3102 Lab Manual

Points of Lab report

1. Experiment No
2. Experiment Name
3. Introduction
4. Objective
5. Components required
6. Working principle
7. Circuit diagram and connections
8. Arduino code implementation
9. Explanation of code
10. Observations and Results
11. Conclusion
12. References

Sensors

1. Ultrasonic Distance Sensor HC-SR04
2. DHT22
3. PIR Motion sensor

Final Lab Topics

1. Write a program to write data for the **DS1307 RTC Module** using Arduino
2. Write a program to write data for the **Servo Motor** using Arduino
3. Write a program to read and write data for **Ultrasonic Distance Sensor HC-SR04** using Arduino

4. Write a program to read and write data for **DHT22 Digital Humidity and Temperature Sensor** using Arduino
5. Write a program to read and write data for the **PIR Motion sensor** using Arduino

Experiment No: 01

Experiment Name: Real-Time Clock (RTC) with Arduino and DS1307 Module

Introduction:

A Real-Time Clock (RTC) is a crucial component in embedded systems where accurate time and date are essential. The **DS1307 RTC module** is a popular choice for such applications, as it provides precise timekeeping even in the absence of power, thanks to its onboard battery. In this experiment, we use the **Arduino** board to interface with the **DS1307 RTC** module and display the time on the Serial Monitor.

Objective:

- To interface the **DS1307 RTC module** with an **Arduino**.
- To display the current time (hour, minute, second) using the **Serial Monitor**.

Components Required:

1. **Arduino Uno**
2. **DS1307 RTC Module**
3. **Jumper Wires**
4. **Breadboard (optional)**
5. **Computer with Arduino IDE**

Working Principle:

The **DS1307** RTC module communicates with the Arduino via the **I2C protocol**. The **SDA (Serial Data)** and **SCL (Serial Clock)** pins on the module are connected to the Arduino's **A4** and **A5** pins, respectively. The **DS1307** keeps track of the current time and date by utilizing a crystal oscillator and an onboard battery for power backup. This ensures that the module continues to keep time even when the system is powered off.

The Arduino will request time data from the **DS1307** at regular intervals and output the time to the **Serial Monitor**.

Circuit Diagram and Connections:

DS1307 Pin	Arduino Pin
SDA	**A4**
SCL	**A5**
VCC	**5V**
GND	**GND**

Arduino Code Implementation:

```
#include <Wire.h>
#include <RTClib.h>

RTC_DS1307 rtc;

void setup() {
  Serial.begin(9600);
  rtc.begin();
}

void loop() {
  DateTime now = rtc.now();

  // Printing time in hh:mm:ss format
  Serial.print(now.hour());
```

```
  Serial.print(':');
  Serial.print(now.minute());
  Serial.print(':');
  Serial.println(now.second());

  delay(1000); // Update every second
}
```

Explanation of Code:

1. **Include Libraries:**
   - `Wire.h`: This library is required for **I2C communication** between the Arduino and the DS1307.
   - `RTClib.h`: This library contains the functions to interface with the DS1307 RTC module.
2. **rtc.begin():**
   - Initializes the DS1307 RTC module to communicate with Arduino.
3. **rtc.now():**
   - Retrieves the current time from the RTC module.
4. **Serial.print() and Serial.println():**
   - Used to output the current time in **HH:MM:SS** format to the **Serial Monitor**.
5. **delay(1000):**
   - Pauses the code for 1 second before updating the time again.

Observations and Results:

- After uploading the code to the Arduino, the **Serial Monitor** will display the time in **HH:MM:SS** format, updating every second.
- The output will look like this:

```
makefile
CopyEdit
10:15:30
10:15:31
10:15:32
```

Conclusion:

This experiment successfully demonstrates the interfacing of an **Arduino** with a **DS1307 RTC module** to display real-time data. The module accurately keeps track of time and provides it to the Arduino, which outputs it to the Serial Monitor. This can be used as a basis for more complex time-related applications, such as alarms or time-based triggers.

References:

1. **DS1307 Datasheet** – https://www.maximintegrated.com/en/products/analog/rtc/DS1307.html
2. **RTClib Documentation** – https://github.com/adafruit/RTClib
3. **Arduino IDE** – https://www.arduino.cc/en/softwar

Experiment No: 2

Experiment Name: Writing Data for Servo Motor using Arduino

Introduction:

Servo motors are widely used in robotic arms, automation systems, and electronic projects due to their precise angular control. A servo motor can be controlled by providing a Pulse Width Modulation (PWM) signal from the Arduino.

Objective:

To interface a servo motor with Arduino and control its position by writing different angle values to it.

Components Required:

1. Arduino Uno
2. Servo Motor
3. Breadboard
4. Jumper Wires

Working Principle:

A servo motor operates by receiving PWM signals from the Arduino. The **pulse width** of the signal determines the angle to which the servo moves. The frequency of the signal remains constant, while the duty cycle determines the motor's position.

Circuit Diagram and Connections:

1. **VCC** → **5V** on Arduino
2. **GND** → **GND** on Arduino
3. **Signal Pin** → **Digital Pin 9**

Arduino Code Implementation:

```
#include <Servo.h>

Servo myServo;

void setup() {
  myServo.attach(9); // Attach servo to pin 9
}

void loop() {
  myServo.write(0);  // Move servo to 0 degrees
  delay(1000);
  myServo.write(90); // Move servo to 90 degrees
  delay(1000);
  myServo.write(180); // Move servo to 180 degrees
```

```
  delay(1000);
}
```

Explanation of Code:

- The **Servo.h** library is included to simplify servo control.
- The `myServo.attach(9);` function assigns the servo to pin 9.
- The servo motor is moved between 0, 90, and 180 degrees with 1-second intervals.

Observations and Results:

- The servo moves to different angles as programmed.
- It accurately holds each position before moving to the next.

Conclusion:

The servo motor was successfully controlled using Arduino, demonstrating precise angular movement through PWM signals.

References:

- Servo Motor Datasheet
- Arduino Official Documentation

Experiment No: 3

Experiment Name: Reading and Writing Data for Ultrasonic Distance Sensor HC-SR04 using Arduino

Introduction:

The HC-SR04 ultrasonic sensor is widely used for distance measurement in embedded systems and robotics. It uses ultrasonic waves to measure the time it takes for sound to travel to an object and back, enabling accurate distance detection. This sensor is commonly found in obstacle detection systems, automated parking solutions, and robotic navigation.

Objective:

The main objective of this experiment is to interface the HC-SR04 sensor with an Arduino board to measure the distance of an object in real-time. The measured distance will be displayed on the Serial Monitor, and we will analyze its accuracy.

Components Required:

1. Arduino Uno
2. HC-SR04 Ultrasonic Sensor
3. Breadboard
4. Jumper Wires

Working Principle:

The HC-SR04 sensor works on the principle of **ultrasonic wave reflection**. It transmits an ultrasonic pulse through its **Trigger Pin**, which propagates through the air. When the wave hits an object, it reflects back to the **Echo Pin** of the sensor. The Arduino measures the time interval between the transmitted and received pulses and calculates the distance using the formula:

Distance=Time×0.0342\text{Distance} = \frac{\text{Time} \times 0.034}{2}

Where:

- The speed of sound in air is **0.034 cm/µs**.
- The division by **2** accounts for the round-trip journey of the wave.

Circuit Diagram and Connections:

1. **VCC** → **5V** on Arduino
2. **GND** → **GND** on Arduino
3. **Trigger Pin** → **Digital Pin 9**
4. **Echo Pin** → **Digital Pin 10**

Arduino Code Implementation:

```
const int trigPin = 9;
const int echoPin = 10;

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  int distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(500);
}
```

Explanation of Code:

- The **Trigger Pin** sends a short pulse to start the measurement.
- The **Echo Pin** captures the reflected signal and calculates the time taken.
- The Arduino processes the signal and converts it into a readable distance format.

Observations and Results:

- The sensor successfully measures distances and displays them on the Serial Monitor.
- The accuracy varies based on the object's surface and environmental factors.

Conclusion:

The HC-SR04 ultrasonic sensor was successfully interfaced with Arduino, demonstrating its capability to measure distance with high precision.

References:

- HC-SR04 Datasheet
- Arduino Official Documentation

Experiment No: 4

Experiment Name: Reading and Writing Data for DHT22 using Arduino

Introduction:

The DHT22 sensor is a digital temperature and humidity sensor known for its high accuracy and stability. It is commonly used in weather monitoring systems, indoor climate control, and industrial applications.

Objective:

To interface the DHT22 sensor with Arduino, read temperature and humidity values, and display them in real-time on the Serial Monitor.

Components Required:

1. Arduino Uno
2. DHT22 Temperature & Humidity Sensor
3. Breadboard
4. Jumper Wires

Working Principle:

The DHT22 sensor contains a capacitive humidity sensor and a thermistor for temperature measurement. It communicates with the Arduino using a **one-wire protocol** to transmit digital temperature and humidity data.

Circuit Diagram and Connections:

1. **VCC** → **5V** on Arduino
2. **GND** → **GND** on Arduino
3. **Data Pin** → **Digital Pin 2**

Arduino Code Implementation:

```
#include <DHT.h>
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
}

void loop() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();

  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.print(" °C Humidity: ");
  Serial.print(hum);
  Serial.println(" %");
  delay(2000);
}
```

Explanation of Code:

- The **DHT library** simplifies data retrieval from the sensor.
- Temperature and humidity values are displayed every 2 seconds.

Observations and Results:

- The sensor provides accurate temperature and humidity readings.

Conclusion:

The DHT22 sensor was successfully integrated with Arduino, providing valuable environmental data.

References:

- DHT22 Datasheet
- Arduino Official Documentation

Experiment No: 5

Experiment Name: Reading and Writing Data for PIR Motion Sensor using Arduino

Introduction:

The PIR (Passive Infrared) sensor detects motion by measuring infrared radiation changes. It is commonly used in security systems, automatic lighting, and smart home applications.

Objective:

To interface a PIR sensor with Arduino and trigger a buzzer alarm when motion is detected.

Components Required:

1. Arduino Uno
2. PIR Motion Sensor
3. Buzzer
4. Breadboard

5. Jumper Wires

Working Principle:

The PIR sensor detects movement when an infrared source (such as a human body) moves in its field of view. It sends a **HIGH signal** to the Arduino when motion is detected.

Circuit Diagram and Connections:

1. **VCC** → **5V** on Arduino
2. **GND** → **GND** on Arduino
3. **Output Pin** → **Digital Pin 7**
4. **Buzzer** → **Digital Pin 10**

Arduino Code Implementation:

```
int pirPin = 7;
int buzzerPin = 10;

void setup() {
  pinMode(pirPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int motion = digitalRead(pirPin);
  if (motion == HIGH) {
    Serial.println("Motion Detected!");
    digitalWrite(buzzerPin, HIGH);
  } else {
    digitalWrite(buzzerPin, LOW);
  }
```

```
  delay(500);
}
```

Explanation of Code:

- The **PIR sensor** detects motion and triggers an alarm.

Observations and Results:

- Motion is successfully detected, and an alert is triggered.

Conclusion:

The PIR motion sensor was successfully interfaced with Arduino, demonstrating its capability in motion detection.

References:

- PIR Sensor Datasheet
- Arduino Official Documentation