levels on the LCD or via Serial Output

Objective: To measure and display voltage levels using an Arduino, either on an LCD or via serial output.

Description: In this experiment, we will use an Arduino to measure voltage levels and display them on an LCD or send them to a computer via serial output. This involves using the Arduino's analog-to-digital converter (ADC) to read voltage levels and convert them to a readable format.

Materials Needed:

- Arduino Uno
- LCD (16x2) or Serial Monitor
- Voltage divider (resistors)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: Voltage monitoring is crucial in many applications to ensure proper operation of electrical systems. The Arduino's ADC can read analog voltage levels and convert them into digital values that can be displayed on an LCD or transmitted to a computer for monitoring and logging purposes.

Pre-Lab Questions:

1. How does an analog-to-digital converter (ADC) work?
2. What is a voltage divider and how can it be used to measure higher voltages?
3. How can the Arduino's `analogRead()` function be used to measure voltage?

Procedure:

1. **Setup the Circuit:**
    - If using an LCD:
        - Connect the LCD to the Arduino as follows:
            - RS to digital pin 12
            - E to digital pin 11
            - D4 to digital pin 5
            - D5 to digital pin 4
            - D6 to digital pin 3
            - D7 to digital pin 2
        - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
    - If using a voltage divider:
        - Connect two resistors in series between the voltage to be measured and ground.
        - Connect the junction of the resistors to an analog input pin on the Arduino (e.g., A0).
    - Connect the ground rail of the breadboard to the GND pin on the Arduino.
2. **Program the Arduino:**
    - Use the `analogRead()` function to read the voltage level at the analog input pin.
    - If using an LCD, use the LiquidCrystal library to display the voltage on the LCD.
    - If using serial output, use the `Serial.begin()` and `Serial.print()` functions to send the voltage readings to the Serial Monitor.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the voltage readings on the LCD or the Serial Monitor.
4. **Calibrate and Experiment:**
    - Calibrate the voltage readings by adjusting the resistor values in the voltage divider.
    - Document the changes and their effects on the voltage readings.

Post-Lab Questions:

1. How did you use the `analogRead()` function to measure voltage levels?

2. What challenges did you encounter while setting up the voltage divider and how did you resolve them?
3. How can voltage monitoring be applied in real-world applications?

Experiment 10b: Voltage Monitoring System: Measuring and Displaying STM Voltage Levels on the LCD or via Serial Output

Objective: To measure and display voltage levels using an STM32 microcontroller, either on an LCD or via serial output.

Description: In this experiment, we will use an STM32 microcontroller to measure voltage levels and display them on an LCD or send them to a computer via serial output. This involves using the STM32's analog-to-digital converter (ADC) to read voltage levels and convert them to a readable format.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- LCD (16x2) or Serial Monitor
- Voltage divider (resistors)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: Voltage monitoring is crucial in many applications to ensure proper operation of electrical systems. The STM32's ADC can read analog voltage levels and convert them into digital values that can be displayed on an LCD or transmitted to a computer for monitoring and logging purposes.

Pre-Lab Questions:

1. How does an analog-to-digital converter (ADC) work in STM32?
2. What is a voltage divider and how can it be used to measure higher voltages?
3. How can the STM32's `HAL_ADC_GetValue()` function be used to measure voltage?

Procedure:

1. **Setup the Circuit:**
    - If using an LCD:
        - Connect the LCD to the STM32 as follows:

        - RS to a GPIO pin (e.g., PC0)
        - E to a GPIO pin (e.g., PC1)
        - D4 to a GPIO pin (e.g., PC2)
        - D5 to a GPIO pin (e.g., PC3)
        - D6 to a GPIO pin (e.g., PC4)
        - D7 to a GPIO pin (e.g., PC5)
      - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
    - If using a voltage divider:
      - Connect two resistors in series between the voltage to be measured and ground.
      - Connect the junction of the resistors to an analog input pin on the STM32 (e.g., PA0).
    - Connect the ground rail of the breadboard to the GND pin on the STM32.
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the ADC peripheral and the GPIO pins for the LCD.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize the ADC and read the voltage level using `HAL_ADC_GetValue()`.
    - If using an LCD, use the LCD library to display the voltage on the LCD.
    - If using serial output, use `HAL_UART_Transmit()` to send the voltage readings to a serial terminal.
4. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Observe the voltage readings on the LCD or the Serial Monitor.
5. **Calibrate and Experiment:**
    - Calibrate the voltage readings by adjusting the resistor values in the voltage divider.
    - Document the changes and their effects on the voltage readings.

Post-Lab Questions:

1. How did you use the `HAL_ADC_GetValue()` function to measure voltage levels?

2. What challenges did you encounter while setting up the voltage divider and how did you resolve them?
3. How can voltage monitoring be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the voltage divider and LCD to prevent damage to the components and the microcontroller.

Experiment 11a: Threshold-based Voltage Alert: Triggering an Arduino Alert (LED/Buzzer) When Voltage Crosses a Predefined Threshold

---

Experiment Title: Threshold-based Voltage Alert: Triggering an Alert (LED/Buzzer) When Voltage Crosses a Predefined Threshold

Objective: To trigger an alert using an LED or buzzer when the voltage crosses a predefined threshold using an Arduino.

Description: In this experiment, we will use an Arduino to monitor voltage levels and trigger an alert when the voltage exceeds or falls below a predefined threshold. The alert will be indicated using an LED or a buzzer. This demonstrates how to implement threshold-based monitoring and alert systems.

Materials Needed:

- Arduino Uno
- LED and/or Buzzer
- Voltage divider (resistors)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Threshold-based alert systems are used in various applications to monitor critical parameters such as voltage, temperature, and pressure. When the monitored parameter crosses a predefined threshold, an alert is triggered to notify the user. This experiment uses the Arduino's analog-to-digital converter (ADC) to read voltage levels and control output devices like LEDs or buzzers.

Pre-Lab Questions:

1. What is a voltage threshold and how is it used in monitoring systems?
2. How can an LED or buzzer be controlled using an Arduino?
3. How can the Arduino's `analogRead()` function be used to monitor voltage levels?

Procedure:

1. **Setup the Circuit:**
    - Connect the voltage divider to an analog input pin on the Arduino (e.g., A0).
    - Connect the positive terminal of the LED or buzzer to a digital pin on the Arduino (e.g., pin 9).
    - Connect the negative terminal of the LED or buzzer to the GND pin on the Arduino.
    - If using an LED, include a current-limiting resistor (220 ohms) in series with the LED.
    - Connect the ground rail of the breadboard to the GND pin on the Arduino.
2. **Program the Arduino:**
    - Write a program to read the voltage level at the analog input pin using `analogRead()`.
    - Implement logic to compare the read voltage against a predefined threshold.
    - Control the LED or buzzer based on whether the voltage exceeds or falls below the threshold.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Test the system by varying the input voltage and observing the behavior of the LED or buzzer.
4. **Calibrate and Experiment:**
    - Adjust the voltage threshold in the code and observe the changes.
    - Document the changes and their effects on the alert behavior.

Post-Lab Questions:

1. How did you use the `analogRead()` function to monitor voltage levels?
2. What challenges did you encounter while setting up the threshold-based alert system and how did you resolve them?
3. How can threshold-based monitoring be applied in real-world applications?

Experiment 11b: Threshold-based Voltage Alert: Triggering an STM Alert (LED/Buzzer) When Voltage Crosses a Predefined Threshold

Objective: To trigger an alert using an LED or buzzer when the voltage crosses a predefined threshold using an STM32 microcontroller.

Description: In this experiment, we will use an STM32 microcontroller to monitor voltage levels and trigger an alert when the voltage exceeds or falls below a predefined threshold. The alert will be indicated using an LED or a buzzer. This demonstrates how to implement threshold-based monitoring and alert systems.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- LED and/or Buzzer
- Voltage divider (resistors)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Threshold-based alert systems are used in various applications to monitor critical parameters such as voltage, temperature, and pressure. When the monitored parameter crosses a predefined threshold, an alert is triggered to notify the user. This experiment uses the STM32's analog-to-digital converter (ADC) to read voltage levels and control output devices like LEDs or buzzers.

Pre-Lab Questions:

1. What is a voltage threshold and how is it used in monitoring systems?
2. How can an LED or buzzer be controlled using an STM32 microcontroller?
3. How can the STM32's ADC be used to monitor voltage levels?

Procedure:

1. **Setup the Circuit:**
    - Connect the voltage divider to an analog input pin on the STM32 (e.g., PA0).

    - Connect the positive terminal of the LED or buzzer to a GPIO pin on the STM32 (e.g., PA1).
    - Connect the negative terminal of the LED or buzzer to the GND pin on the STM32.
    - If using an LED, include a current-limiting resistor (220 ohms) in series with the LED.
    - Connect the ground rail of the breadboard to the GND pin on the STM32.
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the ADC peripheral and the GPIO pins for the LED or buzzer.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize the ADC and read the voltage level using `HAL_ADC_GetValue()`.
    - Implement logic to compare the read voltage against a predefined threshold.
    - Control the LED or buzzer based on whether the voltage exceeds or falls below the threshold.
4. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Test the system by varying the input voltage and observing the behavior of the LED or buzzer.
5. **Calibrate and Experiment:**
    - Adjust the voltage threshold in the code and observe the changes.
    - Document the changes and their effects on the alert behavior.

Post-Lab Questions:

1. How did you use the `HAL_ADC_GetValue()` function to monitor voltage levels?
2. What challenges did you encounter while setting up the threshold-based alert system and how did you resolve them?
3. How can threshold-based monitoring be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the voltage divider and alert devices to prevent damage to the components and the microcontroller.

Experiment 12a: Current Monitoring System: Measuring and Displaying Arduino Current Values in a Circuit

---

Experiment Title: Current Monitoring System: Measuring and Displaying Current Values in a Circuit

Objective: To measure and display current values in a circuit using an Arduino.

Description: In this experiment, we will use an Arduino to measure the current flowing through a circuit and display the values on an LCD or via serial output. This involves using a current sensor to convert the current to a measurable voltage that the Arduino can read.

Materials Needed:

- Arduino Uno
- Current sensor (e.g., ACS712)
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Current monitoring is crucial in many applications to ensure that electrical devices operate within safe limits. The ACS712 current sensor is a popular choice for measuring current because it provides an analog output voltage proportional to the current. The Arduino's analog-to-digital converter (ADC) can then read this voltage and calculate the current.

Pre-Lab Questions:

1. How does a current sensor like the ACS712 work?
2. What is the relationship between the sensor output voltage and the measured current?
3. How can the Arduino's `analogRead()` function be used to measure the sensor's output?

Procedure:

1. **Setup the Circuit:**

- Connect the VCC and GND pins of the ACS712 to the 5V and GND pins on the Arduino.
- Connect the OUT pin of the ACS712 to an analog input pin on the Arduino (e.g., A0).
- Connect the circuit to be measured in series with the ACS712's input terminals.
- If using an LCD:
  - Connect the LCD to the Arduino as follows:
    - RS to digital pin 12
    - E to digital pin 11
    - D4 to digital pin 5
    - D5 to digital pin 4
    - D6 to digital pin 3
    - D7 to digital pin 2
  - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.

2. **Program the Arduino:**
   - Use the `analogRead()` function to read the voltage level at the analog input pin.
   - Convert the voltage reading to current using the sensor's transfer function.
   - If using an LCD, use the LiquidCrystal library to display the current on the LCD.
   - If using serial output, use the `Serial.begin()` and `Serial.print()` functions to send the current readings to the Serial Monitor.
3. **Upload and Test:**
   - Upload the code to the Arduino.
   - Observe the current readings on the LCD or the Serial Monitor.
4. **Calibrate and Experiment:**
   - Calibrate the current readings by adjusting the sensor's transfer function in the code.
   - Document the changes and their effects on the current readings.

Post-Lab Questions:

1. How did you use the `analogRead()` function to measure current values?
2. What challenges did you encounter while setting up the current monitoring system and how did you resolve them?

3. How can current monitoring be applied in real-world applications?

Experiment 12b: Current Monitoring System: Measuring and Displaying STM Current Values in a Circuit

Objective: To measure and display current values in a circuit using an STM32 microcontroller.

Description: In this experiment, we will use an STM32 microcontroller to measure the current flowing through a circuit and display the values on an LCD or via serial output. This involves using a current sensor to convert the current to a measurable voltage that the STM32 can read.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- Current sensor (e.g., ACS712)
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Current monitoring is crucial in many applications to ensure that electrical devices operate within safe limits. The ACS712 current sensor is a popular choice for measuring current because it provides an analog output voltage proportional to the current. The STM32's analog-to-digital converter (ADC) can then read this voltage and calculate the current.

Pre-Lab Questions:

1. How does a current sensor like the ACS712 work?
2. What is the relationship between the sensor output voltage and the measured current?
3. How can the STM32's ADC be used to measure the sensor's output?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the ACS712 to the 5V and GND pins on the STM32.
    - Connect the OUT pin of the ACS712 to an analog input pin on the STM32 (e.g., PA0).
    - Connect the circuit to be measured in series with the ACS712's input terminals.

- If using an LCD:
  - Connect the LCD to the STM32 as follows:
    - RS to a GPIO pin (e.g., PC0)
    - E to a GPIO pin (e.g., PC1)
    - D4 to a GPIO pin (e.g., PC2)
    - D5 to a GPIO pin (e.g., PC3)
    - D6 to a GPIO pin (e.g., PC4)
    - D7 to a GPIO pin (e.g., PC5)
  - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the ADC peripheral and the GPIO pins for the LCD.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize the ADC and read the voltage level using `HAL_ADC_GetValue()`.
   - Convert the voltage reading to current using the sensor's transfer function.
   - If using an LCD, use the LCD library to display the current on the LCD.
   - If using serial output, use `HAL_UART_Transmit()` to send the current readings to a serial terminal.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Observe the current readings on the LCD or the Serial Monitor.
5. **Calibrate and Experiment:**
   - Calibrate the current readings by adjusting the sensor's transfer function in the code.
   - Document the changes and their effects on the current readings.

Post-Lab Questions:

1. How did you use the `HAL_ADC_GetValue()` function to measure current values?
2. What challenges did you encounter while setting up the current monitoring system and how did you resolve them?
3. How can current monitoring be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the current sensor and LCD to prevent damage to the components and the microcontroller.

Experiment 13a: Overcurrent Alert System: Activating an Arduino Alert if the Current Exceeds a Set Limit for Safety

---

Experiment Title: Overcurrent Alert System: Activating an Alert if the Current Exceeds a Set Limit for Safety

Objective: To monitor current levels in a circuit and activate an alert (LED/Buzzer) if the current exceeds a predefined safety limit using an Arduino.

Description: In this experiment, we will use an Arduino to monitor the current flowing through a circuit and activate an alert when the current exceeds a set limit. This is achieved using a current sensor and a comparator mechanism to ensure the safety of electrical devices by preventing overcurrent conditions.

Materials Needed:

- Arduino Uno
- Current sensor (e.g., ACS712)
- LED and/or Buzzer
- Resistors (220 ohms for LED)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Overcurrent conditions can damage electrical components and create safety hazards. By monitoring current levels and triggering an alert when the current exceeds a safe threshold, we can protect devices and enhance system reliability. The ACS712 current sensor outputs a voltage proportional to the current, which the Arduino can read and compare against a set limit.

Pre-Lab Questions:

1. What is an overcurrent condition and why is it important to monitor?
2. How does the ACS712 current sensor work?
3. How can an Arduino be used to compare current levels and activate alerts?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the ACS712 to the 5V and GND pins on the Arduino.
    - Connect the OUT pin of the ACS712 to an analog input pin on the Arduino (e.g., A0).
    - Connect the circuit to be measured in series with the ACS712's input terminals.
    - Connect the positive terminal of the LED or buzzer to a digital pin on the Arduino (e.g., pin 9).
    - Connect the negative terminal of the LED or buzzer to the GND pin on the Arduino.
    - If using an LED, include a current-limiting resistor (220 ohms) in series with the LED.
    - Connect the ground rail of the breadboard to the GND pin on the Arduino.
2. **Program the Arduino:**
    - Use the `analogRead()` function to read the voltage level at the analog input pin.
    - Convert the voltage reading to current using the sensor's transfer function.
    - Implement logic to compare the measured current against a predefined limit.
    - Control the LED or buzzer based on whether the current exceeds the limit.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Test the system by varying the input current and observing the behavior of the LED or buzzer.
4. **Calibrate and Experiment:**
    - Adjust the current limit in the code and observe the changes.
    - Document the changes and their effects on the alert behavior.

Post-Lab Questions:

1. How did you use the `analogRead()` function to monitor current levels?
2. What challenges did you encounter while setting up the overcurrent alert system and how did you resolve them?
3. How can overcurrent monitoring be applied in real-world applications?

Experiment 13b: Overcurrent Alert System: Activating an STM Alert if the Current Exceeds a Set Limit for Safety

Objective: To monitor current levels in a circuit and activate an alert (LED/Buzzer) if the current exceeds a predefined safety limit using an STM32 microcontroller.

Description: In this experiment, we will use an STM32 microcontroller to monitor the current flowing through a circuit and activate an alert when the current exceeds a set limit. This is achieved using a current sensor and a comparator mechanism to ensure the safety of electrical devices by preventing overcurrent conditions.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- Current sensor (e.g., ACS712)
- LED and/or Buzzer
- Resistors (220 ohms for LED)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: Overcurrent conditions can damage electrical components and create safety hazards. By monitoring current levels and triggering an alert when the current exceeds a safe threshold, we can protect devices and enhance system reliability. The ACS712 current sensor outputs a voltage proportional to the current, which the STM32 can read and compare against a set limit.

Pre-Lab Questions:

1. What is an overcurrent condition and why is it important to monitor?
2. How does the ACS712 current sensor work?
3. How can an STM32 be used to compare current levels and activate alerts?

Procedure:

1. **Setup the Circuit:**

- Connect the VCC and GND pins of the ACS712 to the 5V and GND pins on the STM32.
- Connect the OUT pin of the ACS712 to an analog input pin on the STM32 (e.g., PA0).
- Connect the circuit to be measured in series with the ACS712's input terminals.
- Connect the positive terminal of the LED or buzzer to a GPIO pin on the STM32 (e.g., PA1).
- Connect the negative terminal of the LED or buzzer to the GND pin on the STM32.
- If using an LED, include a current-limiting resistor (220 ohms) in series with the LED.
- Connect the ground rail of the breadboard to the GND pin on the STM32.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the ADC peripheral and the GPIO pins for the LED or buzzer.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize the ADC and read the voltage level using `HAL_ADC_GetValue()`.
   - Convert the voltage reading to current using the sensor's transfer function.
   - Implement logic to compare the measured current against a predefined limit.
   - Control the LED or buzzer based on whether the current exceeds the limit.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Test the system by varying the input current and observing the behavior of the LED or buzzer.
5. **Calibrate and Experiment:**
   - Adjust the current limit in the code and observe the changes.
   - Document the changes and their effects on the alert behavior.

Post-Lab Questions:

1. How did you use the `HAL_ADC_GetValue()` function to monitor current levels?
2. What challenges did you encounter while setting up the overcurrent alert system and how did you resolve them?

3. How can overcurrent monitoring be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the current sensor and alert devices to prevent damage to the components and the microcontroller.

Experiment 14a: Arduino Temperature and Pressure Monitoring: Measuring and Displaying Temperature and Pressure Data for Environmental Monitoring

---

Experiment Title: Temperature and Pressure Monitoring: Measuring and Displaying Temperature and Pressure Data for Environmental Monitoring

Objective: To measure and display temperature and pressure data using an Arduino for environmental monitoring.

Description: In this experiment, we will use an Arduino to measure temperature and pressure using sensors such as the BMP280 or similar. The measured data will be displayed on an LCD or sent to a computer via serial output. This project demonstrates how to integrate multiple sensors and display their data for monitoring environmental conditions.

Materials Needed:

- Arduino Uno
- BMP280 sensor module (or similar)
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: Environmental monitoring involves measuring various parameters such as temperature and pressure to understand and manage environmental conditions. The BMP280 sensor is a digital sensor that provides precise measurements of both temperature and pressure, making it suitable for weather stations, altimeters, and other monitoring applications.

Pre-Lab Questions:

1. How does the BMP280 sensor measure temperature and pressure?
2. What are the typical applications of temperature and pressure monitoring?
3. How can the Arduino's `Wire` library be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the BMP280 to the 3.3V/5V and GND pins on the Arduino.
    - Connect the SDA (data) pin of the BMP280 to the A4 pin on the Arduino.
    - Connect the SCL (clock) pin of the BMP280 to the A5 pin on the Arduino.
    - If using an LCD:
        - Connect the LCD to the Arduino as follows:
            - RS to digital pin 12
            - E to digital pin 11
            - D4 to digital pin 5
            - D5 to digital pin 4
            - D6 to digital pin 3
            - D7 to digital pin 2
        - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Program the Arduino:**
    - Install the Adafruit BMP280 library from the Arduino Library Manager.
    - Use the `Wire` library to initialize I2C communication with the BMP280 sensor.
    - Write code to read temperature and pressure data from the BMP280.
    - If using an LCD, use the LiquidCrystal library to display the data.
    - If using serial output, use the `Serial.begin()` and `Serial.print()` functions to send the data to the Serial Monitor.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the temperature and pressure readings on the LCD or the Serial Monitor.
4. **Calibrate and Experiment:**
    - Verify the accuracy of the sensor readings by comparing them with known values.
    - Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the `Wire` library to communicate with the BMP280 sensor?
2. What challenges did you encounter while setting up the temperature and pressure monitoring system and how did you resolve them?
3. How can temperature and pressure monitoring be applied in real-world applications?

Experiment 14b: Temperature and Pressure Monitoring: Measuring and Displaying Temperature and Pressure Data for Environmental Monitoring

Objective: To measure and display temperature and pressure data using an STM32 microcontroller for environmental monitoring.

Description: In this experiment, we will use an STM32 microcontroller to measure temperature and pressure using sensors such as the BMP280 or similar. The measured data will be displayed on an LCD or sent to a computer via serial output. This project demonstrates how to integrate multiple sensors and display their data for monitoring environmental conditions.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- BMP280 sensor module (or similar)
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: Environmental monitoring involves measuring various parameters such as temperature and pressure to understand and manage environmental conditions. The BMP280 sensor is a digital sensor that provides precise measurements of both temperature and pressure, making it suitable for weather stations, altimeters, and other monitoring applications.

Pre-Lab Questions:

1. How does the BMP280 sensor measure temperature and pressure?
2. What are the typical applications of temperature and pressure monitoring?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the BMP280 to the 3.3V/5V and GND pins on the STM32.

- Connect the SDA (data) pin of the BMP280 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
- Connect the SCL (clock) pin of the BMP280 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
- If using an LCD:
  - Connect the LCD to the STM32 as follows:
    - RS to a GPIO pin (e.g., PC0)
    - E to a GPIO pin (e.g., PC1)
    - D4 to a GPIO pin (e.g., PC2)
    - D5 to a GPIO pin (e.g., PC3)
    - D6 to a GPIO pin (e.g., PC4)
    - D7 to a GPIO pin (e.g., PC5)
  - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the I2C peripheral and the GPIO pins for the LCD.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize I2C communication with the BMP280 sensor using the HAL library.
   - Write code to read temperature and pressure data from the BMP280.
   - If using an LCD, use the LCD library to display the data.
   - If using serial output, use `HAL_UART_Transmit()` to send the data to a serial terminal.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Observe the temperature and pressure readings on the LCD or the Serial Monitor.
5. **Calibrate and Experiment:**
   - Verify the accuracy of the sensor readings by comparing them with known values.
   - Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the BMP280 sensor?
2. What challenges did you encounter while setting up the temperature and pressure monitoring system and how did you resolve them?
3. How can temperature and pressure monitoring be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors and LCD to prevent damage to the components and the microcontroller.

Experiment 15a: Arduino Weather Station: Creating a Basic Weather Station Using BMP280 for Educational Purposes

---

Experiment Title: Weather Station: Creating a Basic Weather Station Using BMP280 for Educational Purposes

Objective: To create a basic weather station that measures and displays temperature, pressure, and altitude using an Arduino and the BMP280 sensor.

Description: In this experiment, we will build a simple weather station using an Arduino and a BMP280 sensor. The station will measure temperature, pressure, and altitude and display the data on an LCD or via serial output. This project demonstrates how to use sensor data to monitor environmental conditions.

Materials Needed:

- Arduino Uno
- BMP280 sensor module
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: A weather station collects data related to the weather and environment. The BMP280 sensor is a digital barometric pressure sensor that can measure temperature, pressure, and altitude. Integrating this sensor with an Arduino allows us to build a basic weather station that can provide real-time data for educational and hobbyist purposes.

Pre-Lab Questions:

1. How does the BMP280 sensor measure temperature, pressure, and altitude?
2. What are the practical applications of a weather station?
3. How can the Arduino's `Wire` library be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the BMP280 to the 3.3V/5V and GND pins on the Arduino.
    - Connect the SDA (data) pin of the BMP280 to the A4 pin on the Arduino.
    - Connect the SCL (clock) pin of the BMP280 to the A5 pin on the Arduino.
    - If using an LCD:
        - Connect the LCD to the Arduino as follows:
            - RS to digital pin 12
            - E to digital pin 11
            - D4 to digital pin 5
            - D5 to digital pin 4
            - D6 to digital pin 3
            - D7 to digital pin 2
        - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Program the Arduino:**
    - Install the Adafruit BMP280 library from the Arduino Library Manager.
    - Use the `Wire` library to initialize I2C communication with the BMP280 sensor.
    - Write code to read temperature, pressure, and altitude data from the BMP280.
    - If using an LCD, use the LiquidCrystal library to display the data.
    - If using serial output, use the `Serial.begin()` and `Serial.print()` functions to send the data to the Serial Monitor.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the temperature, pressure, and altitude readings on the LCD or the Serial Monitor.
4. **Calibrate and Experiment:**
    - Verify the accuracy of the sensor readings by comparing them with known values.
    - Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the `Wire` library to communicate with the BMP280 sensor?
2. What challenges did you encounter while setting up the weather station and how did you resolve them?
3. How can the weather station be enhanced to include additional sensors and features?

Experiment 15b: STM Weather Station: Creating a Basic Weather Station Using BMP280 for Educational Purposes

Objective: To create a basic weather station that measures and displays temperature, pressure, and altitude using an STM32 microcontroller and the BMP280 sensor.

Description: In this experiment, we will build a simple weather station using an STM32 microcontroller and a BMP280 sensor. The station will measure temperature, pressure, and altitude and display the data on an LCD or via serial output. This project demonstrates how to use sensor data to monitor environmental conditions.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- BMP280 sensor module
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: A weather station collects data related to the weather and environment. The BMP280 sensor is a digital barometric pressure sensor that can measure temperature, pressure, and altitude. Integrating this sensor with an STM32 microcontroller allows us to build a basic weather station that can provide real-time data for educational and hobbyist purposes.

Pre-Lab Questions:

1. How does the BMP280 sensor measure temperature, pressure, and altitude?
2. What are the practical applications of a weather station?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**

- Connect the VCC and GND pins of the BMP280 to the 3.3V/5V and GND pins on the STM32.
- Connect the SDA (data) pin of the BMP280 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
- Connect the SCL (clock) pin of the BMP280 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
- If using an LCD:
  - Connect the LCD to the STM32 as follows:
    - RS to a GPIO pin (e.g., PC0)
    - E to a GPIO pin (e.g., PC1)
    - D4 to a GPIO pin (e.g., PC2)
    - D5 to a GPIO pin (e.g., PC3)
    - D6 to a GPIO pin (e.g., PC4)
    - D7 to a GPIO pin (e.g., PC5)
  - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the I2C peripheral and the GPIO pins for the LCD.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize I2C communication with the BMP280 sensor using the HAL library.
   - Write code to read temperature, pressure, and altitude data from the BMP280.
   - If using an LCD, use the LCD library to display the data.
   - If using serial output, use `HAL_UART_Transmit()` to send the data to a serial terminal.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Observe the temperature, pressure, and altitude readings on the LCD or the Serial Monitor.
5. **Calibrate and Experiment:**
   - Verify the accuracy of the sensor readings by comparing them with known values.

- Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the BMP280 sensor?
2. What challenges did you encounter while setting up the weather station and how did you resolve them?
3. How can the weather station be enhanced to include additional sensors and features?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors and LCD to prevent damage to the components and the microcontroller.

Experiment 16a: Arduino Motion Detection System: Detecting and Responding to Motion or Changes in Orientation Using the MPU6050

Experiment Title: Motion Detection System: Detecting and Responding to Motion or Changes in Orientation Using the MPU6050

Objective: To detect and respond to motion or changes in orientation using the MPU6050 sensor and an Arduino.

Description: In this experiment, we will use an Arduino and the MPU6050 sensor to detect motion and changes in orientation. The MPU6050 is a 6-axis motion tracking device that includes a 3-axis accelerometer and a 3-axis gyroscope. This project demonstrates how to read data from the MPU6050 and use it to trigger responses such as activating an LED or buzzer.

Materials Needed:

- Arduino Uno
- MPU6050 sensor module
- LED and/or Buzzer
- Resistors (220 ohms for LED)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: The MPU6050 sensor provides data on acceleration and angular velocity, which can be used to detect motion and orientation changes. By integrating this sensor with an Arduino, we can create systems that respond to physical movements, which are useful in applications such as gesture control, gaming, and safety systems.

Pre-Lab Questions:

1. How does the MPU6050 sensor measure acceleration and angular velocity?
2. What are some practical applications of motion detection systems?
3. How can the Arduino's `Wire` library be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the MPU6050 to the 3.3V/5V and GND pins on the Arduino.
    - Connect the SDA (data) pin of the MPU6050 to the A4 pin on the Arduino.
    - Connect the SCL (clock) pin of the MPU6050 to the A5 pin on the Arduino.
    - Connect the positive terminal of the LED or buzzer to a digital pin on the Arduino (e.g., pin 9).
    - Connect the negative terminal of the LED or buzzer to the GND pin on the Arduino.
    - If using an LED, include a current-limiting resistor (220 ohms) in series with the LED.
    - Connect the ground rail of the breadboard to the GND pin on the Arduino.
2. **Program the Arduino:**
    - Install the MPU6050 library from the Arduino Library Manager.
    - Use the `Wire` library to initialize I2C communication with the MPU6050 sensor.
    - Write code to read acceleration and gyroscope data from the MPU6050.
    - Implement logic to detect significant motion or orientation changes.
    - Control the LED or buzzer based on detected motion or orientation changes.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Test the system by moving or changing the orientation of the MPU6050 and observing the response of the LED or buzzer.
4. **Calibrate and Experiment:**
    - Adjust the sensitivity of motion detection in the code and observe the changes.
    - Document any adjustments made and their effects on the system's responsiveness.

Post-Lab Questions:

1. How did you use the `Wire` library to communicate with the MPU6050 sensor?
2. What challenges did you encounter while setting up the motion detection system and how did you resolve them?
3. How can motion detection systems be applied in real-world applications?

Experiment 16b: STM Motion Detection System: Detecting and Responding to Motion or Changes in Orientation Using the MPU6050

Objective: To detect and respond to motion or changes in orientation using the MPU6050 sensor and an STM32 microcontroller.

Description: In this experiment, we will use an STM32 microcontroller and the MPU6050 sensor to detect motion and changes in orientation. The MPU6050 is a 6-axis motion tracking device that includes a 3-axis accelerometer and a 3-axis gyroscope. This project demonstrates how to read data from the MPU6050 and use it to trigger responses such as activating an LED or buzzer.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- MPU6050 sensor module
- LED and/or Buzzer
- Resistors (220 ohms for LED)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: The MPU6050 sensor provides data on acceleration and angular velocity, which can be used to detect motion and orientation changes. By integrating this sensor with an STM32 microcontroller, we can create systems that respond to physical movements, which are useful in applications such as gesture control, gaming, and safety systems.

Pre-Lab Questions:

1. How does the MPU6050 sensor measure acceleration and angular velocity?
2. What are some practical applications of motion detection systems?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**

- Connect the VCC and GND pins of the MPU6050 to the 3.3V/5V and GND pins on the STM32.
- Connect the SDA (data) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
- Connect the SCL (clock) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
- Connect the positive terminal of the LED or buzzer to a GPIO pin on the STM32 (e.g., PA0).
- Connect the negative terminal of the LED or buzzer to the GND pin on the STM32.
- If using an LED, include a current-limiting resistor (220 ohms) in series with the LED.
- Connect the ground rail of the breadboard to the GND pin on the STM32.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the I2C peripheral and the GPIO pins for the LED or buzzer.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize I2C communication with the MPU6050 sensor using the HAL library.
   - Write code to read acceleration and gyroscope data from the MPU6050.
   - Implement logic to detect significant motion or orientation changes.
   - Control the LED or buzzer based on detected motion or orientation changes.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Test the system by moving or changing the orientation of the MPU6050 and observing the response of the LED or buzzer.
5. **Calibrate and Experiment:**
   - Adjust the sensitivity of motion detection in the code and observe the changes.
   - Document any adjustments made and their effects on the system's responsiveness.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the MPU6050 sensor?

2. What challenges did you encounter while setting up the motion detection system and how did you resolve them?
3. How can motion detection systems be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors and alert devices to prevent damage to the components and the microcontroller.

Experiment 17a: Arduino Tilt Sensing System: Implementing Tilt Sensing and Displaying the Angle of Tilt for Various Applications

---

Experiment Title: Tilt Sensing System: Implementing Tilt Sensing and Displaying the Angle of Tilt for Various Applications

Objective: To measure and display the angle of tilt using an Arduino and an accelerometer (e.g., MPU6050) for various applications.

Description: In this experiment, we will use an Arduino and an accelerometer (MPU6050) to measure the angle of tilt. The angle of tilt will be calculated from the accelerometer data and displayed on an LCD or via serial output. This project demonstrates how to use sensor data to determine orientation and angle, which can be applied in various applications such as robotics, gaming, and device orientation detection.

Materials Needed:

- Arduino Uno
- MPU6050 sensor module
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: Tilt sensing is the process of determining the orientation of an object relative to the ground. Accelerometers can measure acceleration due to gravity, which can be used to calculate the angle of tilt. The MPU6050 sensor provides data on acceleration along three axes, which can be used to determine the tilt angle.

Pre-Lab Questions:

1. How does the MPU6050 sensor measure acceleration?
2. What is the principle behind calculating the angle of tilt using accelerometer data?
3. How can the Arduino's `Wire` library be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the MPU6050 to the 3.3V/5V and GND pins on the Arduino.
    - Connect the SDA (data) pin of the MPU6050 to the A4 pin on the Arduino.
    - Connect the SCL (clock) pin of the MPU6050 to the A5 pin on the Arduino.
    - If using an LCD:
        - Connect the LCD to the Arduino as follows:
            - RS to digital pin 12
            - E to digital pin 11
            - D4 to digital pin 5
            - D5 to digital pin 4
            - D6 to digital pin 3
            - D7 to digital pin 2
        - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Program the Arduino:**
    - Install the MPU6050 library from the Arduino Library Manager.
    - Use the `Wire` library to initialize I2C communication with the MPU6050 sensor.
    - Write code to read acceleration data from the MPU6050.
    - Calculate the angle of tilt using the acceleration data.
    - If using an LCD, use the LiquidCrystal library to display the tilt angle.
    - If using serial output, use the `Serial.begin()` and `Serial.print()` functions to send the tilt angle data to the Serial Monitor.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the tilt angle readings on the LCD or the Serial Monitor.
4. **Calibrate and Experiment:**
    - Verify the accuracy of the tilt angle readings by comparing them with known angles.
    - Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the `Wire` library to communicate with the MPU6050 sensor?
2. What challenges did you encounter while setting up the tilt sensing system and how did you resolve them?
3. How can tilt sensing be applied in real-world applications?

Experiment 17b: STM Tilt Sensing System: Implementing Tilt Sensing and Displaying the Angle of Tilt for Various Applications

Objective: To measure and display the angle of tilt using an STM32 microcontroller and an accelerometer (e.g., MPU6050) for various applications.

Description: In this experiment, we will use an STM32 microcontroller and an accelerometer (MPU6050) to measure the angle of tilt. The angle of tilt will be calculated from the accelerometer data and displayed on an LCD or via serial output. This project demonstrates how to use sensor data to determine orientation and angle, which can be applied in various applications such as robotics, gaming, and device orientation detection.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- MPU6050 sensor module
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: Tilt sensing is the process of determining the orientation of an object relative to the ground. Accelerometers can measure acceleration due to gravity, which can be used to calculate the angle of tilt. The MPU6050 sensor provides data on acceleration along three axes, which can be used to determine the tilt angle.

Pre-Lab Questions:

1. How does the MPU6050 sensor measure acceleration?
2. What is the principle behind calculating the angle of tilt using accelerometer data?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**

- Connect the VCC and GND pins of the MPU6050 to the 3.3V/5V and GND pins on the STM32.
- Connect the SDA (data) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
- Connect the SCL (clock) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
- If using an LCD:
  - Connect the LCD to the STM32 as follows:
    - RS to a GPIO pin (e.g., PC0)
    - E to a GPIO pin (e.g., PC1)
    - D4 to a GPIO pin (e.g., PC2)
    - D5 to a GPIO pin (e.g., PC3)
    - D6 to a GPIO pin (e.g., PC4)
    - D7 to a GPIO pin (e.g., PC5)
  - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the I2C peripheral and the GPIO pins for the LCD.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize I2C communication with the MPU6050 sensor using the HAL library.
   - Write code to read acceleration data from the MPU6050.
   - Calculate the angle of tilt using the acceleration data.
   - If using an LCD, use the LCD library to display the tilt angle.
   - If using serial output, use `HAL_UART_Transmit()` to send the tilt angle data to a serial terminal.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Observe the tilt angle readings on the LCD or the Serial Monitor.
5. **Calibrate and Experiment:**
   - Verify the accuracy of the tilt angle readings by comparing them with known angles.

- Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the MPU6050 sensor?
2. What challenges did you encounter while setting up the tilt sensing system and how did you resolve them?
3. How can tilt sensing be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors and LCD to prevent damage to the components and the microcontroller.

Experiment 17b: Tilt Sensing System: Implementing Tilt Sensing and Displaying the Angle of Tilt for Various Applications

Objective: To measure and display the angle of tilt using an STM32 microcontroller and an accelerometer (e.g., MPU6050) for various applications.

Description: In this experiment, we will use an STM32 microcontroller and an accelerometer (MPU6050) to measure the angle of tilt. The angle of tilt will be calculated from the accelerometer data and displayed on an LCD or via serial output. This project demonstrates how to use sensor data to determine orientation and angle, which can be applied in various applications such as robotics, gaming, and device orientation detection.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- MPU6050 sensor module
- LCD (16x2) or Serial Monitor
- Breadboard
- Jumper wires
- Potentiometer (10k ohms) (if using LCD)
- USB cable for programming

Background Information: Tilt sensing is the process of determining the orientation of an object relative to the ground. Accelerometers can measure acceleration due to gravity, which can be used to calculate the angle of tilt. The MPU6050 sensor provides data on acceleration along three axes, which can be used to determine the tilt angle.

Pre-Lab Questions:

1. How does the MPU6050 sensor measure acceleration?
2. What is the principle behind calculating the angle of tilt using accelerometer data?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**

- Connect the VCC and GND pins of the MPU6050 to the 3.3V/5V and GND pins on the STM32.
- Connect the SDA (data) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
- Connect the SCL (clock) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
- If using an LCD:
  - Connect the LCD to the STM32 as follows:
    - RS to a GPIO pin (e.g., PC0)
    - E to a GPIO pin (e.g., PC1)
    - D4 to a GPIO pin (e.g., PC2)
    - D5 to a GPIO pin (e.g., PC3)
    - D6 to a GPIO pin (e.g., PC4)
    - D7 to a GPIO pin (e.g., PC5)
  - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.

2. **Configure the Microcontroller:**
   - Open STM32CubeMX and create a new project.
   - Configure the I2C peripheral and the GPIO pins for the LCD.
   - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
   - Initialize I2C communication with the MPU6050 sensor using the HAL library.
   - Write code to read acceleration data from the MPU6050.
   - Calculate the angle of tilt using the acceleration data.
   - If using an LCD, use the LCD library to display the tilt angle.
   - If using serial output, use `HAL_UART_Transmit()` to send the tilt angle data to a serial terminal.
4. **Upload and Test:**
   - Compile and upload the code to the STM32 board.
   - Observe the tilt angle readings on the LCD or the Serial Monitor.
5. **Calibrate and Experiment:**
   - Verify the accuracy of the tilt angle readings by comparing them with known angles.

- Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the MPU6050 sensor?
2. What challenges did you encounter while setting up the tilt sensing system and how did you resolve them?
3. How can tilt sensing be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors and LCD to prevent damage to the components and the microcontroller.

Experiment 18a: Arduino RC Timer Circuit: Experimenting with Resistor-Capacitor Circuits to Create Timing Mechanisms

---

Experiment Title: RC Timer Circuit: Experimenting with Resistor-Capacitor Circuits to Create Timing Mechanisms

Objective: To understand and experiment with resistor-capacitor (RC) circuits to create timing mechanisms and measure time intervals using an Arduino.

Description: In this experiment, we will use an Arduino to create and measure time intervals using RC circuits. By charging and discharging a capacitor through a resistor, we can create predictable time delays. This project demonstrates the principles of RC circuits and their applications in timing mechanisms.

Materials Needed:

- Arduino Uno
- Capacitors (various values, e.g., 10µF, 100µF)
- Resistors (various values, e.g., 1kΩ, 10kΩ)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: An RC circuit consists of a resistor and a capacitor connected in series or parallel. When a voltage is applied, the capacitor charges or discharges through the resistor, creating a time delay. The time constant (τ) of an RC circuit is given by $\tau = R \times C$, where R is the resistance and C is the capacitance. This time constant determines how quickly the capacitor charges or discharges.

Pre-Lab Questions:

1. What is an RC circuit and how does it work?
2. How is the time constant of an RC circuit calculated?
3. What are some practical applications of RC circuits in electronics?

Procedure:

1. **Setup the Circuit:**
    - Connect a resistor and a capacitor in series on the breadboard.
    - Connect one end of the resistor to a digital pin on the Arduino (e.g., pin 8).
    - Connect the junction between the resistor and capacitor to an analog input pin on the Arduino (e.g., A0).
    - Connect the other end of the capacitor to the ground (GND) pin on the Arduino.
    - Connect a reference voltage (5V) to the resistor-capacitor junction through a separate digital pin (e.g., pin 9).
2. **Program the Arduino:**
    - Write a program to set the digital pin high (5V) to charge the capacitor and then set it low (0V) to discharge the capacitor.
    - Use the `analogRead()` function to measure the voltage across the capacitor.
    - Calculate the time taken for the capacitor to charge or discharge to a specific voltage level.
3. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the charging and discharging curves by monitoring the voltage across the capacitor over time.
4. **Experiment with Different Values:**
    - Repeat the experiment with different resistor and capacitor values.
    - Document the changes in the time constant and their effects on the charging/discharging behavior.

Post-Lab Questions:

1. How did you calculate the time constant for each RC circuit configuration?
2. What challenges did you encounter while measuring the time intervals and how did you resolve them?
3. How can RC circuits be applied in real-world timing applications?

Experiment 18b: STM RC Timer Circuit: Experimenting with Resistor-Capacitor Circuits to Create Timing Mechanisms

Objective: To understand and experiment with resistor-capacitor (RC) circuits to create timing mechanisms and measure time intervals using an STM32 microcontroller.

Description: In this experiment, we will use an STM32 microcontroller to create and measure time intervals using RC circuits. By charging and discharging a capacitor through a resistor, we can create predictable time delays. This project demonstrates the principles of RC circuits and their applications in timing mechanisms.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- Capacitors (various values, e.g., 10µF, 100µF)
- Resistors (various values, e.g., 1kΩ, 10kΩ)
- Breadboard
- Jumper wires
- USB cable for programming

Background Information: An RC circuit consists of a resistor and a capacitor connected in series or parallel. When a voltage is applied, the capacitor charges or discharges through the resistor, creating a time delay. The time constant (τ) of an RC circuit is given by $\tau = R \times C$, where R is the resistance and C is the capacitance. This time constant determines how quickly the capacitor charges or discharges.

Pre-Lab Questions:

1. What is an RC circuit and how does it work?
2. How is the time constant of an RC circuit calculated?
3. What are some practical applications of RC circuits in electronics?

Procedure:

1. **Setup the Circuit:**
    - Connect a resistor and a capacitor in series on the breadboard.
    - Connect one end of the resistor to a GPIO pin on the STM32 (e.g., PA0).

    - Connect the junction between the resistor and capacitor to an analog input pin on the STM32 (e.g., PA1).
    - Connect the other end of the capacitor to the ground (GND) pin on the STM32.
    - Connect a reference voltage (3.3V) to the resistor-capacitor junction through a separate GPIO pin (e.g., PA2).
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the GPIO pins for output and the ADC peripheral for analog input.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize the GPIO and ADC peripherals.
    - Write code to set the GPIO pin high (3.3V) to charge the capacitor and then set it low (0V) to discharge the capacitor.
    - Use `HAL_ADC_GetValue()` to measure the voltage across the capacitor.
    - Calculate the time taken for the capacitor to charge or discharge to a specific voltage level.
4. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Observe the charging and discharging curves by monitoring the voltage across the capacitor over time.
5. **Experiment with Different Values:**
    - Repeat the experiment with different resistor and capacitor values.
    - Document the changes in the time constant and their effects on the charging/discharging behavior.

Post-Lab Questions:

1. How did you calculate the time constant for each RC circuit configuration?
2. What challenges did you encounter while measuring the time intervals and how did you resolve them?
3. How can RC circuits be applied in real-world timing applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the resistors and capacitors to prevent damage to the components and the microcontroller.

Experiment 19a: Arduino Signal Filtering: Designing Low-pass and High-pass Filters for Signal Processing

---

Experiment Title: Signal Filtering: Designing Low-pass and High-pass Filters for Signal Processing

Objective: To design and implement low-pass and high-pass filters using passive components and to analyze their effect on signals using an Arduino.

Description: In this experiment, we will design and implement low-pass and high-pass filters using resistors and capacitors. We will then use an Arduino to generate and analyze signals, demonstrating how these filters affect different frequency components.

Materials Needed:

- Arduino Uno
- Resistors (various values, e.g., 1kΩ, 10kΩ)
- Capacitors (various values, e.g., 10nF, 100nF)
- Breadboard
- Jumper wires
- Signal generator (optional, for more accurate signal testing)
- Oscilloscope (optional, for detailed signal analysis)
- USB cable for programming

Background Information: Filters are fundamental components in signal processing, used to remove unwanted parts of a signal or to extract useful parts. A low-pass filter allows signals with a frequency lower than a certain cutoff frequency to pass through, while attenuating higher frequency signals. Conversely, a high-pass filter allows signals with a frequency higher than the cutoff frequency to pass through, while attenuating lower frequency signals.

Pre-Lab Questions:

1. What is the difference between a low-pass filter and a high-pass filter?
2. How are the cutoff frequencies of RC filters calculated?
3. What are some practical applications of low-pass and high-pass filters?

Procedure:

Part A: Low-pass Filter

1. **Design the Low-pass Filter:**
    - Choose a resistor (R) and a capacitor (C) to create a low-pass filter.
    - Calculate the cutoff frequency using the formula fc=12πRCf_c = \frac{1}{2\pi RC}fc=2πRC1.
2. **Setup the Circuit:**
    - Connect the resistor and capacitor in series on the breadboard.
    - Connect the input signal (e.g., from a signal generator or Arduino PWM output) to the resistor.
    - Connect the junction between the resistor and capacitor to an analog input pin on the Arduino (e.g., A0).
    - Connect the other end of the capacitor to ground.
3. **Program the Arduino:**
    - Write a program to generate a test signal using PWM.
    - Use the `analogRead()` function to measure the output signal after the filter.
    - Optionally, use the `Serial.print()` function to send the measured values to the Serial Monitor for analysis.
4. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the output signal on the Serial Monitor or an oscilloscope.

Part B: High-pass Filter

1. **Design the High-pass Filter:**
    - Choose a resistor (R) and a capacitor (C) to create a high-pass filter.
    - Calculate the cutoff frequency using the formula fc=12πRCf_c = \frac{1}{2\pi RC}fc=2πRC1.
2. **Setup the Circuit:**
    - Connect the resistor and capacitor in series on the breadboard.
    - Connect the input signal (e.g., from a signal generator or Arduino PWM output) to the capacitor.

    - Connect the junction between the capacitor and resistor to an analog input pin on the Arduino (e.g., A0).
    - Connect the other end of the resistor to ground.
3. **Program the Arduino:**
    - Write a program to generate a test signal using PWM.
    - Use the `analogRead()` function to measure the output signal after the filter.
    - Optionally, use the `Serial.print()` function to send the measured values to the Serial Monitor for analysis.
4. **Upload and Test:**
    - Upload the code to the Arduino.
    - Observe the output signal on the Serial Monitor or an oscilloscope.

Experiment with Different Values:

- Repeat the experiments with different resistor and capacitor values.
- Document the changes in the cutoff frequency and their effects on the signal filtering.

Post-Lab Questions:

1. How did you calculate the cutoff frequency for each filter configuration?
2. What challenges did you encounter while designing and testing the filters, and how did you resolve them?
3. How can signal filtering be applied in real-world applications?

Experiment 19b: Signal Filtering: Designing Low-pass and High-pass Filters for Signal Processing

Objective: To design and implement low-pass and high-pass filters using passive components and to analyze their effect on signals using an STM32 microcontroller.

Description: In this experiment, we will design and implement low-pass and high-pass filters using resistors and capacitors. We will then use an STM32 microcontroller to generate and analyze signals, demonstrating how these filters affect different frequency components.

Materials Needed:

- STM32 Development Board (e.g., STM32F4 Discovery)
- Resistors (various values, e.g., 1kΩ, 10kΩ)
- Capacitors (various values, e.g., 10nF, 100nF)
- Breadboard
- Jumper wires
- Signal generator (optional, for more accurate signal testing)
- Oscilloscope (optional, for detailed signal analysis)
- USB cable for programming

Background Information: Filters are fundamental components in signal processing, used to remove unwanted parts of a signal or to extract useful parts. A low-pass filter allows signals with a frequency lower than a certain cutoff frequency to pass through, while attenuating higher frequency signals. Conversely, a high-pass filter allows signals with a frequency higher than the cutoff frequency to pass through, while attenuating lower frequency signals.

Pre-Lab Questions:

1. What is the difference between a low-pass filter and a high-pass filter?
2. How are the cutoff frequencies of RC filters calculated?
3. What are some practical applications of low-pass and high-pass filters?

Procedure:

Part A: Low-pass Filter

1. **Design the Low-pass Filter:**
    - Choose a resistor (R) and a capacitor (C) to create a low-pass filter.
    - Calculate the cutoff frequency using the formula fc=12πRCf_c = \frac{1}{2\pi RC}fc=2πRC1.
2. **Setup the Circuit:**
    - Connect the resistor and capacitor in series on the breadboard.
    - Connect the input signal (e.g., from a signal generator or STM32 PWM output) to the resistor.
    - Connect the junction between the resistor and capacitor to an analog input pin on the STM32 (e.g., PA0).
    - Connect the other end of the capacitor to ground.
3. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the GPIO pins for PWM output and ADC input.
    - Generate the initialization code and open it in STM32CubeIDE.
4. **Write the Code:**
    - Initialize the PWM output and ADC peripherals.
    - Write code to generate a test signal using PWM.
    - Use `HAL_ADC_GetValue()` to measure the output signal after the filter.
    - Optionally, use `HAL_UART_Transmit()` to send the measured values to a serial terminal for analysis.
5. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Observe the output signal on a serial terminal or an oscilloscope.

Part B: High-pass Filter

1. **Design the High-pass Filter:**
    - Choose a resistor (R) and a capacitor (C) to create a high-pass filter.
    - Calculate the cutoff frequency using the formula fc=12πRCf_c = \frac{1}{2\pi RC}fc=2πRC1.
2. **Setup the Circuit:**
    - Connect the resistor and capacitor in series on the breadboard.

    - Connect the input signal (e.g., from a signal generator or STM32 PWM output) to the capacitor.
    - Connect the junction between the capacitor and resistor to an analog input pin on the STM32 (e.g., PA0).
    - Connect the other end of the resistor to ground.
3. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the GPIO pins for PWM output and ADC input.
    - Generate the initialization code and open it in STM32CubeIDE.
4. **Write the Code:**
    - Initialize the PWM output and ADC peripherals.
    - Write code to generate a test signal using PWM.
    - Use `HAL_ADC_GetValue()` to measure the output signal after the filter.
    - Optionally, use `HAL_UART_Transmit()` to send the measured values to a serial terminal for analysis.
5. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Observe the output signal on a serial terminal or an oscilloscope.

Experiment with Different Values:

- Repeat the experiments with different resistor and capacitor values.
- Document the changes in the cutoff frequency and their effects on the signal filtering.

Post-Lab Questions:

1. How did you calculate the cutoff frequency for each filter configuration?
2. What challenges did you encounter while designing and testing the filters, and how did you resolve them?
3. How can signal filtering be applied in real-world applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the resistors and capacitors to prevent damage to the components and the microcontroller.

Advanced EV Projects Using STM32

Project 20: Battery Monitoring System

Objective: Measure and display the voltage of a simulated battery pack.

Components:

- STM32F103C8T6
- Voltage Detection Sensor Module
- JHD162G M7 (16x2 LCD)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller to measure the voltage of a simulated battery pack using a voltage detection sensor module. The measured voltage will be displayed on a JHD162G M7 (16x2 LCD).

Background Information: Battery monitoring is crucial in electric vehicles (EVs) to ensure the battery pack operates within safe limits. The voltage detection sensor module provides a scaled-down voltage that the microcontroller can read. This voltage is then displayed on an LCD for real-time monitoring.

Pre-Lab Questions:

1. How does a voltage detection sensor module work?
2. Why is it important to monitor battery voltage in EVs?
3. How can the STM32's ADC be used to read analog voltage levels?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the voltage detection sensor module to the 3.3V and GND pins on the STM32.

    - Connect the output pin of the voltage detection sensor module to an analog input pin on the STM32 (e.g., PA0).
    - Connect the LCD to the STM32 as follows:
        - RS to GPIO pin (e.g., PC0)
        - E to GPIO pin (e.g., PC1)
        - D4 to GPIO pin (e.g., PC2)
        - D5 to GPIO pin (e.g., PC3)
        - D6 to GPIO pin (e.g., PC4)
        - D7 to GPIO pin (e.g., PC5)
    - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the ADC peripheral for the voltage detection sensor module input.
    - Configure the GPIO pins for the LCD.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize the ADC and GPIO peripherals.
    - Write code to read the analog voltage from the sensor module using `HAL_ADC_GetValue()`.
    - Convert the ADC value to the corresponding voltage.
    - Display the voltage on the LCD using the LCD library.
4. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Observe the voltage readings on the LCD.
5. **Calibrate and Experiment:**
    - Verify the accuracy of the voltage readings by comparing them with known values.
    - Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the ADC to measure voltage levels?

2. What challenges did you encounter while setting up the battery monitoring system and how did you resolve them?
3. How can battery monitoring be applied in real-world EV applications?

Project 21: Current Monitoring System

Objective: Measure and display the current drawn by a simulated EV motor.

Components:

- STM32F103C8T6
- ACS712 (Current Sensor)
- JHD162G M7 (16x2 LCD)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller to measure the current drawn by a simulated EV motor using an ACS712 current sensor. The measured current will be displayed on a JHD162G M7 (16x2 LCD).

Background Information: Current monitoring is essential in EVs to ensure that the motor operates within safe limits. The ACS712 current sensor provides an analog output voltage proportional to the current flowing through it. This voltage can be read by the microcontroller and displayed on an LCD.

Pre-Lab Questions:

1. How does the ACS712 current sensor work?
2. Why is it important to monitor current in EV motors?
3. How can the STM32's ADC be used to read analog current levels?

Procedure:

1. **Setup the Circuit:**
    - Connect the VCC and GND pins of the ACS712 to the 5V and GND pins on the STM32.
    - Connect the output pin of the ACS712 to an analog input pin on the STM32 (e.g., PA0).
    - Connect the simulated motor in series with the ACS712's input terminals.

    - Connect the LCD to the STM32 as follows:
        - RS to GPIO pin (e.g., PC0)
        - E to GPIO pin (e.g., PC1)
        - D4 to GPIO pin (e.g., PC2)
        - D5 to GPIO pin (e.g., PC3)
        - D6 to GPIO pin (e.g., PC4)
        - D7 to GPIO pin (e.g., PC5)
    - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the ADC peripheral for the current sensor input.
    - Configure the GPIO pins for the LCD.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize the ADC and GPIO peripherals.
    - Write code to read the analog voltage from the current sensor using `HAL_ADC_GetValue()`.
    - Convert the ADC value to the corresponding current using the sensor's transfer function.
    - Display the current on the LCD using the LCD library.
4. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Observe the current readings on the LCD.
5. **Calibrate and Experiment:**
    - Verify the accuracy of the current readings by comparing them with known values.
    - Document any adjustments made to improve accuracy and their effects on the readings.

Post-Lab Questions:

1. How did you use the ADC to measure current levels?
2. What challenges did you encounter while setting up the current monitoring system and how did you resolve them?

3. How can current monitoring be applied in real-world EV applications?

Project 22: PWM Motor Control Simulation

Objective: Control the speed of a simulated motor using PWM.

Components:

- STM32F103C8T6
- LED System
- Breadboard
- Jumper wires
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller to control the speed of a simulated motor using PWM. The LED system will visually represent the motor speed.

Background Information: Pulse Width Modulation (PWM) is a technique used to control the power delivered to electrical devices such as motors. By varying the duty cycle of the PWM signal, the effective voltage and hence the speed of the motor can be controlled.

Pre-Lab Questions:

1. What is PWM and how does it work?
2. How can PWM be used to control motor speed?
3. How can the STM32's PWM peripheral be configured to generate PWM signals?

Procedure:

1. **Setup the Circuit:**
    - Connect the anode of an LED to a PWM-capable GPIO pin on the STM32 (e.g., PA0).
    - Connect the cathode of the LED to the GND pin on the STM32.
    - Use a current-limiting resistor (220 ohms) in series with the LED.
    - Repeat for additional LEDs if using a multi-LED system.
2. **Configure the Microcontroller:**
    - Open STM32CubeMX and create a new project.
    - Configure the GPIO pins for PWM output.

    - Configure the TIM peripheral to generate PWM signals.
    - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - Initialize the GPIO and TIM peripherals.
    - Write code to generate PWM signals with varying duty cycles.
    - Implement control logic to adjust the PWM duty cycle based on user input or predefined conditions.
    - Use the LEDs to visually represent the motor speed.
4. **Upload and Test:**
    - Compile and upload the code to the STM32 board.
    - Observe the LED brightness as a representation of the motor speed.
5. **Experiment with Different Duty Cycles:**
    - Change the PWM duty cycle and observe the corresponding changes in LED brightness.
    - Document the effects of different duty cycles on the simulated motor speed.

Post-Lab Questions:

1. How did you use PWM to control the simulated motor speed?
2. What challenges did you encounter while setting up the PWM motor control simulation and how did you resolve them?
3. How can PWM motor control be applied in real-world EV applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors, LEDs, and other components to prevent damage to the components and the microcontroller.

Project 23: Accelerometer-based Impact Detection

Objective: Use an accelerometer to detect tilt and impacts.

Components:

- STM32F103C8T6
- MPU6050 (Accelerometer and Gyroscope)
- Breadboard
- Jumper wires
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller and an MPU6050 accelerometer to detect tilt and impacts. The accelerometer will provide data on acceleration and angular velocity, which will be used to detect changes in orientation and sudden impacts. This project is essential for developing safety and stability systems in electric vehicles (EVs).

Background Information: Accelerometers are widely used in automotive applications to detect impacts and monitor orientation. The MPU6050 sensor combines a 3-axis accelerometer and a 3-axis gyroscope, providing comprehensive motion tracking capabilities. By analyzing the data from these sensors, we can detect tilt angles and sudden impacts, which are crucial for vehicle safety systems.

Pre-Lab Questions:

1. How does the MPU6050 sensor measure acceleration and angular velocity?
2. What are some practical applications of impact detection in EVs?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - **Connecting the MPU6050:**
        - Connect the VCC and GND pins of the MPU6050 to the 3.3V and GND pins on the STM32, respectively.
        - Connect the SDA (data) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).

        - Connect the SCL (clock) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
2. **Configure the Microcontroller:**
    - **Using STM32CubeMX:**
        - Open STM32CubeMX and create a new project.
        - Configure the I2C peripheral for communication with the MPU6050.
        - Configure the necessary GPIO pins.
        - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - **Initialization:**
        - Initialize the I2C communication with the MPU6050 sensor using the HAL library.
        - Write code to configure the MPU6050 and start data acquisition.
    - **Data Acquisition:**
        - Read acceleration and gyroscope data from the MPU6050.
    - **Processing Data:**
        - Implement algorithms to detect tilt and impact based on the accelerometer and gyroscope data.
    - **Output Data:**
        - Output detection results to a serial terminal for analysis and debugging.
4. **Upload and Test:**
    - **Uploading Code:**
        - Compile and upload the code to the STM32 board.
    - **Testing:**
        - Test the system by tilting the MPU6050 and simulating impacts.
        - Observe the results on the serial terminal.
5. **Calibrate and Experiment:**
    - **Calibration:**
        - Adjust the sensitivity of the impact and tilt detection in the code.
        - Verify the accuracy of the system by comparing it with known tilt angles and impact forces.
    - **Documentation:**
        - Document any adjustments made and their effects on the system's responsiveness.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the MPU6050 sensor?
2. What challenges did you encounter while setting up the impact detection system and how did you resolve them?
3. How can impact detection be applied in real-world EV applications?

Project 24: Environmental Monitoring in EV

Objective: Monitor the environmental conditions within an EV.

Components:

- STM32F103C8T6
- BMP280 (Temperature and Pressure Sensor)
- JHD162G M7 (16x2 LCD)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller to monitor environmental conditions within an EV using a BMP280 sensor. The measured temperature and pressure data will be displayed on a JHD162G M7 (16x2 LCD). This project is crucial for maintaining optimal environmental conditions for passenger comfort and system efficiency in EVs.

Background Information: Monitoring environmental conditions inside an EV is crucial for passenger comfort and system efficiency. The BMP280 sensor provides accurate measurements of temperature and atmospheric pressure, which can be used to monitor the cabin environment. This data can be displayed on an LCD for real-time monitoring.

Pre-Lab Questions:

1. How does the BMP280 sensor measure temperature and pressure?
2. What are the practical applications of environmental monitoring in EVs?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - **Connecting the BMP280:**
        - Connect the VCC and GND pins of the BMP280 to the 3.3V/5V and GND pins on the STM32.

        - Connect the SDA (data) pin of the BMP280 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).
        - Connect the SCL (clock) pin of the BMP280 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
    - **Connecting the LCD:**
        - Connect the LCD to the STM32 as follows:
            - RS to GPIO pin (e.g., PC0)
            - E to GPIO pin (e.g., PC1)
            - D4 to GPIO pin (e.g., PC2)
            - D5 to GPIO pin (e.g., PC3)
            - D6 to GPIO pin (e.g., PC4)
            - D7 to GPIO pin (e.g., PC5)
        - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Configure the Microcontroller:**
    - **Using STM32CubeMX:**
        - Open STM32CubeMX and create a new project.
        - Configure the I2C peripheral for communication with the BMP280.
        - Configure the GPIO pins for the LCD.
        - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - **Initialization:**
        - Initialize I2C communication with the BMP280 sensor using the HAL library.
        - Write code to configure the BMP280 and start data acquisition.
    - **Data Acquisition:**
        - Read temperature and pressure data from the BMP280.
    - **Displaying Data:**
        - Display the data on the LCD using the LiquidCrystal library.
    - **Output Data:**
        - Output the environmental data to the serial terminal for logging and further analysis.
4. **Upload and Test:**
    - **Uploading Code:**

        - Compile and upload the code to the STM32 board.
    - **Testing:**
        - Observe the temperature and pressure readings on the LCD.
        - Verify the accuracy of the readings by comparing them with known reference values.
5. **Calibrate and Experiment:**
    - **Calibration:**
        - Adjust the calibration parameters in the code to improve the accuracy of the sensor readings.
    - **Documentation:**
        - Document any adjustments made and their effects on the accuracy and reliability of the system.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the BMP280 sensor?
2. What challenges did you encounter while setting up the environmental monitoring system and how did you resolve them?
3. How can environmental monitoring be applied in real-world EV applications?

Project 25: Battery Overcurrent Protection

Objective: Detect and respond to overcurrent situations in a simulated battery system.

Components:

- STM32F103C8T6
- ACS712 (Current Sensor)
- LED System
- 5V Passive Buzzer
- Breadboard
- Jumper wires
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller to detect overcurrent situations in a simulated battery system using an ACS712 current sensor. When an overcurrent condition is detected, the system will activate an LED and a buzzer to indicate the fault. This project is essential for protecting battery systems in EVs from damage due to excessive current.

Background Information: Overcurrent protection is crucial in EVs to prevent damage to the battery and associated circuitry. The ACS712 current sensor provides an analog output voltage proportional to the current flowing through it, which can be used to detect overcurrent conditions. This voltage can be read by the microcontroller and used to trigger protective measures.

Pre-Lab Questions:

1. How does the ACS712 current sensor work?
2. Why is overcurrent protection important in battery systems?
3. How can the STM32's ADC be used to read analog current levels?

Procedure:

1. **Setup the Circuit:**
    - **Connecting the ACS712:**

        - Connect the VCC and GND pins of the ACS712 to the 5V and GND pins on the STM32.
        - Connect the output pin of the ACS712 to an analog input pin on the STM32 (e.g., PA0).
        - Connect the simulated battery system in series with the ACS712's input terminals.
    - **Connecting the LED and Buzzer:**
        - Connect the anode of an LED to a GPIO pin on the STM32 (e.g., PA1).
        - Connect the cathode of the LED to the GND pin on the STM32, with a current-limiting resistor (220 ohms) in series.
        - Connect the positive terminal of the buzzer to a GPIO pin on the STM32 (e.g., PA2).
        - Connect the negative terminal of the buzzer to the GND pin on the STM32.
2. **Configure the Microcontroller:**
    - **Using STM32CubeMX:**
        - Open STM32CubeMX and create a new project.
        - Configure the ADC peripheral for the current sensor input.
        - Configure the GPIO pins for the LED and buzzer.
        - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - **Initialization:**
        - Initialize the ADC and GPIO peripherals.
        - Write code to read the analog voltage from the current sensor using `HAL_ADC_GetValue()`.
    - **Data Processing:**
        - Convert the ADC value to the corresponding current using the sensor's transfer function.
    - **Overcurrent Detection:**
        - Implement logic to detect overcurrent conditions based on the measured current.
        - Activate the LED and buzzer when an overcurrent condition is detected.
    - **Output Data:**
        - Output detection results to a serial terminal for analysis.

4. **Upload and Test:**
    - **Uploading Code:**
        - Compile and upload the code to the STM32 board.
    - **Testing:**
        - Test the system by simulating overcurrent conditions and observing the response of the LED and buzzer.
        - Verify the accuracy and responsiveness of the system.
5. **Calibrate and Experiment:**
    - **Calibration:**
        - Adjust the overcurrent threshold in the code to ensure accurate detection.
    - **Documentation:**
        - Document any adjustments made and their effects on the system's performance and reliability.

Post-Lab Questions:

1. How did you use the ADC to measure current levels?
2. What challenges did you encounter while setting up the overcurrent protection system and how did you resolve them?
3. How can overcurrent protection be applied in real-world EV applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors, LEDs, and other components to prevent damage to the components and the microcontroller.

Project 26: Simulated CAN Bus Communication

Objective: Simulate CAN bus communication using the STM32 and display data.

Components:

- STM32F103C8T6
- SN65HVD230 CAN Bus Transceiver
- JHD162G M7 (16x2 LCD)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller to simulate CAN bus communication. The data transmitted over the CAN bus will be displayed on a JHD162G M7 (16x2 LCD). This project demonstrates the principles of CAN bus communication, which is widely used in automotive applications for robust and reliable data exchange between different electronic control units (ECUs).

Background Information: The Controller Area Network (CAN) bus is a robust vehicle bus standard designed to facilitate communication among various electronic components in a vehicle without a host computer. CAN bus is used for real-time data exchange, ensuring timely responses to critical vehicle operations.

Pre-Lab Questions:

1. What is the CAN bus and how does it facilitate communication in vehicles?
2. What are the roles of the CAN transceiver and the microcontroller in CAN communication?
3. How can the STM32's CAN peripheral be configured and used?

Procedure:

1. **Setup the Circuit:**
    - **Connecting the CAN Transceiver:**

        - Connect the VCC and GND pins of the SN65HVD230 to the 3.3V and GND pins on the STM32.
        - Connect the CANH and CANL pins of the transceiver to a CAN bus line (twisted pair of wires).
        - Connect the RXD pin of the transceiver to a CAN RX-capable GPIO pin on the STM32 (e.g., PA11).
        - Connect the TXD pin of the transceiver to a CAN TX-capable GPIO pin on the STM32 (e.g., PA12).
    - **Connecting the LCD:**
        - Connect the LCD to the STM32 as follows:
            - RS to GPIO pin (e.g., PC0)
            - E to GPIO pin (e.g., PC1)
            - D4 to GPIO pin (e.g., PC2)
            - D5 to GPIO pin (e.g., PC3)
            - D6 to GPIO pin (e.g., PC4)
            - D7 to GPIO pin (e.g., PC5)
        - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Configure the Microcontroller:**
    - **Using STM32CubeMX:**
        - Open STM32CubeMX and create a new project.
        - Configure the CAN peripheral for communication.
        - Configure the GPIO pins for CAN and the LCD.
        - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - **Initialization:**
        - Initialize the CAN peripheral and configure the CAN bus settings (e.g., baud rate, filters).
        - Initialize the LCD using the LiquidCrystal library.
    - **Data Transmission and Reception:**
        - Write code to send and receive CAN messages.
        - Implement functions to display received CAN data on the LCD.
    - **Testing:**

        - Simulate CAN bus communication by sending predefined messages and displaying them on the LCD.
4. **Upload and Test:**
    - **Uploading Code:**
        - Compile and upload the code to the STM32 board.
    - **Testing:**
        - Test the system by transmitting and receiving CAN messages.
        - Verify the accuracy and responsiveness of the system.
5. **Calibrate and Experiment:**
    - **Experimentation:**
        - Experiment with different CAN bus configurations and message types.
        - Document any adjustments made and their effects on the communication performance.

Post-Lab Questions:

1. How did you configure the CAN peripheral for communication?
2. What challenges did you encounter while setting up the CAN bus communication system and how did you resolve them?
3. How can CAN bus communication be applied in real-world EV applications?

Project 27: Energy Consumption Display

Objective: Calculate and display the energy consumption of a simulated EV system.

Components:

- STM32F103C8T6
- ACS712 (Current Sensor)
- Voltage Detection Sensor Module
- JHD162G M7 (16x2 LCD)
- Breadboard
- Jumper wires
- Potentiometer (10k ohms)
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller to calculate and display the energy consumption of a simulated EV system. The current and voltage will be measured using an ACS712 current sensor and a voltage detection sensor module, respectively. The calculated energy consumption will be displayed on a JHD162G M7 (16x2 LCD).

Background Information: Monitoring energy consumption in EVs is crucial for optimizing efficiency and extending battery life. By measuring the current and voltage, we can calculate the power consumption in real-time and accumulate energy usage over time.

Pre-Lab Questions:

1. How do the ACS712 current sensor and voltage detection sensor module work?
2. Why is it important to monitor energy consumption in EVs?
3. How can the STM32's ADC be used to read analog current and voltage levels?

Procedure:

1. **Setup the Circuit:**
    - **Connecting the Sensors:**
        - Connect the VCC and GND pins of the ACS712 to the 5V and GND pins on the STM32.

        - Connect the output pin of the ACS712 to an analog input pin on the STM32 (e.g., PA0).
        - Connect the VCC and GND pins of the voltage detection sensor module to the 3.3V and GND pins on the STM32.
        - Connect the output pin of the voltage detection sensor module to another analog input pin on the STM32 (e.g., PA1).
    - **Connecting the LCD:**
        - Connect the LCD to the STM32 as follows:
            - RS to GPIO pin (e.g., PC0)
            - E to GPIO pin (e.g., PC1)
            - D4 to GPIO pin (e.g., PC2)
            - D5 to GPIO pin (e.g., PC3)
            - D6 to GPIO pin (e.g., PC4)
            - D7 to GPIO pin (e.g., PC5)
        - Connect the potentiometer's middle pin to the LCD's V0 pin for contrast adjustment, and the other two pins to 5V and GND.
2. **Configure the Microcontroller:**
    - **Using STM32CubeMX:**
        - Open STM32CubeMX and create a new project.
        - Configure the ADC peripheral for the current and voltage sensor inputs.
        - Configure the GPIO pins for the LCD.
        - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - **Initialization:**
        - Initialize the ADC and GPIO peripherals.
        - Write code to read the analog voltage from the current and voltage sensors using `HAL_ADC_GetValue()`.
    - **Calculations:**
        - Convert the ADC values to the corresponding current and voltage.
        - Calculate the power consumption using the formula: Power (W) = Voltage (V) * Current (A).
        - Accumulate the energy consumption over time.
    - **Displaying Data:**

        - Display the real-time power and accumulated energy consumption on the LCD using the LiquidCrystal library.
    - **Testing:**
        - Simulate different current and voltage conditions and observe the calculated energy consumption.
4. **Upload and Test:**
    - **Uploading Code:**
        - Compile and upload the code to the STM32 board.
    - **Testing:**
        - Test the system by measuring current and voltage and verifying the accuracy of the energy consumption calculations.
5. **Calibrate and Experiment:**
    - **Calibration:**
        - Adjust the calibration parameters in the code to improve the accuracy of the sensor readings.
    - **Documentation:**
        - Document any adjustments made and their effects on the accuracy and reliability of the system.

Post-Lab Questions:

1. How did you use the ADC to measure current and voltage levels?
2. What challenges did you encounter while setting up the energy consumption display system and how did you resolve them?
3. How can energy consumption monitoring be applied in real-world EV applications?

Project 28: Tilt-based Safety Alert System

Objective: Trigger safety alerts based on vehicle tilt.

Components:

- STM32F103C8T6
- MPU6050 (Accelerometer and Gyroscope)
- LED System
- 5V Passive Buzzer
- Breadboard
- Jumper wires
- USB cable for programming

Description: In this project, we will use an STM32F103C8T6 microcontroller and an MPU6050 accelerometer to detect vehicle tilt and trigger safety alerts. The system will activate an LED and a buzzer when a tilt beyond a safe threshold is detected, enhancing vehicle safety.

Background Information: Tilt detection is critical for vehicle safety, especially in EVs with high centers of gravity. The MPU6050 sensor combines a 3-axis accelerometer and a 3-axis gyroscope, providing comprehensive motion tracking capabilities. By analyzing the data from these sensors, we can detect unsafe tilt angles and trigger alerts to prevent accidents.

Pre-Lab Questions:

1. How does the MPU6050 sensor measure acceleration and angular velocity?
2. What are some practical applications of tilt detection in EVs?
3. How can the STM32's I2C peripheral be used to communicate with I2C sensors?

Procedure:

1. **Setup the Circuit:**
    - **Connecting the MPU6050:**
        - Connect the VCC and GND pins of the MPU6050 to the 3.3V and GND pins on the STM32.
        - Connect the SDA (data) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB7).

        - Connect the SCL (clock) pin of the MPU6050 to an I2C-capable GPIO pin on the STM32 (e.g., PB6).
    - **Connecting the LED and Buzzer:**
        - Connect the anode of an LED to a GPIO pin on the STM32 (e.g., PA1).
        - Connect the cathode of the LED to the GND pin on the STM32, with a current-limiting resistor (220 ohms) in series.
        - Connect the positive terminal of the buzzer to a GPIO pin on the STM32 (e.g., PA2).
        - Connect the negative terminal of the buzzer to the GND pin on the STM32.
2. **Configure the Microcontroller:**
    - **Using STM32CubeMX:**
        - Open STM32CubeMX and create a new project.
        - Configure the I2C peripheral for communication with the MPU6050.
        - Configure the GPIO pins for the LED and buzzer.
        - Generate the initialization code and open it in STM32CubeIDE.
3. **Write the Code:**
    - **Initialization:**
        - Initialize I2C communication with the MPU6050 sensor using the HAL library.
        - Write code to configure the MPU6050 and start data acquisition.
    - **Data Acquisition:**
        - Read acceleration and gyroscope data from the MPU6050.
    - **Tilt Detection:**
        - Implement algorithms to detect tilt based on the accelerometer and gyroscope data.
        - Trigger the LED and buzzer when a tilt beyond a safe threshold is detected.
    - **Output Data:**
        - Output detection results to a serial terminal for analysis and debugging.
4. **Upload and Test:**
    - **Uploading Code:**
        - Compile and upload the code to the STM32 board.
    - **Testing:**

        - Test the system by tilting the MPU6050 and observing the response of the LED and buzzer.
        - Verify the accuracy and responsiveness of the system.
5. **Calibrate and Experiment:**
    - **Calibration:**
        - Adjust the tilt detection threshold in the code to ensure accurate detection.
    - **Documentation:**
        - Document any adjustments made and their effects on the system's performance and reliability.

Post-Lab Questions:

1. How did you use the HAL library to communicate with the MPU6050 sensor?
2. What challenges did you encounter while setting up the tilt-based safety alert system and how did you resolve them?
3. How can tilt detection and safety alerts be applied in real-world EV applications?

Safety Note: Ensure all connections are correct and secure before powering the boards. Verify the correct orientation and pin connections of the sensors, LEDs, and other components to prevent damage to the components and the microcontroller.

REACH OUT TO US

DIYguru Mobility Pvt. Ltd. 374, MG Road, Delhi - 110030 Ph: 011-42340578 / 9910918719
E-Mail: support@diyguru.org

CENTER FOR FUTURE MOBILITY

OFFICE LOCATIONS (COEs)

Delhi (HQ) 374, MG Road, Delhi – 110030

Pune (COE) 523, Gera's Imperium Rise, Wipro Circle, Rajiv Gandhi Infotech Park, Hinjewadi Phase 2, Pune, Maharashtra 411057, India

Mumbai (COE) 5th Floor, Ark-7, Station Rd, next to Rangoli Sarees, Juhu Chandan Society, Jambli Naka, Thane West, Thane, Maharashtra 400601, India

Bangladesh: DIYguru, Level 6, Niketan, Gulshan, Dhaka 1212

Malaysia: DIYguru, No 06-01 Jalan Padan, Johar Bahru