Savitribai Phule Pune University
(Formerly University of Pune)

T.Y.B.Sc.(Computer Science)

With

Major: Computer Science

(Faculty of Science and Technology)

(For Colleges Affiliated to Savitribai Phule Pune University)

Choice Based Credit System (CBCS) Syllabus Under National Education Policy (NEP)

To be implemented from Academic Year 2026-2027

Title of the Course: B.Sc.(Computer Science)

Savitribai Phule Pune University
(Formerly University of Pune)

T.Y.B.Sc.(Computer Science) Semester V

WORKBOOK

Course Code: CS-3012-MJ-P

Lab Course on
CS-3011-MJ-T: Embedded System

WORKBOOK

T.Y.B.Sc. (Computer Science)

Subject Name- Operating Systems

Course Type: **Major Core** Course Code: CS-3012-MJ-P
Lab Course on

CS-3011-MJ-T: Embedded System

SEMESTER V

Student Name:			
College:			
Roll No:		**Exam Seat No:**	
Year:		**Division:**	

BOARD OF STUDIES

1. Dr. Patil Ranjeet
2. Dr. Joshi vinayak
3. Dr. Wani Vilas
4. Dr. Mulay Prashant
5. Dr. Sardesai Anjali
6. Dr. Shelar Madhukar
7. Dr. Bharambe Manisha
8. Dr. Deshpande Madhuri
9. Dr. Kardile Vilas
10. Dr. Korpal Ritambhara
11. Dr. Gangurde Rajendra
12. Dr. Manza Ramesh
13. Dr. Bachav Archana
14. Dr. Bhat Shridhar

Co-ordinators

- **Dr. Prashant Mulay**

Annasaheb Magar College, Hadapsar , Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

- **Dr. Sardesai Anjali**

Modern College of Arts, Science and Commerce, Shivajinagar, Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

Editor

Prof. (Dr.) Vilas B. More (Co-ordinator) Member, Faculty of Science and Technology, BOS Electronic Science, SPPU. Pune. Professor & Head, Department of Electronics, Annasaheb Magar College, Pune

Prepared by:

Dr. Bhalchandra B. Pawar (Member)	Asst. Prof. in Electronics MIT College, Alandi, Pune
Mr. Prashil D. Deshmukh (Member)	Asst. Prof. in Electronics Sinhgad College of Science, Pune
Mr. Kachru A. Ingle (Member)	Asst. Prof. in Electronics Annasaheb Magar College, Pune

CERTIFICATE

This is to certify that Mr./Miss _____

of class **T.Y.B.Sc.(C.S.)** Div____, Roll No._____ Exam. No. _____has completed satisfactorily ___Assignments/Experiments in the subject **Lab Course-CS-3012-MJ-P (Practical Course CS-3011-MJ-T: Embedded Systems)** as prescribed by Savitribai Phule Pune University, during the **academic year ……. Semester-V**.

Place :

Date: / /

Practical INCHARGE **H.O.D./Principal/Director**

Examined By	Date and Sign of Examiner
Internal Examiner:	
External Examiner:	

ABOUT THE WORK BOOK

This lab-book is intended to be used by B.Sc.(Computer Science) students for CS-3012-MJ-P: Lab Course on CS-3011-MJ-T (Embedded System), Semester V.

The objectives of this workbook are:

a) To cover the complete scope of the prescribed syllabus.
b) To bring uniformity in the conduct of the course across different colleges.
c) To facilitate the continuous assessment of students.
d) To provide students with a ready reference while performing practical experiments.
e) To provide students with hands-on experience in interfacing various sensors and actuators with Arduino, and enable them to control AC/DC loads and regulate motor speed using Arduino-based applications.

How to use this book?

This book is mandatory for the completion of the CS-3012-MJ-P: Lab Course on CS-3011-MJ-T (Embedded System). It is a measure of the performance of the student for the entire duration of the course.

Instructions to the students

a) Students should bring this workbook during all practical and laboratory sessions.
b) Printouts of the source code and program outputs are optional, unless specifically instructed by the course instructor.
c) Students should read the topics listed in the **Theory Section** of this workbook before performing the corresponding practical experiments/assignments.
d) Student has to perform any 3 experiments from each group (A, B, C & D) of section-I and any one activity from section-II.
e) Students should complete the exercises assigned by the subject teacher or practical instructor as part of the journal work. However, they are encouraged to solve additional exercises for further practice and better understanding.
f) Each assignment/experiment will be evaluated on a **5-point scale (0–5)** according to the assessment criteria specified in this workbook.

i.	Not done	0
ii.	Incomplete	1
iii.	Late Complete	2
iv.	Needs improvement	3
v.	Complete	4
vi.	Well Done	5

Instructions to the Instructors:

1) Ensure that students follow the instructions provided in this manual while performing the practical experiments.
2) Instructors should use the programs provided in the workbook to demonstrate the practical alongside the relevant theoretical concepts.
3) After a student successfully completes a program, the instructor should encourage the student to modify or enhance the program according to the instructions (mark with *) given in the manual.
4) While introducing the hardware required for each experiment, greater emphasis should be placed on developing programming skills, logical thinking, and problem-solving abilities.
5) Verify each student's program and sign in the space provided after the successful completion of the activity.
6) Evaluate each assignment/experiment on a **5-point scale** as specified in the assessment criteria by marking the appropriate box.
7) Record the awarded marks on the **Assignment/Experiment Completion Page** of the respective laboratory course.
8) Students should be encouraged to use the actual hardware while performing the practical experiments to gain hands-on experience.
9) The college may use **any suitable Arduino-compatible development board or circuit board** for conducting practical demonstrations, provided it meets the objectives of the experiment.

List of Experiments

Student Name: ……..……..

Roll No. :…… **Batch:** …… **Division:** ……....

Expt. No.	Name of Experiment	Page No.	Perform Date	Remark	Signature of Practical Incharge
SECTION – I, Group-A: Basics of Embedded Systems & Arduino (Any 3)					
1	Study of Arduino UNO board: pin configuration, power supply, onboard components, Installation and familiarization of Arduino IDE; writing and executing first sketch (on board led Blink program) **(Compulsory)**				
2	Digital output interfacing: LED blinking using delay and loop constructs.				
3	Interfacing of LEDs with Arduino for pattern generation.				
4	Digital input interfacing: Multiple Switch/button interfacing with Multiple LEDs control				
SECTION – I, Group-B: Analog I/O and Timing (Any 3)					
5	Analog input interfacing using potentiometer and displaying values on Serial Monitor.				
6	PWM-based LED brightness control using analogWrite()				
7	Display of real time on serial monitor with Interfacing to an External Real-Time Clock (RTC)				
8	Interrupt-based input handling using external interrupts.				
SECTION – I, Group-C: Sensor Interfacing (Any 3)					
9	Interfacing temperature sensor (LM35 / DHT11) and displaying values on Display.				
10	Interfacing light sensor (LDR) for automatic light control system.				
11	Interfacing ultrasonic / IR sensor for Displacement measurement.				
12	Interfacing of speakers / buzzers to generate different tones.				

SECTION – I, Group-D: Actuators and Communication (Any 3)					
13	Interfacing relay module for AC/DC load control (demonstration-level).				
14	Interfacing DC motor using motor driver (L293D / L298) to control speed using Potentiometer.				
15	Controlling an RGB LED Using the BlinkM Module (I2C communication).				
16	Serial communication using USB: data transmission between Arduino and PC.				
SECTION - II: ACTIVITY (Any *ONE* equivalent to 3 experiments)					
1. Electronics Hobby Project – Design and develop a small electronics-based project and submit a report. 2. Field Visit Report – Visit an electronics industry and submit a detailed report.					

SECTION - I

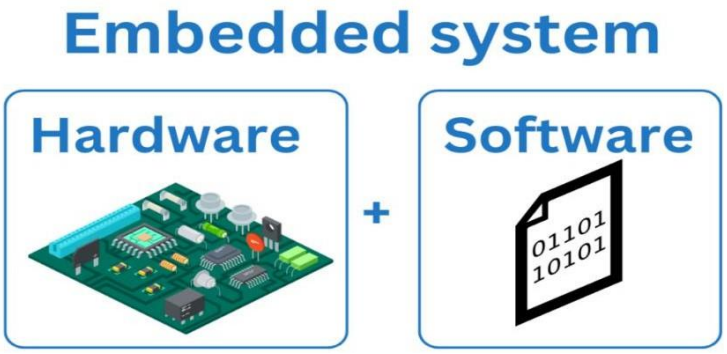
Group-A

Basics of Embedded Systems & Arduino

"About Embedded System" (Total Slots = 1)

What is an Embedded System?

An embedded system is a dedicated computer-based system designed to perform one or more specific functions within an electronic device. **It combines hardware and software components** to control, monitor, and operate various applications efficiently and reliably.



Unlike general-purpose computers, embedded systems are designed for specialized tasks and are commonly integrated into machines, appliances, industrial equipment, and automation systems.

Components of an Embedded System:

Hardware Components

An embedded system typically consists of:

- Microcontroller or microprocessor
- Memory units (RAM, ROM, Flash)
- Input devices such as sensors, switches, and keypads
- Output devices such as LEDs, displays, relays, and motors
- Communication interfaces

Software Components

The software controls the operation of the embedded hardware. Programs are usually developed using programming languages such as C, C++, or Python depending on the application requirements.

Working Principle of an Embedded System:

The embedded system receives input signals from connected devices or sensors, processes the information using programmed instructions, and generates the required output. The entire process is performed automatically and continuously to achieve accurate and reliable system operation.

Features of an Embedded Systems:

- Dedicated functionality
- Compact and lightweight design
- Fast response time
- Low power consumption
- Reliable and stable operation
- Real-time processing capability

Applications of an Embedded System:

Embedded systems are widely used in:

- Home appliances
- Industrial automation
- Automotive electronics
- Medical equipment
- Consumer electronics
- Communication systems
- Smart devices and IoT applications

Advantages of an Embedded System:

- High efficiency
- Reduced operational cost
- Compact size
- Improved reliability
- Low maintenance requirements
- Energy-efficient operation

Safety and Usage Guidelines

- Operate the system only within specified voltage limits.
- Avoid exposure to moisture, dust, and extreme temperatures.
- Ensure proper power supply connections before operation.

- Do not modify hardware or software without authorization.
- Follow recommended maintenance procedures for reliable performance.

Conclusion

Embedded systems are essential components of modern electronic and automation technologies. Their ability to perform dedicated tasks with speed, accuracy, and reliability makes them suitable for a wide range of industrial, commercial, and consumer applications.

"The Microcontroller used in Embedded System"

Introduction

A microcontroller is the core component of an embedded system. It is a compact integrated circuit designed to perform specific control operations in electronic devices. A microcontroller combines a processor, memory, and input/output peripherals on a single chip.

Microcontrollers are widely used in automation systems, consumer electronics, medical devices, automotive systems, and industrial applications.

Main Components of a Microcontroller

A typical microcontroller consists of:

- Central Processing Unit (CPU)
- Random Access Memory (RAM)
- Read Only Memory (ROM/Flash)
- Input/Output (I/O) Ports
- Timers and Counters
- Communication Interfaces such as UART, SPI, and I2C

Functions of a Microcontroller

The microcontroller performs the following functions:

- Reads input signals from sensors or switches
- Processes data according to programmed instructions
- Controls output devices such as LEDs, motors, and displays
- Communicates with external devices and systems

Commonly Used Microcontrollers

Some popular microcontrollers used in embedded systems include:

- 8051 Microcontroller
- PIC Microcontroller

- AVR Microcontroller
- ARM Cortex Microcontroller
- ESP32 and ESP8266
- **Arduino-based controllers**

Group-A	Experiment No.-1 (Total Slots = 1)	
Title:	Study of Arduino UNO board: pin configuration, power supply, onboard components, Installation and familiarization of Arduino IDE; writing and executing first sketch (on board led Blink program)(Compulsory)	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

"Study of Arduino UNO board"

Introduction to Arduino

Arduino is an open-source electronic development platform widely used for embedded systems, automation, robotics, and Internet of Things (IoT) applications. An Arduino board consists of a programmable microcontroller along with input/output pins that can interface with sensors, displays, motors, and other electronic devices.

Arduino boards are popular because they are easy to program, cost-effective, and suitable for beginners as well as advanced developers.

Main Components of an Arduino Board

An Arduino board typically includes:

- Microcontroller
- Digital Input/Output Pins
- Analog Input Pins
- USB Interface
- Power Supply Connector
- Crystal Oscillator
- Reset Button
- Voltage Regulator

Working Principle

The Arduino board reads input signals from connected devices such as sensors or switches, processes the data using programmed instructions, and controls output devices like LEDs, motors, displays, or relays.

Programs for Arduino are written using the Arduino IDE and uploaded to the board through a USB connection.

Common Arduino Boards

Commonly used Arduino boards include:

- Arduino Uno
- Arduino Nano

- Arduino Mega
- Arduino Leonardo
- Arduino Due

Features

- Easy programming using Arduino IDE
- Open-source hardware and software
- Multiple digital and analog pins
- USB communication support
- PWM and serial communication capability
- Low power consumption

Applications

Arduino boards are widely used in:

- Embedded system development
- Robotics projects
- Home automation
- IoT applications
- Industrial monitoring systems
- Sensor interfacing projects

Advantages

- User-friendly programming environment
- Low-cost development platform
- Large community support
- Easy hardware interfacing
- Suitable for prototyping and learning

Basic Block Diagram

The Arduino Uno is based on the ATmega328P microcontroller and provides a total of 20 input/output pins, including 14 digital pins and 6 analog input pins. These pins enable the Arduino to interface with various sensors, actuators, displays, communication modules, and other electronic devices.

Pin Configuration:

Pin	Type	Description
D0 (RX)	Digital	UART Serial Receive
D1 (TX)	Digital	UART Serial Transmit
D2	Digital	Digital I/O, External Interrupt (INT0)
D3	Digital/PWM	PWM Output, External Interrupt (INT1)
D4	Digital	Digital Input/Output
D5	Digital/PWM	PWM Output
D6	Digital/PWM	PWM Output
D7	Digital	Digital Input/Output
D8	Digital	Digital Input/Output
D9	Digital/PWM	PWM Output
D10	Digital/PWM	PWM Output, SPI (SS)
D11	Digital/PWM	PWM Output, SPI (MOSI)
D12	Digital	SPI (MISO)
D13	Digital	SPI (SCK), Built-in LED
A0	Analog	Analog Input 0
A1	Analog	Analog Input 1
A2	Analog	Analog Input 2
A3	Analog	Analog Input 3
A4	Analog	Analog Input, I²C (SDA)
A5	Analog	Analog Input, I²C (SCL)

Power Pins:

Pin	Function
VIN	External power input (7–12 V recommended)
5V	Regulated 5 V output
3.3V	Regulated 3.3 V output
GND	Ground (0 V)
RESET	Resets the microcontroller
IOREF	Reference voltage for shields

Digital Input/Output Pins

- The Arduino Uno has **14 digital pins (D0–D13)**.
- Each pin can be configured as an **INPUT** or **OUTPUT** using the pinMode() function.
- A digital pin can output **HIGH (5 V)** or **LOW (0 V)** using digitalWrite().
- Digital input is read using the digitalRead() function.

Analog Input Pins

- The Arduino Uno provides **6 analog input pins (A0–A5)**.
- These pins are connected to a **10-bit Analog-to-Digital Converter (ADC)**.
- The ADC converts analog voltages (0–5 V) into digital values ranging from **0 to 1023**.
- Analog values are read using the analogRead() function.

PWM Pins

The PWM (Pulse Width Modulation) pins are: **D3, D5, D6, D9, D10, and D11**

These pins generate PWM signals using the analogWrite() function and are commonly used for:

- LED brightness control
- DC motor speed control
- Servo motor applications
- Power control

Communication Pins

UART Communication

- **D0 (RX)** – Receive data
- **D1 (TX)** – Transmit data

I²C Communication

- **A4** – SDA (Serial Data)
- **A5** – SCL (Serial Clock)

SPI Communication

- **D10** – SS (Slave Select)
- **D11** – MOSI (Master Out Slave In)
- **D12** – MISO (Master In Slave Out)
- **D13** – SCK (Serial Clock)

External Interrupt Pins

Arduino Uno supports two external interrupt pins:

- **D2 → INT0**
- **D3 → INT1**

These pins are used to respond immediately to external events such as switch presses or sensor signals.

Built-in LED

- The Arduino Uno has a built-in LED connected to **Digital Pin 13 (D13)**.
- It is commonly used for testing programs such as the **Blink** example.

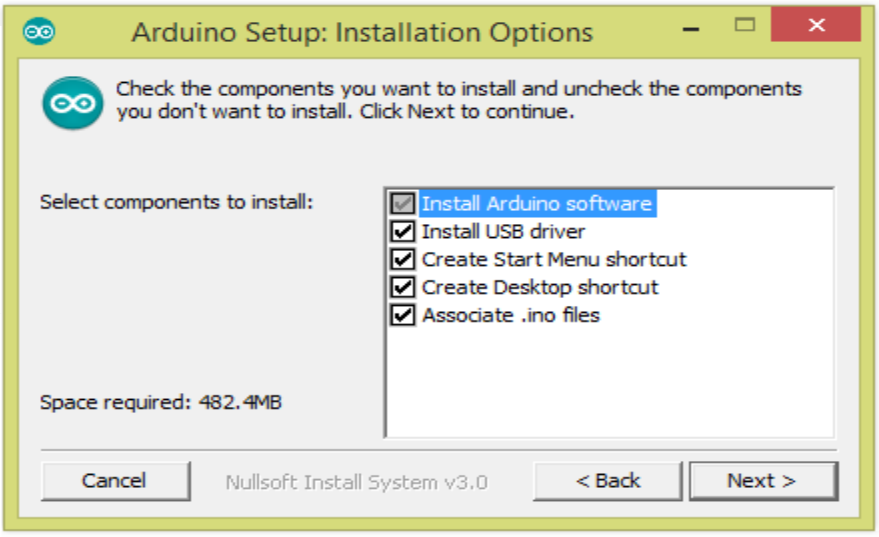
Installation and familiarization Arduino IDE

Steps to Installation of Arduino IDE software

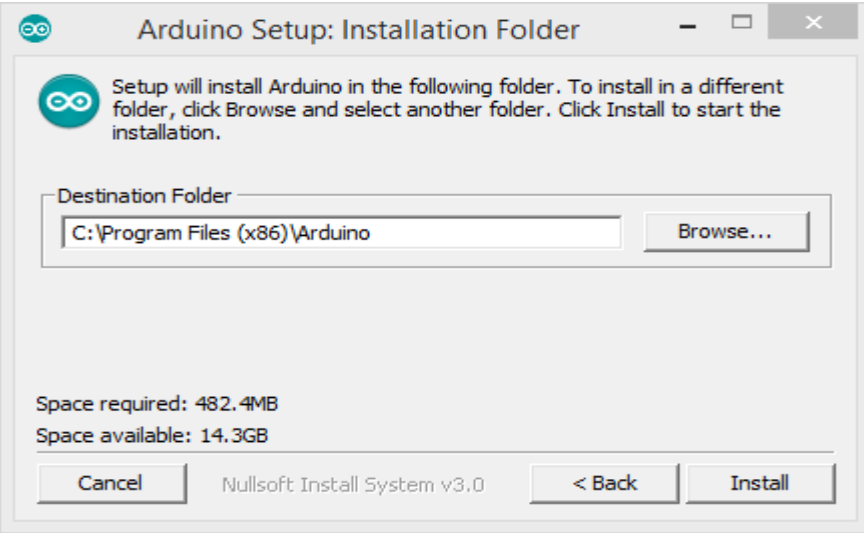
Download Link: https://www.arduino.cc/en/Main/Software

1. Double Click on arduino.exe file.
2. Then click on “I Agree”.

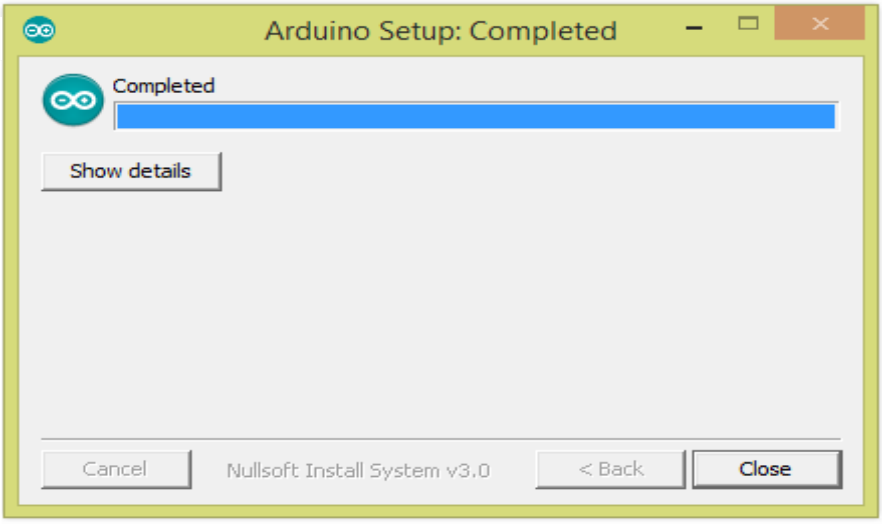
3. Without changing anything click on “Next >”.



4. Select the destination folder & click on “Install”.



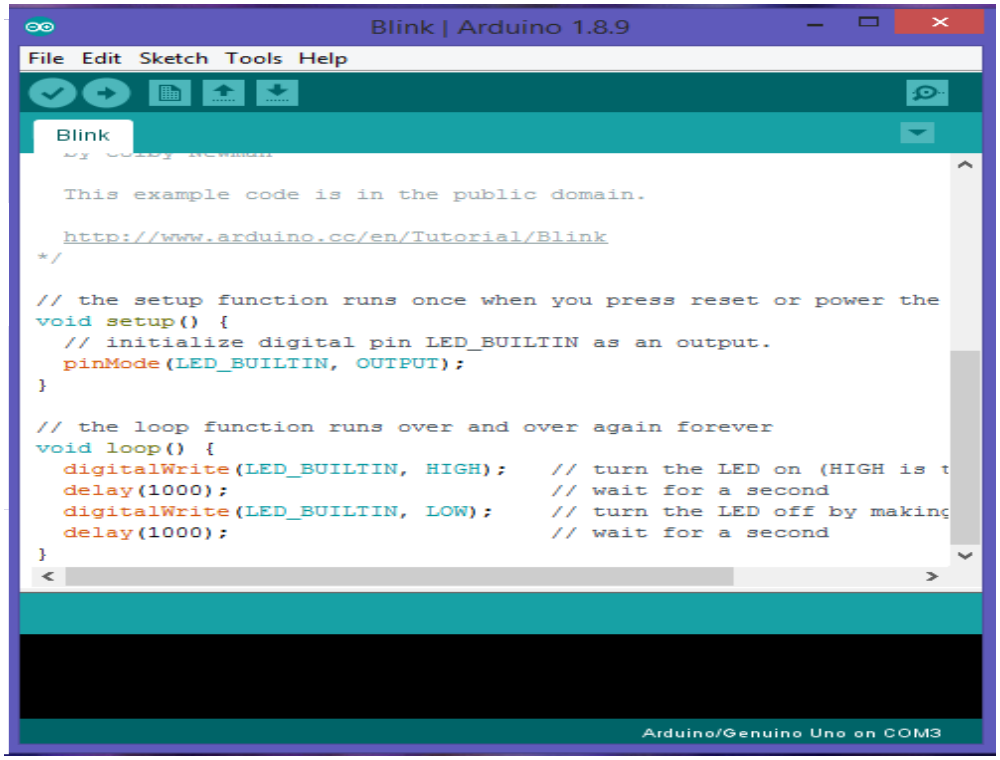
5. After installation completed click on “Close”.



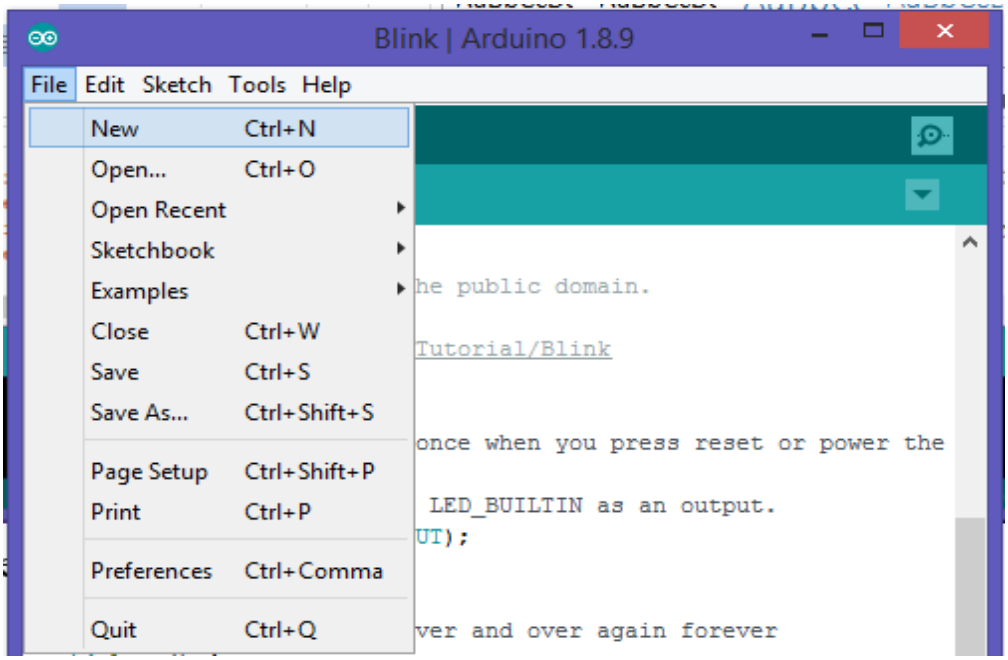
****

Steps to use of Arduino IDE software

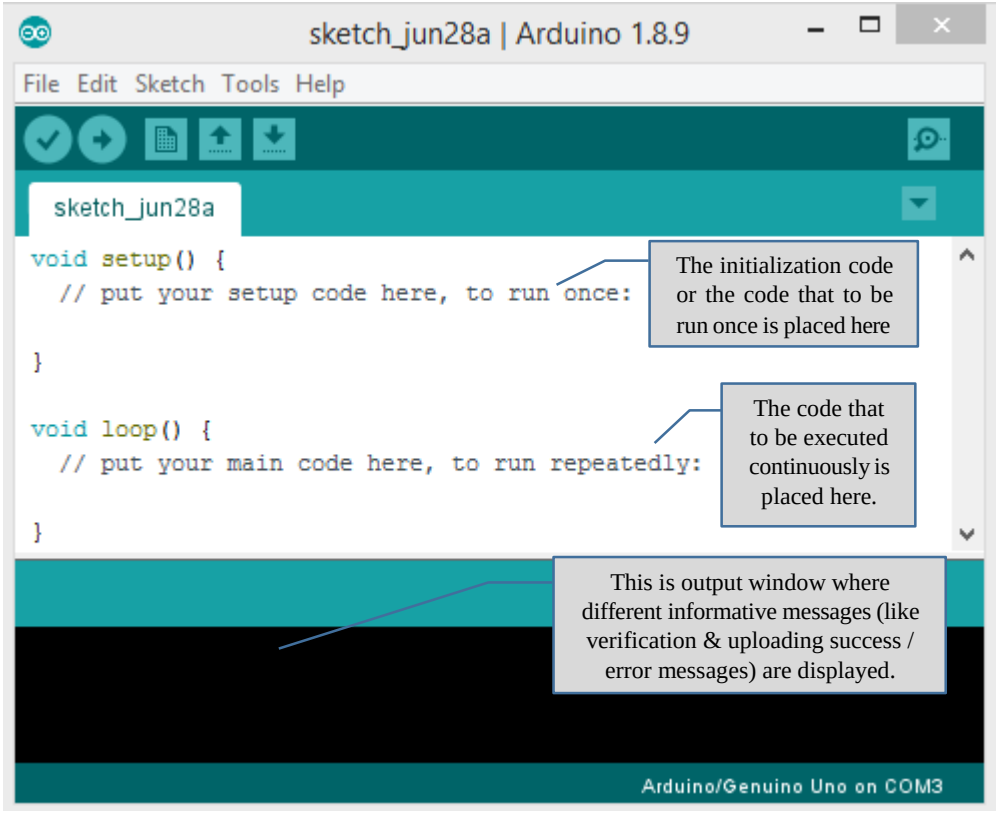
1. Double Click on “Arduino” icon.
2. The Arduino IDE will be opened with previously open project.



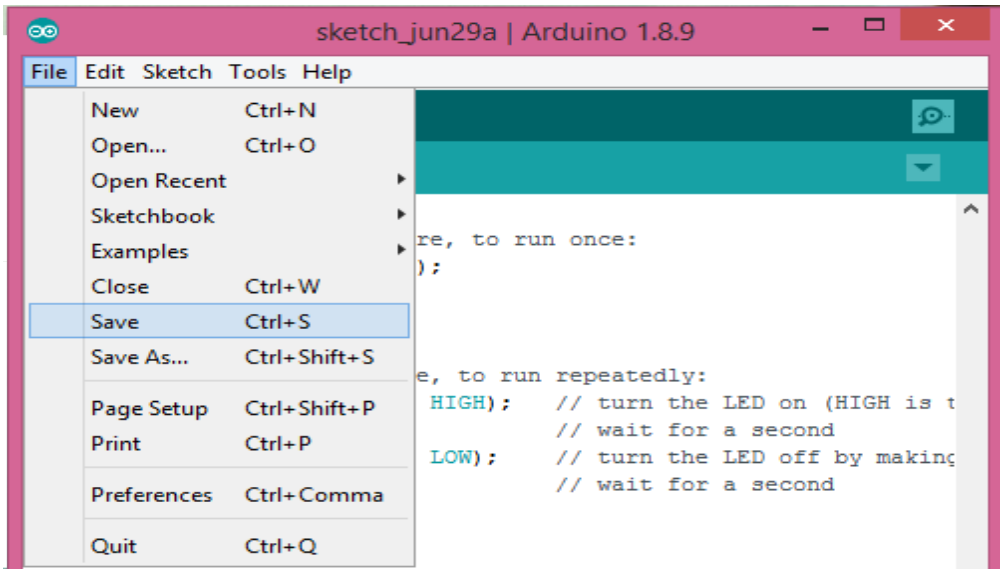
3. For new project go to File -> New.



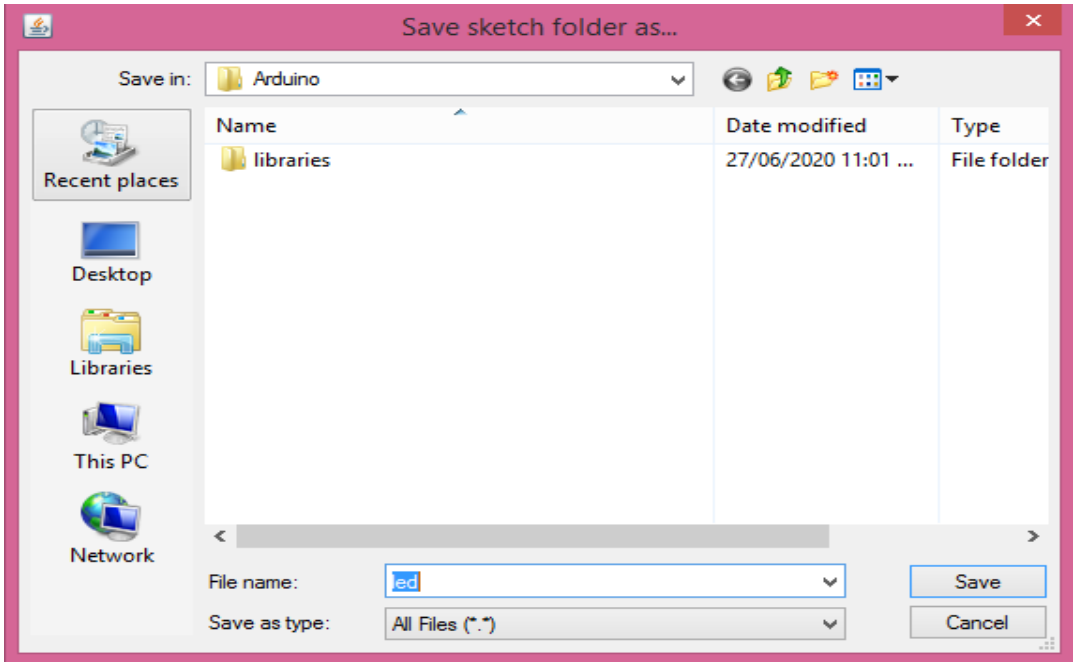
4. New Project is open.
5. It contains two main sections. setup() & loop().



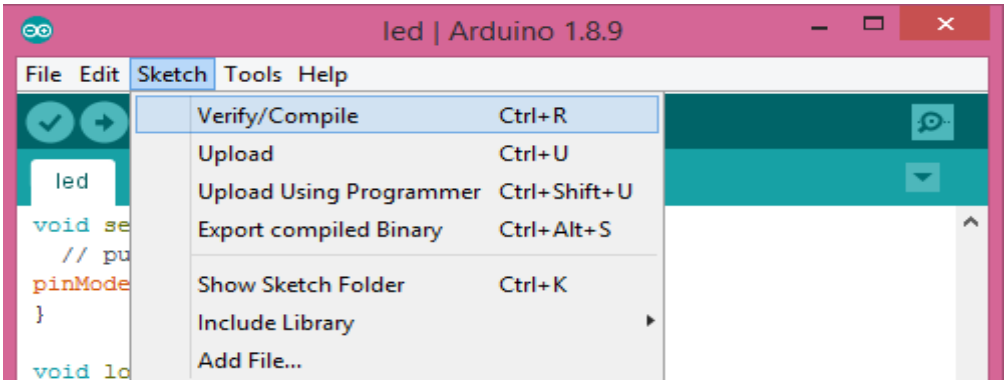
6. After writing your program save it, go to File -> Save.



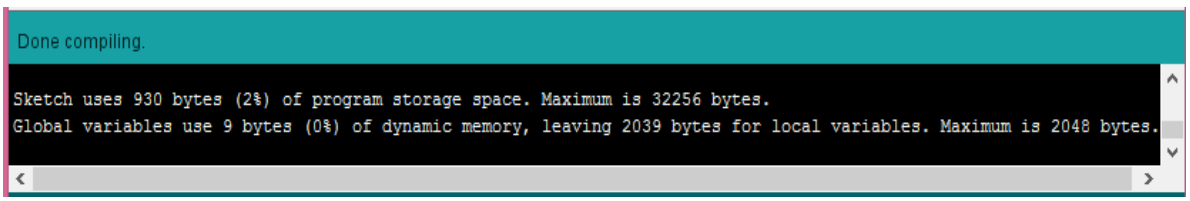
7. Select the folder where you want to save your file (if don't want to save in default Arduino folder), then give the appropriate name to the file and click on "save".



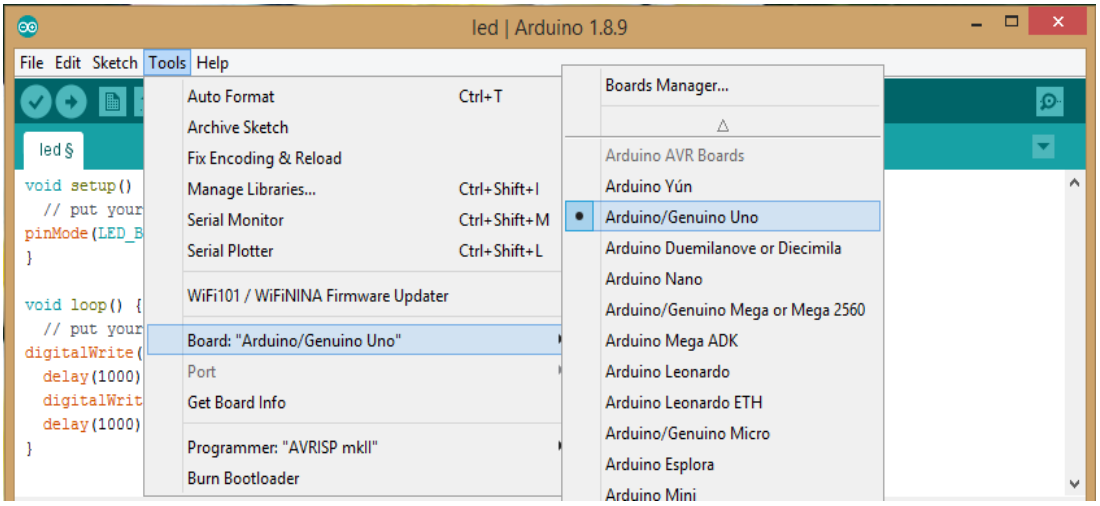
8. Then to compile the written program, select Sketch -> Verify / Compile option. **OR** click on "Verify" button.



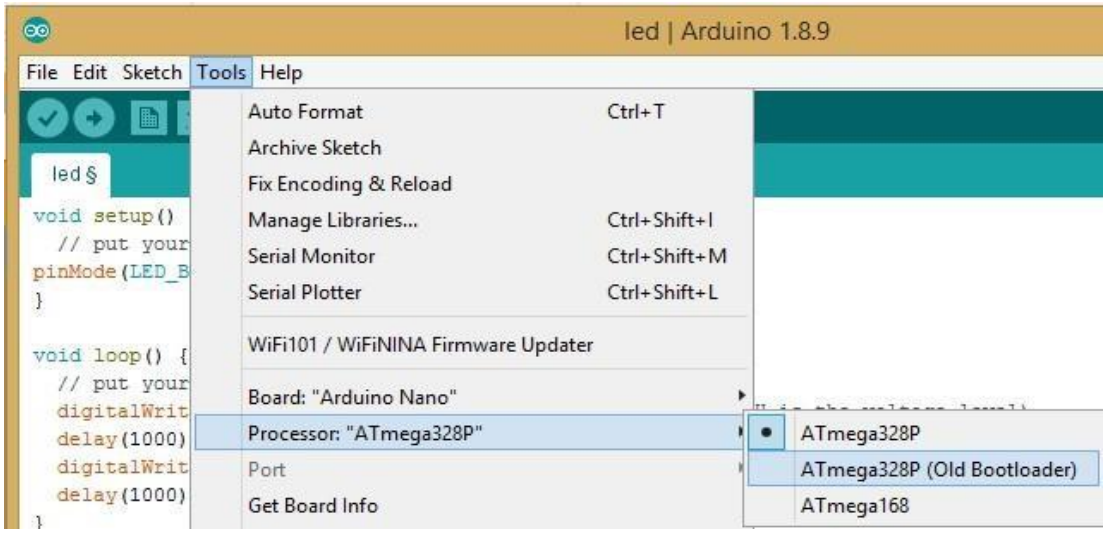
9. After compilation is done, output window display message "Done Compiling" (if there are no errors in a program). Also displays information about how much program & data memory is used by the program.



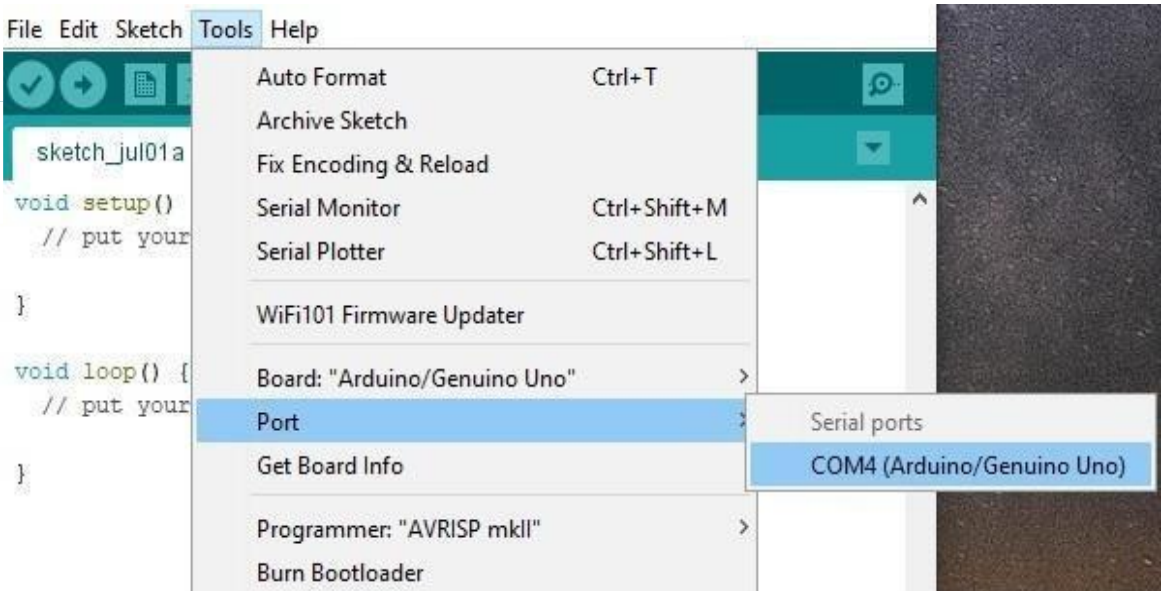
10. Then select the desired Arduino board, go to Tools -> Board: -> Arduino / Genuino Uno (if Uno is used) or select Arduino Nano (if nano is used).



11. For Arduino Nano, there is option for selection of Processor. This has three choices user can select one as per the processor & Bootloader used in the available Nano board.



12. After selecting the board, select the COM port where selected board is connected. Go to the Tools -> Port.

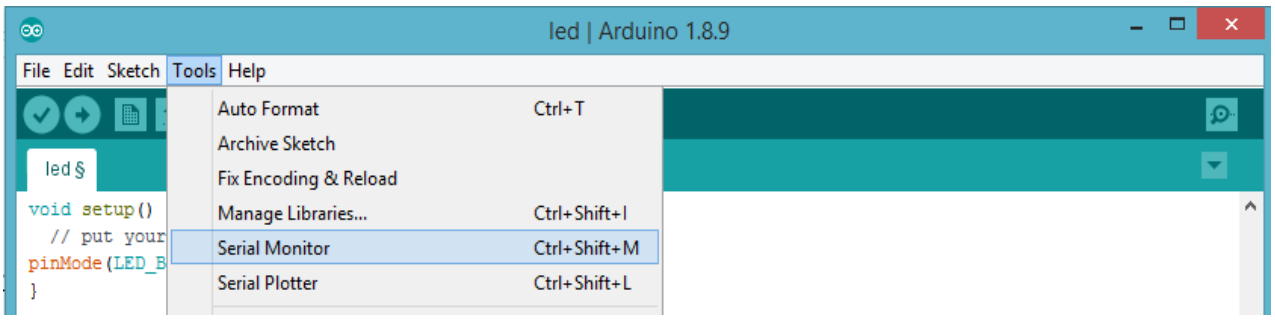


13. Once board & COM port is selected, upload the program on to the connected board. Go to Sketch -

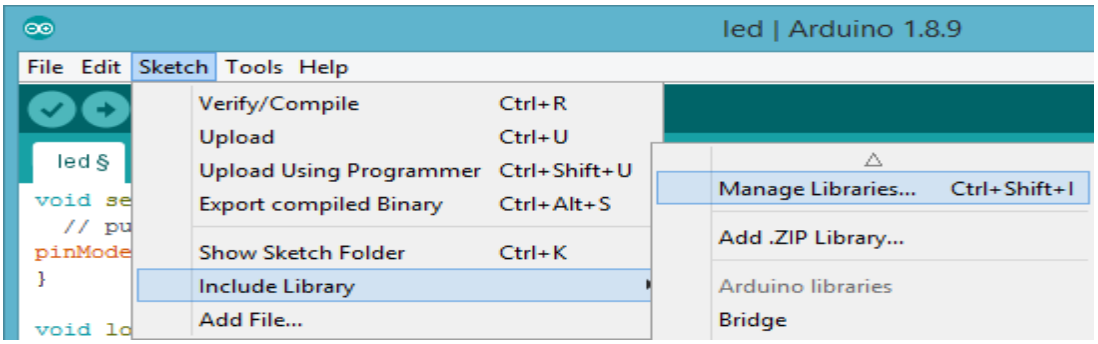
    > Upload. (Arduino has on board programmer, hence no need of external programmer.)

14. Perform necessary connections to the Arduino and observed the output. In order to execute program again from start press Reset button on the Arduino board.

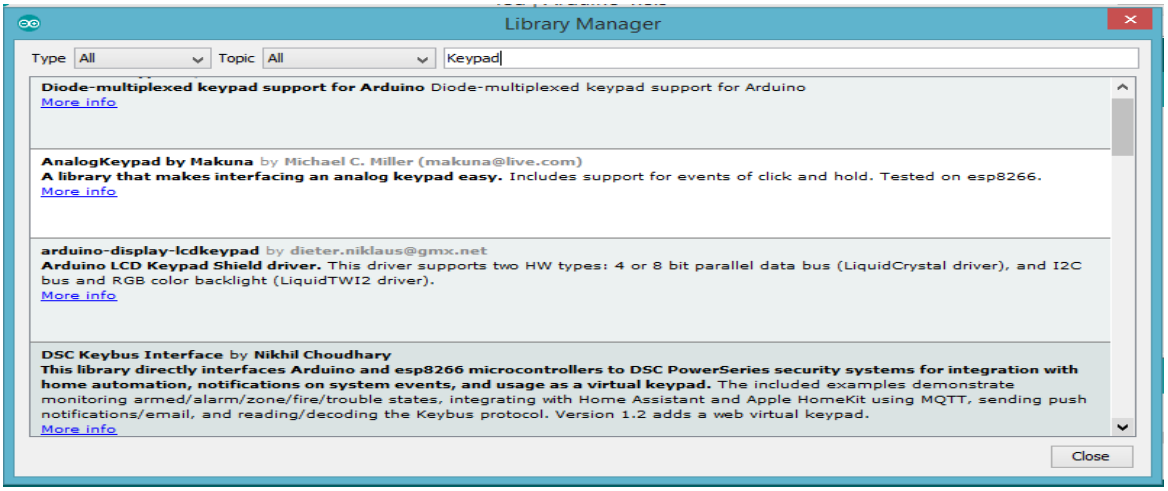
15. If you want to use serial monitor, go to Tools -> Serial Monitor. (Serial monitor displays the data coming from serial port of the Arduino).



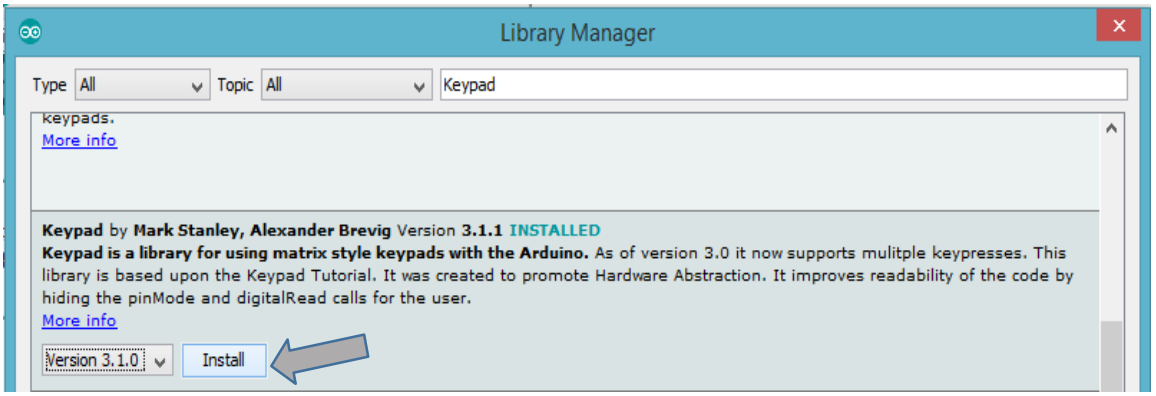
16. If library for any component (sensor / shield) is not available in the Arduino IDE, it can be added externally. Go to Sketch -> Include Library -> Manage Libraries… (Requires internet connection)



17. The new window will open, where you can search the library you want. E.g. type the name keypad for keypad library, as follows:

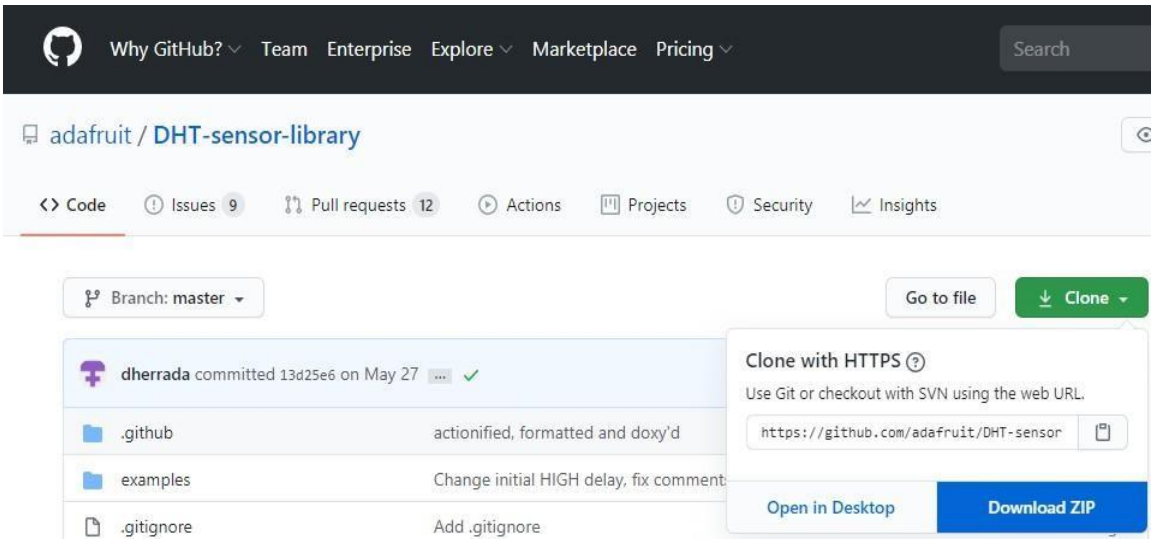


18. Select the desired Keypad library, select version and click on install.

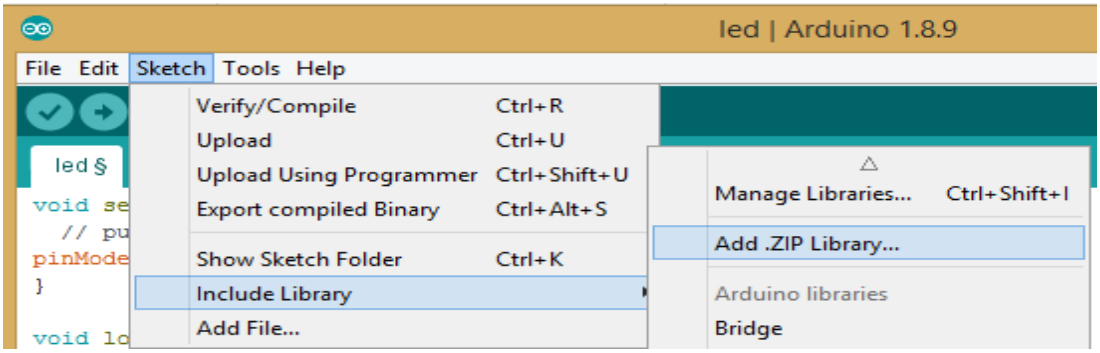


19. If the desired library is not found here, then there is another option to add library. Search the library in the GitHub. E.g. search for DHT11 sensor library.

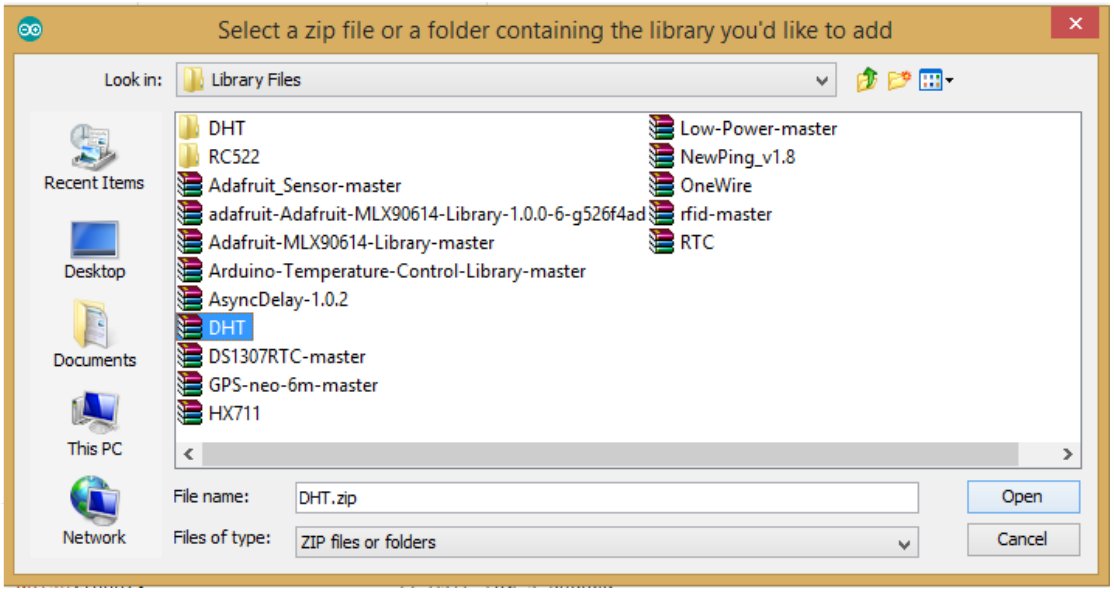
20. In the GitHub website select “Download ZIP” option.



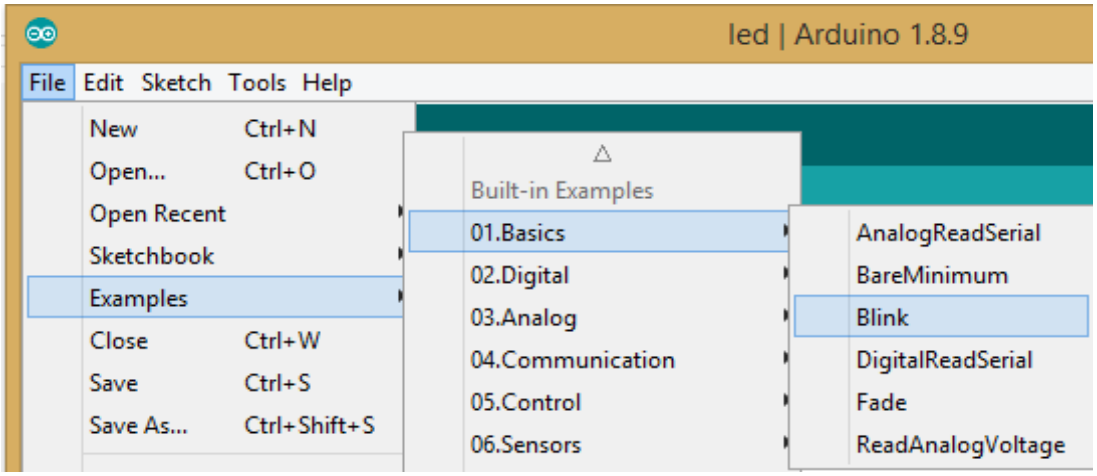
21. Then go to Arduino IDE, select Sketch -> Include Library -> Add .ZIP Library…



22. Select the downloaded ZIP file and click on “Open”.



23. You can go through the in-build examples available in the Arduino IDE and try them on actual Arduino board as per your interest.



****

***try your first sketch on Arduino IDE**

Group-A	Experiment No.-2 (Total Slots = 1)	
Title:	Digital output interfacing: LED blinking using delay and loop constructs	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim: To interface an LED with a microcontroller and generate a blinking effect using digital output, delay() function, and loop() construct.

Objectives:

- To understand digital output interfacing.
- To learn the use of output pins in a microcontroller.
- To study the working of delay() and loop() functions.
- To control an LED using a program.

Components Required:

- Arduino Uno/Nano
- LED
- 220Ω resistor
- Breadboard
- Jumper wires
- USB cable
- PC/Laptop

Theory

Digital output interfacing is the process of connecting output devices such as LEDs, buzzers, and relays to a microcontroller.

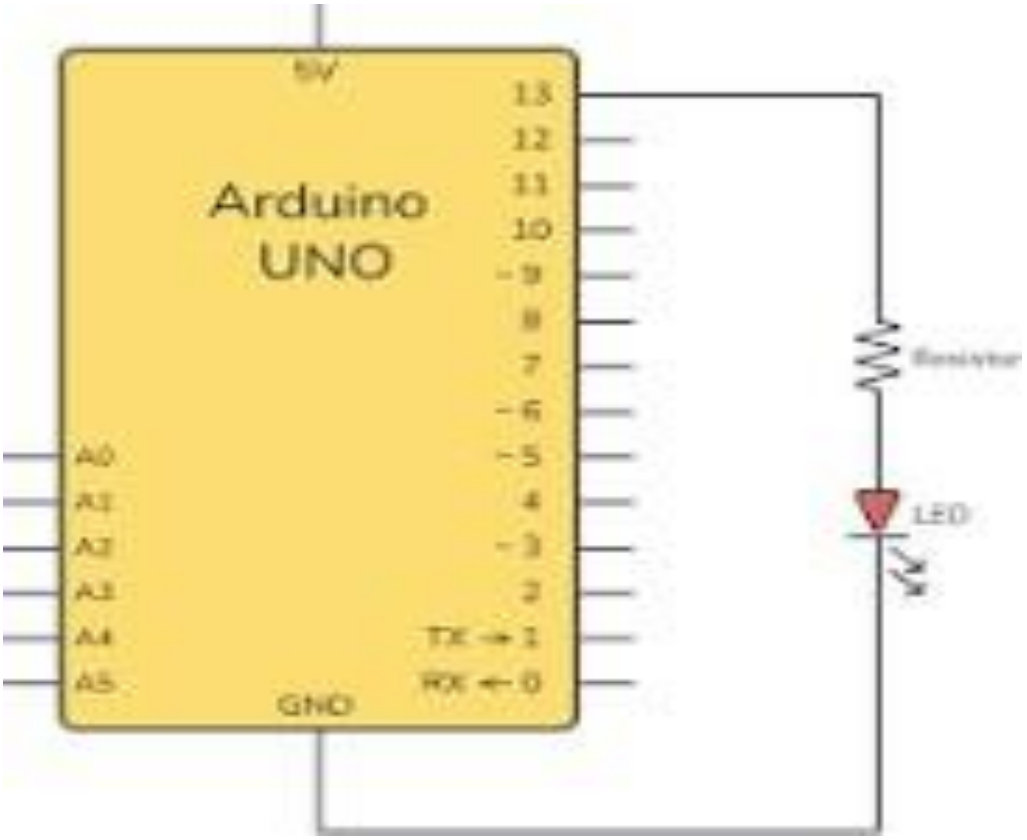
An LED (Light Emitting Diode) glows when current passes through it. The microcontroller sends HIGH or LOW signals through its digital output pins to control the LED.

In this experiment:

- HIGH turns the LED ON.
- LOW turns the LED OFF.
- delay() creates a time gap.
- loop() repeats the blinking continuously.

Algorithm

1. Start the program.
2. Set pin 13 as OUTPUT.
3. Turn the LED ON.
4. Wait for 1 second.
5. Turn the LED OFF.
6. Wait for 1 second.
7. Repeat steps 3–6 continuously.



Key Concepts :

Function	Purpose
`pinMode()`	Configure pin direction
`digitalWrite()`	Send HIGH or LOW signal
`delay()`	Pause execution
`loop()`	Repeat instructions continuously

Program:

```
// LED Blinking Program

void setup() {
  pinMode(13, OUTPUT);   // Configure pin 13 as output
}

void loop() {
  digitalWrite(13, HIGH); // Turn LED ON
  delay(1000);            // Wait for 1 second

  digitalWrite(13, LOW);  // Turn LED OFF
  delay(1000);            // Wait for 1 second
}
```

Result:
The LED was successfully interfaced with the microcontroller and blinked continuously using delay and loop constructs.

***Modify the program by connecting LED to different digital pin and with different delay. And note down the observation.**

Group-A	Experiment No.-3 (Total Slots = 1)	
Title:	Interfacing of LEDs with Arduino for pattern generation.	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim:
To interface multiple LEDs with an Arduino board and generate various blinking patterns using digital output programming.

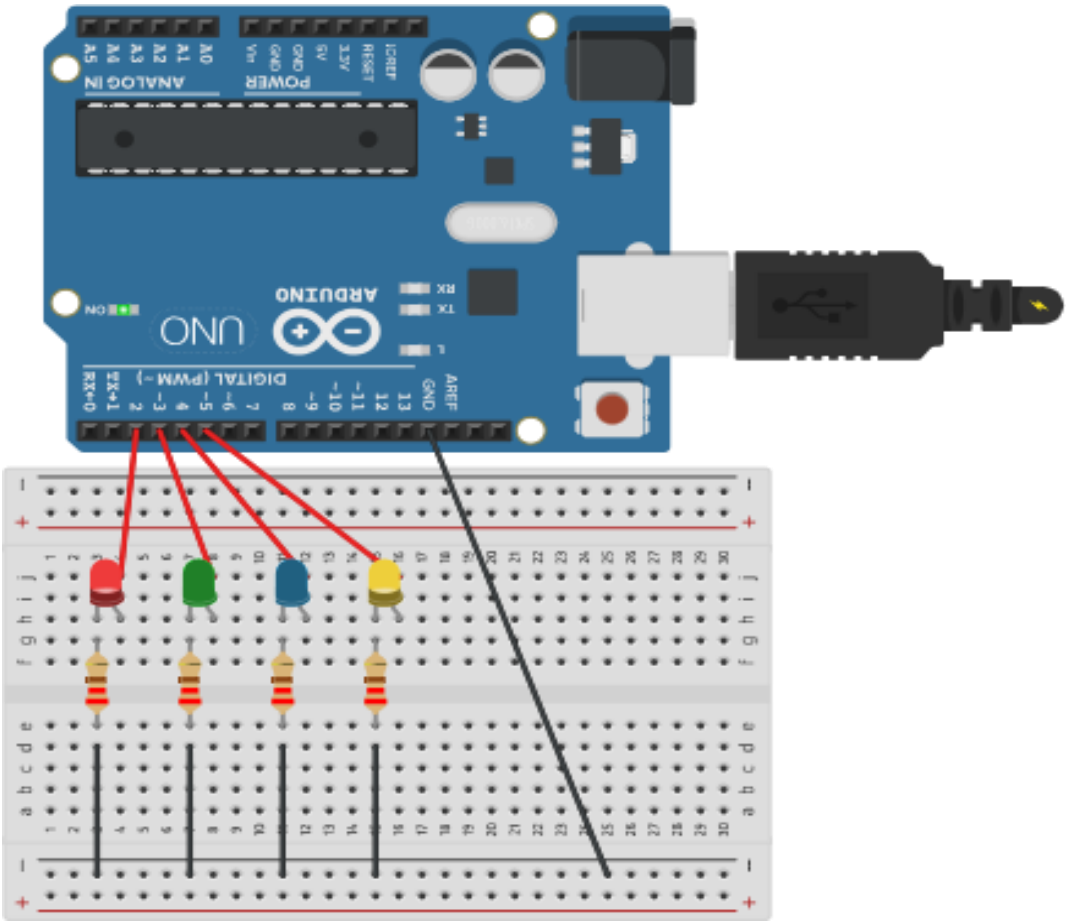
Objectives:
- To understand LED interfacing with Arduino.
- To generate different LED patterns using programming.
- To study digital output control.
- To learn sequential and alternate LED blinking techniques.

Components Required:
- Arduino Uno/Nano
- LEDs 4
- 220Ω resistor 4
- Breadboard
- Jumper wires
- USB cable
- PC/Laptop

Theory
LED pattern generation is a digital output interfacing technique in which multiple LEDs are controlled in a specific sequence to create visual patterns.
The Arduino board sends HIGH and LOW signals through digital pins:
- HIGH → LED ON
- LOW → LED OFF

Using loops and delays, different lighting patterns can be generated such as:
- Sequential blinking
- Alternate blinking
- Running light pattern
- Flashing pattern

This experiment demonstrates how multiple output devices can be controlled simultaneously.



Algorithm

1. Start the program.
2. Configure pins 2–5 as OUTPUT.
3. Turn LEDs ON one by one.
4. Add delay between each operation.
5. Turn LEDs OFF one by one.
6. Repeat the sequence continuously.

Program:

```
// LED Pattern Generation using Arduino

int led1 = 2;
int led2 = 3;
int led3 = 4;
int led4 = 5;

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}

void loop() {

  // Sequential ON pattern
  digitalWrite(led1, HIGH);
  delay(300);

  digitalWrite(led2, HIGH);
  delay(300);

  digitalWrite(led3, HIGH);
  delay(300);

  digitalWrite(led4, HIGH);
  delay(300);

  // Sequential OFF pattern
  digitalWrite(led1, LOW);
  delay(300);

  digitalWrite(led2, LOW);
  delay(300);

  digitalWrite(led3, LOW);
  delay(300);

  digitalWrite(led4, LOW);
  delay(300);
}
```

Result

The LEDs were successfully interfaced with the Arduino board and different lighting patterns were generated using digital output programming.

*Modify the program for different LED's pattern.

Group-A	Experiment No.-4 (Total Slots = 1)	
Title:	Digital input interfacing: Multiple Switch/button interfacing with Multiple LEDs control	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim:

To interface multiple switches/buttons with a microcontroller and control multiple LEDs using digital input and output interfacing.

Objectives:

- To understand push buttons interfacing with Arduino.
- To understand the status of switch.
- To study LED control by accepting digital input.

Components Required:

- Arduino Uno/Nano
- Push Buttons-3
- LEDs -3
- 220Ω resistor -3
- 10KΩ resistor-3
- Breadboard
- Jumper wires
- USB cable
- PC/Laptop

Theory:

Digital input interfacing is used to read the status of switches or buttons connected to a microcontroller. The microcontroller checks whether the switch is pressed (HIGH) or released (LOW) and controls the corresponding LEDs.

In this experiment:

- Each push button acts as a digital input device.
- Each LED acts as a digital output device.
- When a button is pressed, the corresponding LED glows.
- When the button is released, the LED turns OFF.

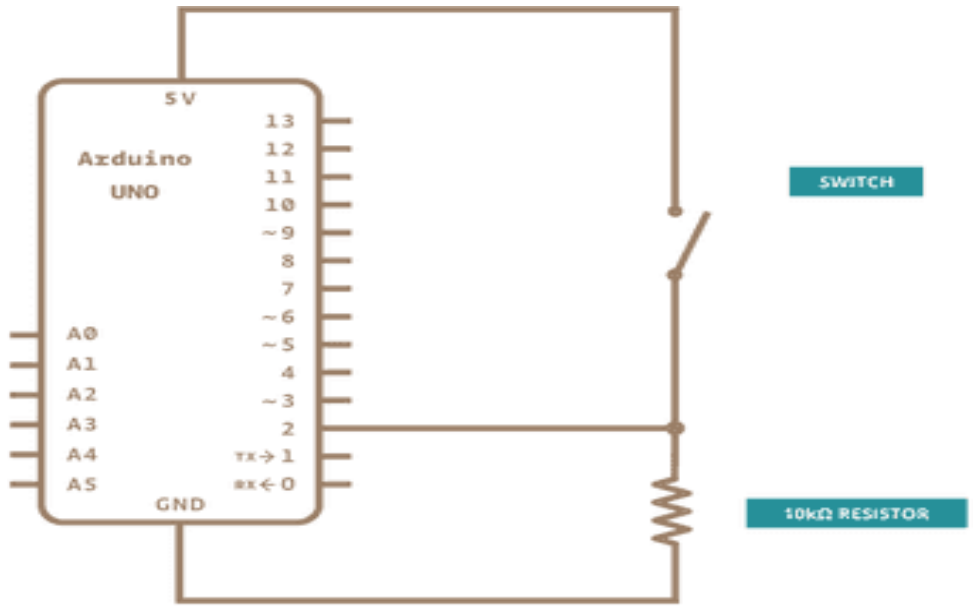
LED Connection

- LED anode → resistor → Arduino output pin
- LED cathode → GND

Switch Connection

- One terminal → +5V
- Other terminal → input pin and pull-down resistor to GND

Program-1: A simple program to interface switch with Arduino and display status of switch pressed.




```
int switchPin = 2;

void setup()
{
    pinMode(switchPin, INPUT);

    Serial.begin(9600);
}

void loop()
{
    int switchState = digitalRead(switchPin);

    if(switchState == HIGH)
    {
        Serial.println("Switch is Pressed");
    }
    else
    {
        Serial.println("Switch is Released");
    }

    delay(500);
}
```

Program-2: To interface multiple switch and LEDs with Arduino.

Connections required:

Connect Switch-1 to pin no. 2
Switch-2 to pin no. 3
Switch-3 to pin no. 4
LED-1 to pin no. 8
LED-2 to pin no. 9
LED-3 to pin no. 10

```
int sw1 = 2;
int sw2 = 3;
int sw3 = 4;

int led1 = 8;
int led2 = 9;
int led3 = 10;

void setup()
{
    pinMode(sw1, INPUT);
    pinMode(sw2, INPUT);
    pinMode(sw3, INPUT);

    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
    pinMode(led3, OUTPUT);
}

void loop()
{
    if(digitalRead(sw1) == HIGH)
        digitalWrite(led1, HIGH);
    else
        digitalWrite(led1, LOW);

    if(digitalRead(sw2) == HIGH)
        digitalWrite(led2, HIGH);
    else
        digitalWrite(led2, LOW);

    if(digitalRead(sw3) == HIGH)
        digitalWrite(led3, HIGH);
    else
        digitalWrite(led3, LOW);
}
```

***Modify the program to read the status of multiple switches and display the message also control the corresponding LED's. And note down your observations.**

SECTION - I

Group-B (Total Slots = 1)

Analog I/O and Timing

Introduction

The **Arduino Uno** provides both **analog input** and **digital timing** functions, enabling it to interact with sensors, actuators, and other electronic devices. Analog input allows the Arduino to read continuously varying voltages from sensors such as LDRs, potentiometers, and temperature sensors. Timing functions enable the Arduino to perform tasks after specific intervals or execute operations at regular time intervals.

Analog Input (Analog I/O)

Analog input is used to read voltage levels that vary continuously between **0 V and 5 V**.

The Arduino Uno has **six analog input pins (A0–A5)** connected to a **10-bit Analog-to-Digital Converter (ADC)**. The ADC converts an analog voltage into a digital value ranging from **0 to 1023**.

Analog Input Specifications

Parameter	Value
Analog Pins	A0 – A5
ADC Resolution	10-bit
Input Voltage Range	0 – 5 V
Digital Output Range	0 – 1023

Analog Input Formula

$$\text{ADC Value} = \frac{Input\ Voltage}{Reference\ Voltage} \times 1023$$

Example

Input Voltage	ADC Value (Approx.)
0 V	0
1 V	205
2.5 V	512
5 V	1023

Reading an Analog Value:

```
int sensorPin = A0;
int sensorValue;

void setup()

{
  Serial.begin(9600);
}

void loop()

{
  sensorValue = analogRead(sensorPin);
  Serial.println(sensorValue);
  delay(500);
}
```

Applications

- Potentiometer
- LDR (Light Sensor)
- Temperature Sensor
- Gas Sensor
- Soil Moisture Sensor
- Pressure Sensor

Analog Output (PWM)

The Arduino Uno does **not provide a true analog output**. Instead, it generates an analog-like signal using **Pulse Width Modulation (PWM)**. PWM rapidly switches a digital pin ON and OFF. By changing the **duty cycle**, the average output voltage can be controlled.

The PWM pins on Arduino Uno are: **D3, D5, D6, D9, D10, and D11**

PWM Output Range

Function	Range
`analogWrite()`	0 – 255

Example

```
int ledPin = 9;

void setup() {
}

void loop() {
  analogWrite(ledPin, 128);   // 50% brightness
}
```

Applications

- LED brightness control
- DC motor speed control
- Fan speed control
- Servo motor control (using libraries)

Timing Functions in Arduino:

Timing functions are used to control when events occur without requiring additional hardware.

1. delay()

The delay() function pauses program execution for a specified number of milliseconds.

Syntax

```
delay(time_in_milliseconds);
```

Example

```
digitalWrite(13, HIGH);
delay(1000);
digitalWrite(13, LOW);
delay(1000);
```

2. delayMicroseconds()

This function pauses execution for a specified number of microseconds.

Syntax

```
delayMicroseconds(time);
```

Example

```
delayMicroseconds(10);
```

3. millis()

The millis() function returns the number of milliseconds elapsed since the Arduino board was powered on or reset.

Example

```
unsigned long currentTime;

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

void loop() {
  currentTime = millis();
  Serial.println(currentTime);
  delay(1000);
}
```

Applications

- Digital clocks
- Event timing
- Non-blocking programs
- Data logging

4. micros()

The `micros()` function returns the number of microseconds since the Arduino started running.

Example

```
unsigned long us;

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

void loop() {
  us = micros();
  Serial.println(us);
  delay(1000);
}
```

Comparison of Timing Functions

Function	Unit	Purpose
`delay()`	Milliseconds	Pauses program execution
`delayMicroseconds()`	Microseconds	Short delay generation
`millis()`	Milliseconds	Measures elapsed time
`micros()`	Microseconds	Measures short time intervals

Applications of Analog I/O and Timing

- Sensor data acquisition
- Automatic lighting systems
- Temperature monitoring
- Motor speed control
- LED brightness control
- Robotics
- Industrial automation
- Data logging
- Smart irrigation systems
- Home automation

Group-B	Experiment No.-5	
Title:	Analog input interfacing using potentiometer and displaying values on Serial Monitor (Total Slots = 1)	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim:
To interface a potentiometer with a microcontroller as an analog input device and display the analog values on the Serial Monitor.

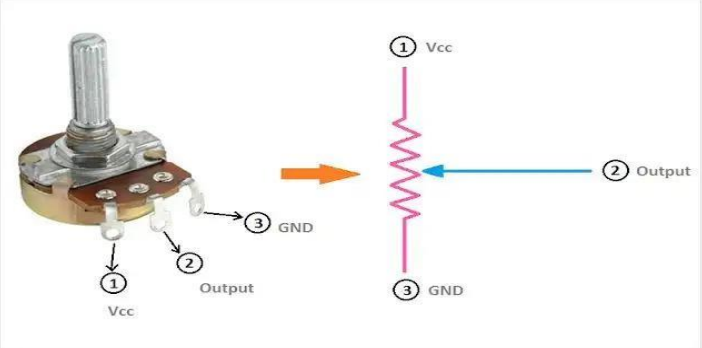
Objective:
To read analog voltage values from a potentiometer using the ADC (Analog-to-Digital Converter) of the microcontroller and display the corresponding digital values on the Serial Monitor.

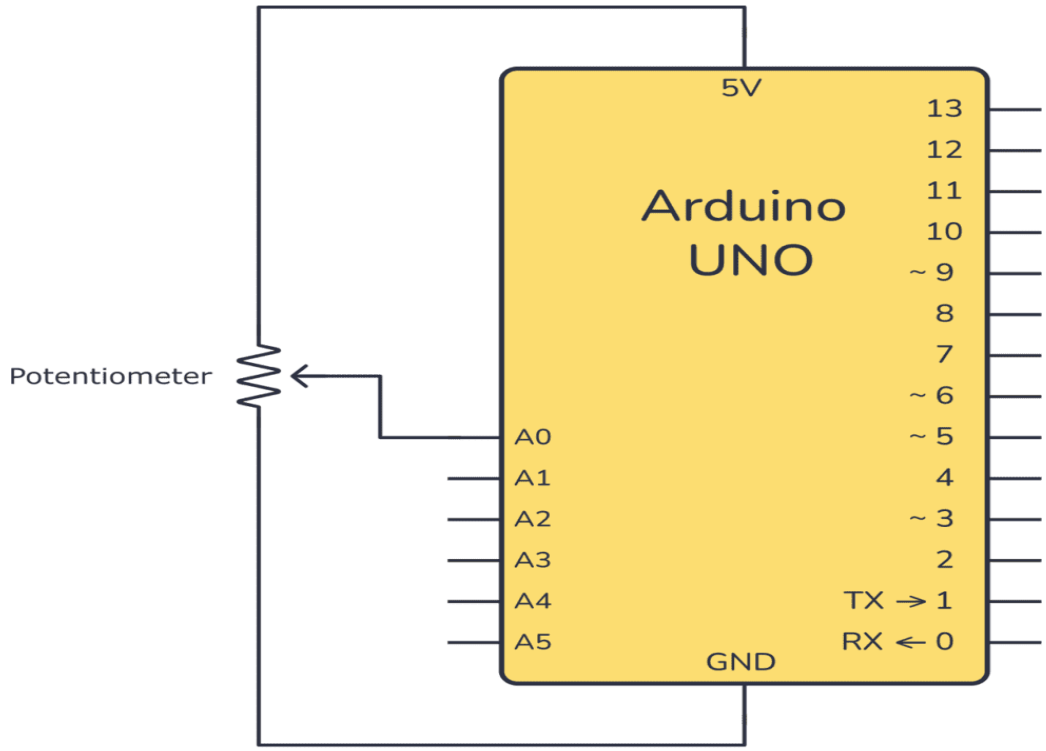
Components Required:
- Arduino Uno board
- Potentiometer (10kΩ)
- Breadboard
- Jumper wires
- USB cable

Theory
A potentiometer is a variable resistor used to provide varying analog voltage input to the microcontroller.
The Arduino Uno has an inbuilt ADC (Analog-to-Digital Converter) that converts analog voltages into digital values.
- Input voltage range: 0V to 5V
- ADC resolution: 10-bit
- Output digital range: 0 to 1023

When the potentiometer knob is rotated:
- Output voltage changes
- Analog value changes accordingly
- Value is displayed on the Serial Monitor

Potentiometer:





Circuit Connections:

Potentiometer Pin	**Arduino Connection**
Left Pin	5V
Middle Pin (Wiper)	A0
Right Pin	GND

Program:

```
int potPin = A0;
int sensorValue = 0;

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

void loop()
{
    sensorValue = analogRead(potPin);

    Serial.print("Potentiometer Value: ");
    Serial.println(sensorValue);

    delay(500);
}
```

Result:
The potentiometer was successfully interfaced with the microcontroller, and varying analog input values were displayed on the Serial Monitor.

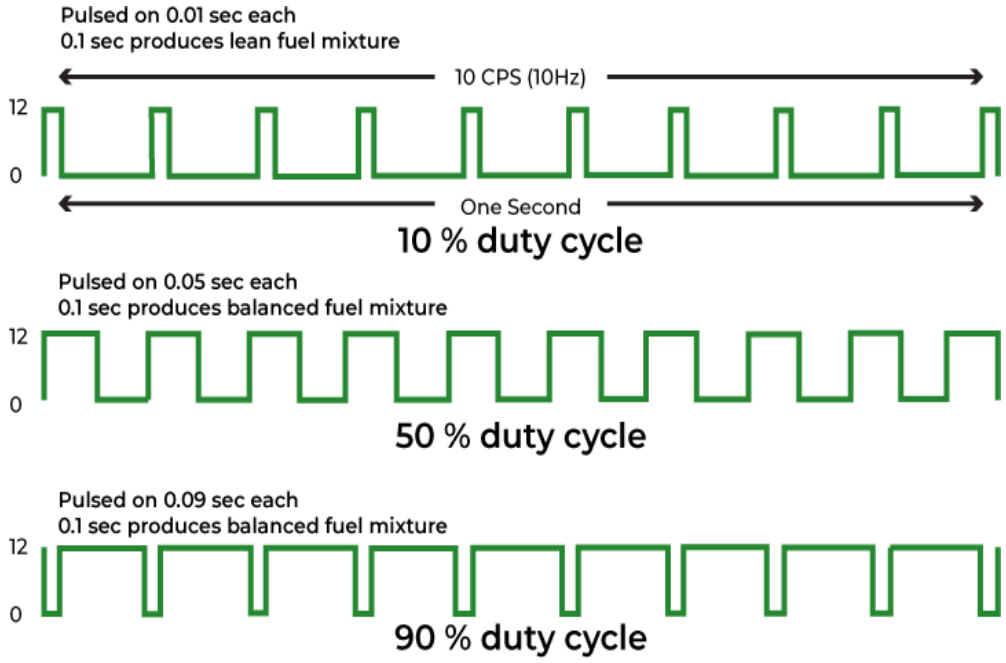
*Modify the program to adjust the potentiometer value and display message accordingly.

Group-B	Experiment No.-6	
Title:	PWM-Based LED Brightness Control Using analogWrite() (Total Slots = 1)	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim: To control the brightness of an LED using Pulse Width Modulation (PWM) with the `analogWrite()` function in Arduino.

Theory:
PWM (Pulse Width Modulation) is a method used to generate an analog-like output using digital signals.
In PWM, the output switches rapidly between ON and OFF states.
The brightness of the LED depends on the **duty cycle**:

$$\text{Duty Cycle} = \frac{TON}{TON+TOFF} \times 100\ \%$$



In PWM:

- The signal rapidly switches between HIGH and LOW.
- The ratio of ON time to total cycle time is called the duty cycle.
- Higher duty cycle → brighter LED.
- Lower duty cycle → dimmer LED.

The `analogWrite()` function generates PWM signals on supported pins.

Syntax:

```
analogWrite(pin, value);
```

Where:

- pin = PWM-enabled pin
- value = brightness level from 0 to 255

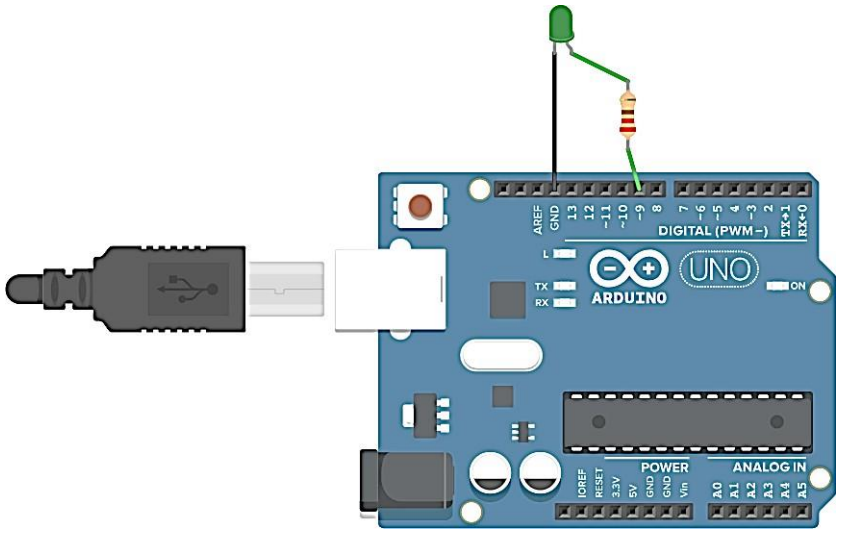
Value	LED Brightness
0	OFF
127	Medium
255	Fully ON

Components Required:

- Arduino Uno/Nano
- LED
- 220Ω resistor
- Breadboard
- Jumper wires
- USB cable

Connections:

Connect Arduino PWM Pin no. 9 to LED anode(+) through resister and GND pin of Arduino to LED cathode(-).



Program:

```
int ledPin = 9;   // PWM pin

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

void loop() {

  // Increase brightness
  for(int brightness = 0; brightness <= 255; brightness++) {
    analogWrite(ledPin, brightness);
    delay(10);
  }

  // Decrease brightness
  for(int brightness = 255; brightness >= 0; brightness--) {
    analogWrite(ledPin, brightness);
    delay(10);
  }
}
```

How does it work??

1. The LED is connected to a PWM-enabled pin.
2. `analogWrite()` sends PWM signals to the LED.
3. The first loop gradually increases duty cycle from 0 to 255.
4. LED brightness increases smoothly.
5. The second loop decreases the duty cycle.
6. LED brightness fades out smoothly.

Result:

The LED brightness was successfully controlled using PWM signals generated by the analogWrite() function. Smooth fading and dimming of the LED was observed.

*Modify the program to blink LED by setting particular value of PWM. This program makes an LED blink at a fixed brightness using PWM. Instead of turning the LED fully ON (HIGH), it is turned ON with a specified PWM value using analogWrite().

Group-B	Experiment No.-7	
Title:	Display of real time on serial monitor with Interfacing to an External Real-Time Clock (RTC) (Total Slots = 1)	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim:
To interface an DS3231 Real-Time Clock (RTC) Module with an Arduino Uno and display the current date and time on the Arduino Serial Monitor.

Objectives:
- To understand the working of a Real-Time Clock (RTC) module.
- To interface the DS3231 RTC module with Arduino using the I²C protocol.
- To display the current date and time on the Serial Monitor.
- To learn the use of RTC libraries in Arduino programming.

Components Required
- Arduino Uno
- DS3231 Real-Time Clock (RTC) Module
- Breadboard
- Jumper wires
- USB cable
- Computer with Arduino IDE
- RTC Library (RTClib)

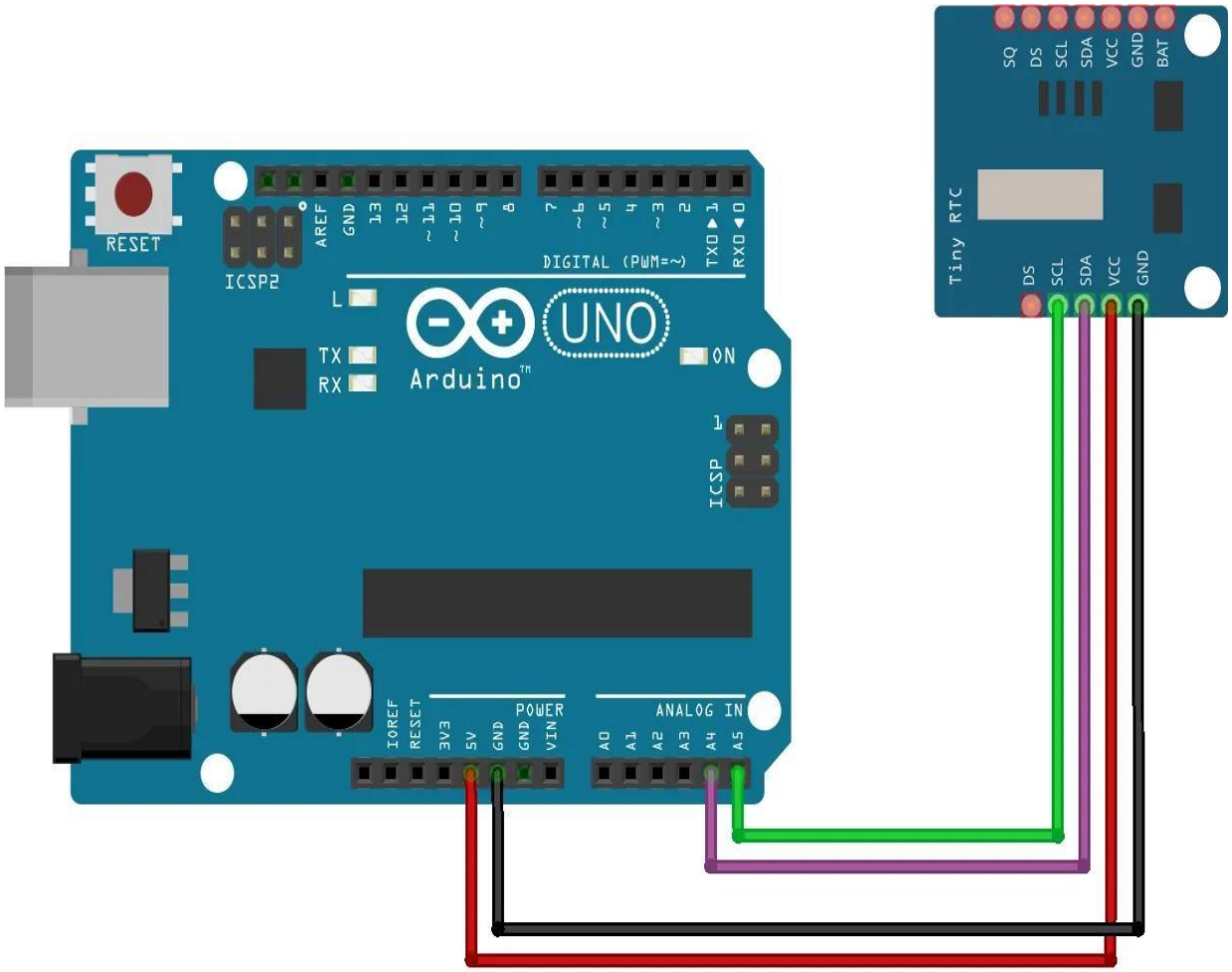
Theory:
A Real-Time Clock (RTC) is an electronic device that keeps track of the current date and time, even when the main power supply is turned off. The DS3231 RTC module contains a highly accurate crystal oscillator and a backup battery, allowing it to maintain time during power interruptions.

The DS3231 communicates with the Arduino using the I²C communication protocol, requiring only two data lines:
- SDA (Serial Data)
- SCL (Serial Clock)

The Arduino reads the current date and time from the RTC module and displays it on the Serial Monitor.

Connections:

RTC Module Pin	Arduino Uno Pin
VCC	5V
GND	GND
SDA	A4
SCL	A5



Program:

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

RTC_DS3231 rtc;

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

  if (!rtc.begin())
  {
    Serial.println("RTC not found!");
    while (1);
  }
```

```
  // Uncomment only once to set RTC to the computer's compile
     time
  // rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
}

void loop()
{

  DateTime now = rtc.now();

  Serial.print("Date: ");
  Serial.print(now.day());
  Serial.print("/");
  Serial.print(now.month());
  Serial.print("/");
  Serial.print(now.year());

  Serial.print("   Time: ");
  Serial.print(now.hour());
  Serial.print(":");
  Serial.print(now.minute());
  Serial.print(":");
  Serial.println(now.second());

  delay(1000);
}
```

Working Principle:

1. The DS3231 RTC module stores the current date and time using its internal clock circuit.
2. A backup battery keeps the clock running even when the Arduino is powered off.
3. The Arduino communicates with the RTC module through the I²C interface (SDA and SCL pins).
4. During execution, the Arduino reads the current date and time from the RTC every second.
5. The retrieved values are displayed on the Serial Monitor using the Serial.print() function.
6. The process repeats continuously, providing a real-time clock display.

Procedure:

1. Connect the DS3231 RTC module to the Arduino according to the circuit diagram.
2. Install the RTClib library in the Arduino IDE.
3. Open the Arduino IDE and enter the program.
4. Select the correct board and COM port.
5. Upload the program to the Arduino.
6. Open the Serial Monitor and set the baud rate to 9600 bps.
7. Observe the current date and time updating every second.
8. If using the RTC for the first time, uncomment the rtc.adjust() line, upload the code once, then comment it again and re-upload.

Output will be observed on serial monitor as follows:

```
Date:  19/06/2026  Time:  10:30:01
Date:  19/06/2026  Time:  10:30:02
Date:  19/06/2026  Time:  10:30:03
Date: 19/06/2026   Time: 10:30:04
```

***Modification the program for Alarm and Scheduling Systems Using an RTC Module**

HINT: If the alarm is set for 7:00 AM, the Arduino continuously checks the RTC time. When the RTC displays 07:00:00, the Arduino turns ON a buzzer for a few seconds to notify the user. Similarly, a relay can be activated to switch ON a light or other electrical device at a specified time.

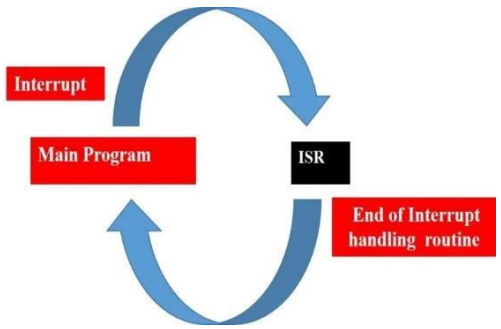
Group-B	Experiment No.-8	
Title:	Interrupt-based input handling using external interrupts. (Total Slots = 1)	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim:
To study and implement interrupt-based input (a push button) handling using external interrupts in a microcontroller and control an LED.

Objective:
- To understand the concept of external interrupts in Arduino.
- To implement interrupt-based input handling using a push button.
- To study the working of Interrupt Service Routines (ISR).
- To control an LED using external interrupt signals.
- To compare interrupt-based operation with polling techniques.

Components Required
- Arduino Uno board
- Push Button
- LED
- 220Ω resistor
- Breadboard
- Jumper wires
- USB cable

Theory:
Interrupt-based input handling using external interrupts is a common technique in embedded systems and microcontrollers to respond immediately to input events (like button presses, sensor signals, or communication requests) without continuously checking the input in a loop.
What is an Interrupt?
An interrupt is a signal that temporarily stops the normal execution of a program so the processor can execute a special function called an Interrupt Service Routine (ISR).
After the ISR finishes, the processor resumes the previous task.



What are the External Interrupts?

External interrupts are generated by hardware signals from outside the Arduino.

In Arduino, external interrupts allow the microcontroller to respond immediately to events like **button presses, motion sensors, encoder signals, etc.,** without continuously checking inputs in the loop() function.

How Interrupts is work ??

1. Main program executes normally.
2. External device generates a signal.
3. Interrupt controller detects the event.
4. CPU pauses current execution.
5. ISR executes automatically.
6. ISR handles the input event.
7. CPU returns to the main program.

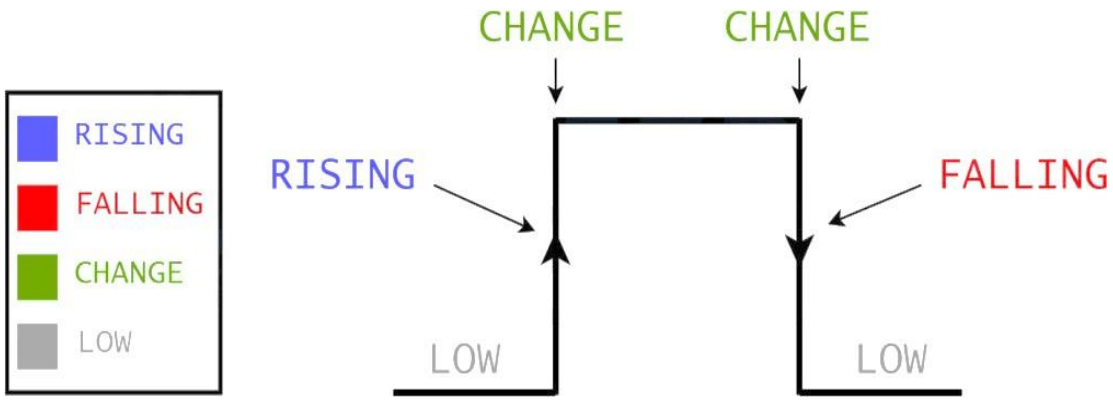
Common Arduino Interrupt Pins:

- Digital Pin 2 for interrupt- 0
- Digital Pin 3 for interrupt- 1

Syntax:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

Parameter	Description	
`pin`	Interrupt pin	**`digitalPinToInterrupt(pin)`** Converts the digital pin number into the interrupt number supported by the board. For Arduino Uno: • Pin 2 → Interrupt 0 • Pin 3 → Interrupt 1
`ISR`	Interrupt Service Routine function	A function that executes automatically when the interrupt occurs. **`void blinkLED()`** **`{`** **`  // interrupt code`** **`}`**
`mode`	Trigger condition	Defines when the interrupt should occur. RISING: LOW → HIGH FALLING: HIGH → LOW CHANGE: Any change LOW: While pin is LOW

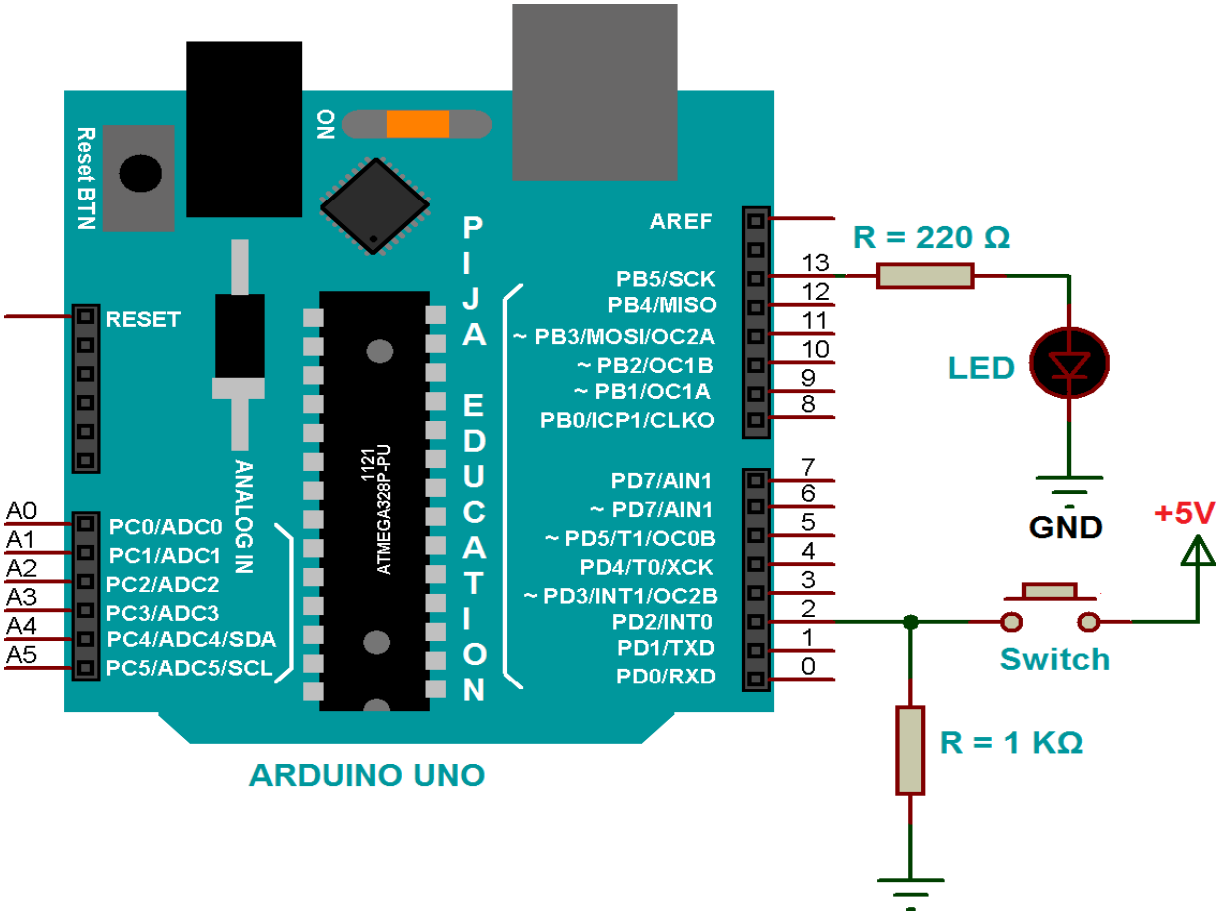



```
attachInterrupt(digitalPinToInterrupt(2), blinkLED, FALLING);
```

`attachInterrupt()` allows Arduino to respond immediately to external events without continuously checking the input in the **`loop()`** function.

Meaning:

This Monitor Arduino pin 2 and Call blinkLED() when signal changes from HIGH to LOW



Program-1:
Simple program to control LED using external interrupt (Switch pressed)
LED connected to pin-13 and switch connected to pin-2.

```
const int button = 2;
const int ledPin = 13;

volatile bool state = false;

void setup()
{
    pinMode(button, INPUT_PULLUP);
    pinMode(ledPin, OUTPUT);

    attachInterrupt(digitalPinToInterrupt(button),changeState,FALLING);
}

void loop()
{
    digitalWrite(ledPin, state);
}
void changeState()
{
    state = !state;
}
```

Why volatile is Used?
In Arduino interrupt programming, a variable shared between:

- Main program (loop())
- Interrupt Service Routine (ISR) must be declared as volatile.

Purpose of volatile
Normally, the compiler optimizes code by storing variable values in CPU registers for faster execution. But ISR can change the variable anytime.
volatile is a keyword used in Arduino and C/C++ programming to tell the compiler that a variable's value can change at any time unexpectedly.

Syntax:
`volatile dataType variableName;`

example from previous program: `volatile bool state = false;`

Keyword	Meaning
`volatile`	Variable can change unexpectedly
`bool`	Boolean data type (true=1 or false=1)
`state`	Variable name
`= false`	Initial value of variable is FALSE

Program-2 :
Program to use an external interrupt in Arduino to start and stop LED blinking.

```
volatile bool blinkEnable = false;

void setup()
{

  pinMode(13, OUTPUT);
  pinMode(2, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(2), changeState, FALLING);
}

void loop()
{

  if (blinkEnable)
  {

    digitalWrite(13, HIGH);
    delay(100);

    digitalWrite(13, LOW);
    delay(100);
  }

  else
  {

   digitalWrite(13, LOW);
  }
}

// Interrupt Service Routine
void changeState()
{

  blinkEnable = !blinkEnable;
}
```

***Modify the above program using an external interrupt to alternately blink two LEDs.**

- **LED on pin 13 blinks when blinkEnable = true**
- **LED on pin 8 blinks when blinkEnable = false**

Pressing the push button connected to pin 2 changes the blinking state using an interrupt.

SECTION - I

Group-C

Sensor Interfacing

(Total Slots = 1)

Introduction:

A **sensor** is an electronic device that detects or measures a physical quantity such as light, temperature, humidity, distance, pressure, or motion, and converts it into an electrical signal. In embedded systems, sensors serve as **input devices**, providing real-world data to the microcontroller.

The **Arduino Uno** can interface with a wide variety of digital and analog sensors. It processes the sensor data and performs appropriate actions, such as displaying information, controlling actuators, or communicating with other devices.

Objectives:

After studying this topic, students will be able to:

- Understand the concept of sensor interfacing.
- Differentiate between analog and digital sensors.
- Interface various sensors with the Arduino Uno.
- Read sensor data using Arduino.
- Develop basic sensor-based applications.

What is Sensor Interfacing?

Sensor interfacing is the process of connecting a sensor to a microcontroller so that the sensor's output can be read, processed, and used to perform a desired operation.

The Arduino reads sensor outputs through:

- **Analog input pins (A0–A5)** for analog sensors.
- **Digital input/output pins (D0–D13)** for digital sensors.
- Communication interfaces such as **I²C, SPI, or UART** for advanced sensors.

Types of Sensors:

1. Analog Sensors

Analog sensors provide a continuously varying voltage proportional to the measured physical quantity. The Arduino converts this voltage into a digital value using its built-in **10-bit Analog-to-Digital Converter (ADC)**.

Examples

- LDR (Light Dependent Resistor)
- Potentiometer

- LM35 Temperature Sensor
- Soil Moisture Sensor
- Gas Sensor (MQ Series)
- Flex Sensor

2. Digital Sensors

Digital sensors provide discrete output signals, typically **HIGH (1)** or **LOW (0)**, or communicate digitally using protocols such as I²C, SPI, or UART.

Examples

- IR Obstacle Sensor
- Push Button
- PIR Motion Sensor
- DHT11 Temperature and Humidity Sensor
- Ultrasonic Sensor (HC-SR04)
- RTC Module (DS3231)

Sensor Interfacing Process

1. Connect the sensor to the Arduino.
2. Supply the required operating voltage.
3. Read the sensor output.
4. Process the data using the Arduino program.
5. Display the results or control an output device.

Analog Sensor Interfacing: Analog sensors are connected to the Arduino's analog input pins.
Example

Sensor	Arduino Pin
LM35	A0
LDR	A0
Potentiometer	A0

Reading Analog Data

```
int sensorValue;

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

void loop()
{
  sensorValue = analogRead(A0);

  Serial.println(sensorValue);

  delay(500);
}
```

Digital Sensor Interfacing: Digital sensors are connected to digital input pins.

Example

Sensor	Arduino Pin
IR Sensor	D2
Push Button	D3

Reading Digital Data

```
const int sensorPin = 2;

void setup()
{
  pinMode(sensorPin, INPUT);

  Serial.begin(9600);
}

void loop()
{
  int sensorState = digitalRead(sensorPin);

  Serial.println(sensorState);

  delay(500);
}
```

Common Sensors Used with Arduino

Sensor	Parameter Measured	Output Type
LDR	Light Intensity	Analog
LM35	Temperature	Analog
Potentiometer	Position/Voltage	Analog
IR Sensor	Object Detection	Digital
HC-SR04	Distance	Digital
DHT11	Temperature & Humidity	Digital
PIR Sensor	Motion	Digital
Soil Moisture Sensor	Soil Moisture	Analog/Digital
MQ Gas Sensor	Gas Concentration	Analog
DS3231 RTC	Date and Time	I²C

Applications of Sensor Interfacing

- Automatic street lighting
- Home automation
- Industrial automation
- Smart irrigation systems

- Fire and gas detection
- Weather monitoring
- Robotics
- Security systems
- Health monitoring devices
- Internet of Things (IoT)

Advantages

- Real-time monitoring of physical parameters.
- Improved automation and control.
- High accuracy and reliability.
- Easy integration with Arduino.
- Low-cost implementation.
- Suitable for educational and industrial applications.

Group-C	Experiment No.-9	
Title:	**Interfacing temperature sensor (LM35 / DHT11) and displaying values on Display. (Total Slots = 1)**	
Perform Date:	**Teacher Sign for attendance**	**Sign for completion with grade**

Aim:
To interface an LM35 or DHT11 temperature sensor with an Arduino Uno and display the measured temperature values on the Serial Monitor.

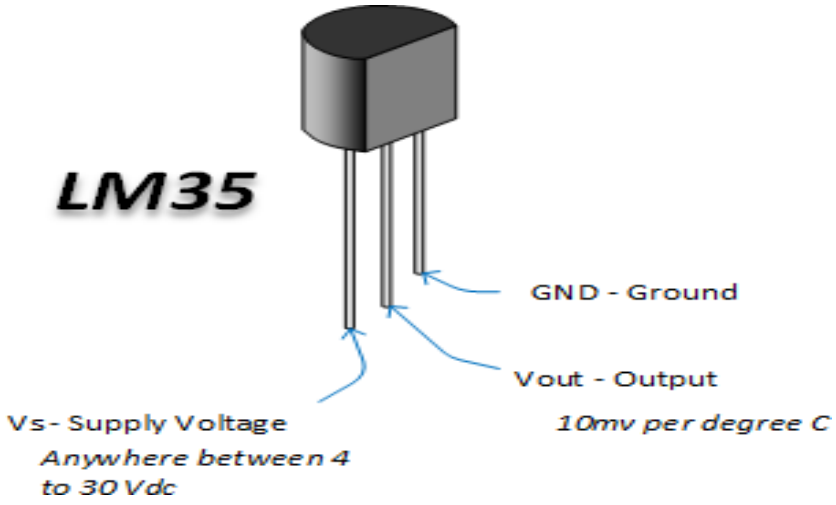
Objective:
- To study the working principle of the LM35 and DHT11 temperature sensors.
- To interface a temperature sensor with an Arduino Uno.
- To measure ambient temperature using the sensor.
- To display the measured temperature (and humidity for DHT11) on the Arduino Serial Monitor.
- To understand the applications of temperature sensors in environmental monitoring and automation.

Components Required:
- Arduino Uno
- LM35 Temperature Sensor **or** DHT11 Temperature & Humidity Sensor
- Breadboard
- Jumper wires
- USB cable
- Computer with Arduino IDE

Theory:
A] Temperature sensing using LM-35:

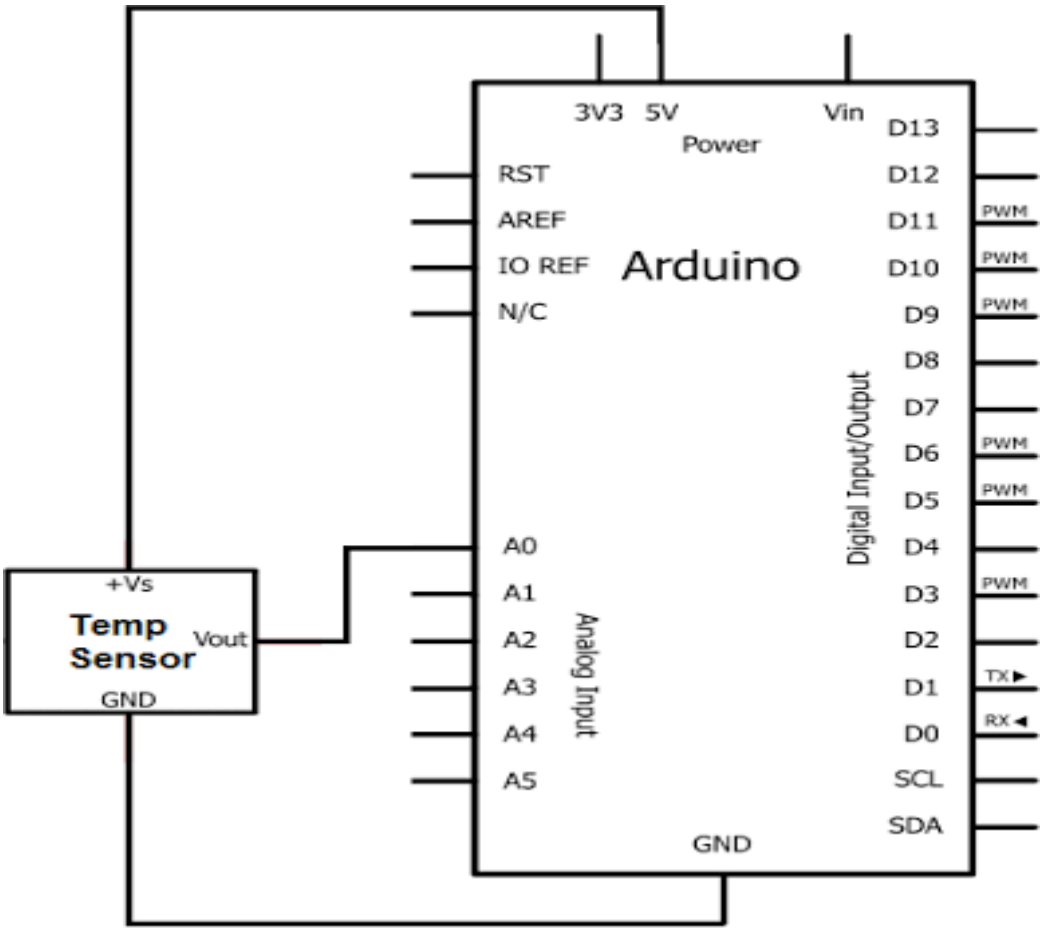
The LM35 is a precision analog temperature sensor whose output voltage is directly proportional to the temperature in degrees Celsius. It provides an output of 10 mV per °C, eliminating the need for external calibration.
Specifications:
- Operating Voltage: 4 V to 30 V
- Temperature Range: −55°C to +150°C
- Output Voltage: 10 mV/°C
- Accuracy: ±0.5°C (typical at 25°C)

The Arduino reads the analog voltage from the LM35 using its ADC (Analog-to-Digital Converter) and converts it into temperature.



Connection:

LM35 Pin	**Arduino Uno Pin**
VCC	5V
OUT	A0
GND	GND



Program:

```
const int LM35 = A0;

void setup()
{
  Serial.begin(9600);
}
void loop()
{
  int sensorValue = analogRead(LM35);

  float voltage = sensorValue * (5.0 / 1023.0);

  float temperature = voltage * 100.0;

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");

  delay(1000);
}
```

Procedure:

1. Connect the LM35 sensor to the Arduino Uno as per the circuit diagram.
2. Open the Arduino IDE and create a new sketch.
3. Enter the program and verify it.
4. Upload the program to the Arduino Uno.
5. Open the Serial Monitor.
6. Set the baud rate to 9600 bps.
7. Observe the temperature readings displayed on the Serial Monitor.
8. Record the readings in the observation table.

The Output can observe on serial monitor as follows:

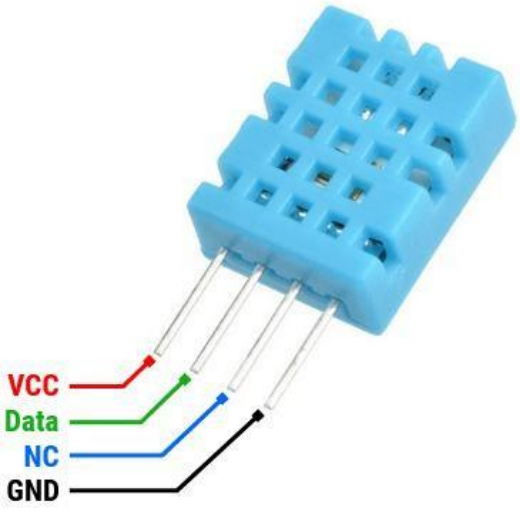
```
Temperature: 28.40 °C
Temperature: 28.50 °C
Temperature: 28.60 °C
Temperature: 28.70 °C
```

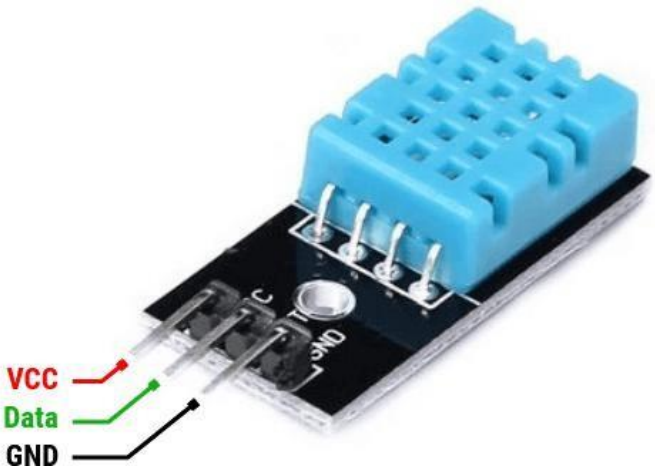
B] Temperature sensing using DHT-11:

The DHT11 is a low-cost digital sensor that measures both temperature and relative humidity. It contains a capacitive humidity sensor and an NTC thermistor. The sensor processes the measured values internally and sends calibrated digital data to the Arduino using a single-wire communication protocol.

Specifications:

- Operating Voltage: 3.3 V – 5 V
- Temperature Range: 0°C to 50°C
- Temperature Accuracy: ±2°C
- Humidity Range: 20%–90% RH
- Humidity Accuracy: ±5% RH
- Output: Digital





Circuit Connections:

DHT11 Pin	Arduino Uno Pin
VCC	5V
DATA	D2
GND	GND

DHT-11 Library: #include <DHT.h>

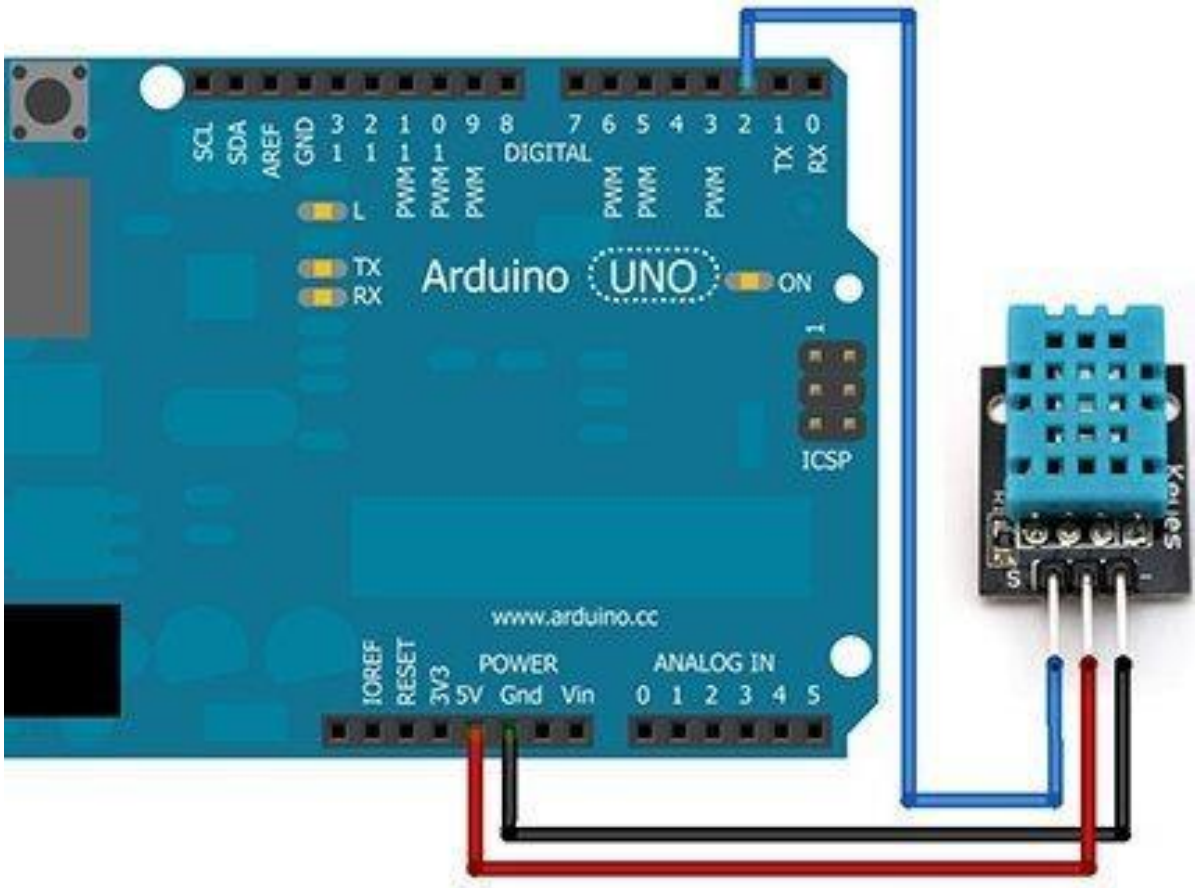
- The #include directive tells the Arduino compiler to include the DHT library in the program.
- The DHT library provides built-in functions to communicate with DHT11 and DHT22 temperature and humidity sensors.
- It eliminates the need to write the communication protocol manually, making the program simpler and more reliable.

Functions Provided by the DHT Library:

Function	Description
`dht.begin();`	Initializes the DHT sensor.
`dht.readTemperature();`	Reads the temperature in degrees Celsius (°C).
`dht.readHumidity();`	Reads the relative humidity (%RH).
`dht.readTemperature(true);`	Reads the temperature in degrees Fahrenheit (°F).

two lines are preprocessor directives that define constants used in the Arduino program.

`#define DHTPIN 2`	Defines the constant DHTPIN with the value 2. It specifies that the DATA pin of the DHT11 sensor is connected to Digital Pin 2 of the Arduino Uno.
`#define DHTTYPE DHT11`	Defines the constant DHTTYPE as DHT11. It tells the DHT library that the connected sensor is a DHT11. This allows the library to use the correct communication protocol and timing.
`DHT dht(DHTPIN, DHTTYPE);`	This statement initializes the DHT sensor by specifying: • **DHTPIN** → Arduino Digital Pin 2 • **DHTTYPE** → Sensor model (DHT11)



Program:

```
#include <DHT.h>

#define DHTPIN 2

#define DHTTYPE DHT11


DHT dht(DHTPIN, DHTTYPE);

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

void loop()
{
  float temperature = dht.readTemperature();

  float humidity = dht.readHumidity();


  if (isnan(temperature) || isnan(humidity))
  {
    Serial.println("Failed to read from DHT11 sensor!");
    return;
  }

  Serial.print("Temperature: ");

  Serial.print(temperature);

  Serial.print(" °C");


  Serial.print("   Humidity: ");

  Serial.print(humidity);

  Serial.println(" %");


  delay(2000);
}
```

The output could be observe on serial monitor:

```
Temperature: 28.0 °C   Humidity: 60 %
Temperature: 28.2 °C   Humidity: 61 %
Temperature: 28.3 °C   Humidity: 60 %
Temperature: 28.5 °C   Humidity: 62 %
```

***Modify the program to display different messages on the Serial Monitor based on the measured temperature/Humidity.**

Group-C	Experiment No.-10	
Title:	**Interfacing light sensor (LDR) for automatic light control system (Total Slots = 1)**	
Perform Date:	**Teacher Sign for attendance**	**Sign for completion with grade**

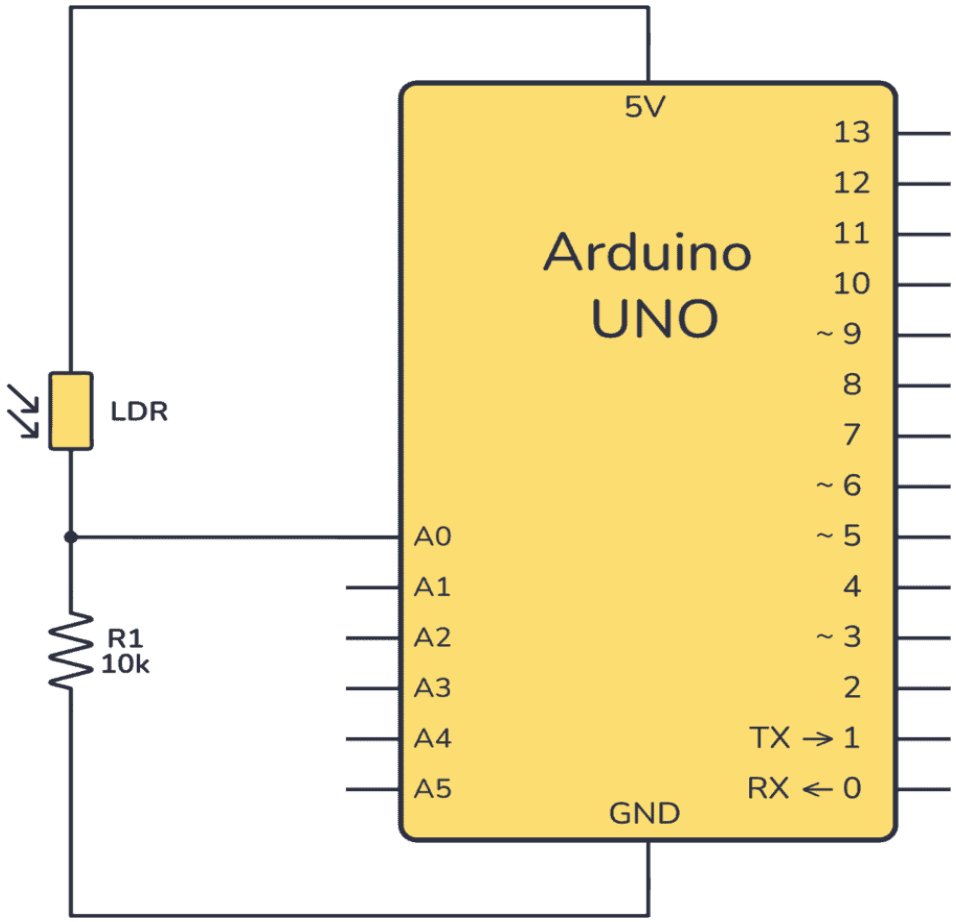
Aim
To interface a Light Dependent Resistor (LDR) with an Arduino Uno and automatically control an LED based on the surrounding light intensity.

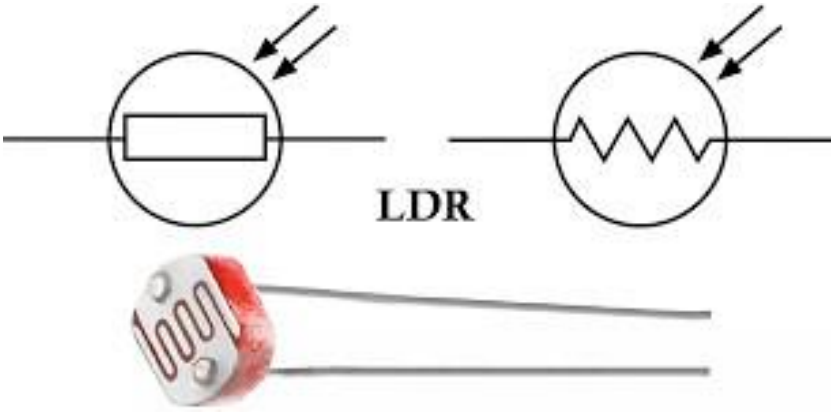
Objectives
- To understand the working principle of an LDR.
- To interface an LDR with Arduino using an analog input.
- To control an LED automatically depending on ambient light.
- To learn analog sensor reading and digital output control in Arduino.

Apparatus Required:
- Arduino Uno
- LDR (Light Dependent Resistor)
- LED
- 10 kΩ resistor
- 220 Ω resistor
- Breadboard
- Jumper wires
- USB cable
- Computer with Arduino IDE

Theory
An LDR (Light Dependent Resistor) is a light-sensitive resistor whose resistance changes with the intensity of light. Its resistance decreases in bright light and increases in darkness.
The LDR is connected in a voltage divider circuit, producing a variable voltage that is read by the Arduino's analog input pin (A0). The Arduino compares the sensor value with a preset threshold. If the light intensity is below the threshold (dark condition), the Arduino turns the LED ON. Otherwise, the LED remains OFF.

First we interface an **LDR (Light Dependent Resistor)** with an **Arduino Uno** and measure the ambient light intensity using the Arduino's analog input.
For this prepare **LDR Voltage Divider** by making connections as follows:
- One terminal of the LDR → **5V**
- Other terminal of the LDR → **Analog Pin A0**
- Connect a **10 kΩ resistor** between **A0** and **GND**


5V
Arduino
UNO
13
12
11
10
~ 9
8
7
~ 6
~ 5
4
~ 3
2
TX → 1
RX ← 0
A0
A1
A2
A3
A4
A5
GND
LDR
R1
10k


LDR

Program-1: To interface an LDR (Light Dependent Resistor) with an Arduino Uno and measure the ambient light intensity using the Arduino's analog input.

```
const int ldrPin = A0;

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

void loop() {
  int ldrValue = analogRead(ldrPin);

  Serial.print("The LDR Value is: ");
  Serial.println(ldrValue);

  delay(500);
}
```

How it Working:

- The LDR changes its resistance according to the amount of light falling on it.
- In bright light, the LDR's resistance decreases, producing a different voltage at pin **A0**.
- In darkness, the LDR's resistance increases, changing the voltage at **A0**.
- The Arduino reads this voltage as an analog value between **0 and 1023** using analogRead().

Light Condition	**Approximate Analog Value**
Bright light	700–1023
Normal light	400–700
Dark	0–400

Program-2: To interface a Light Dependent Resistor (LDR) with an Arduino Uno and automatically control an LED based on the surrounding light intensity.

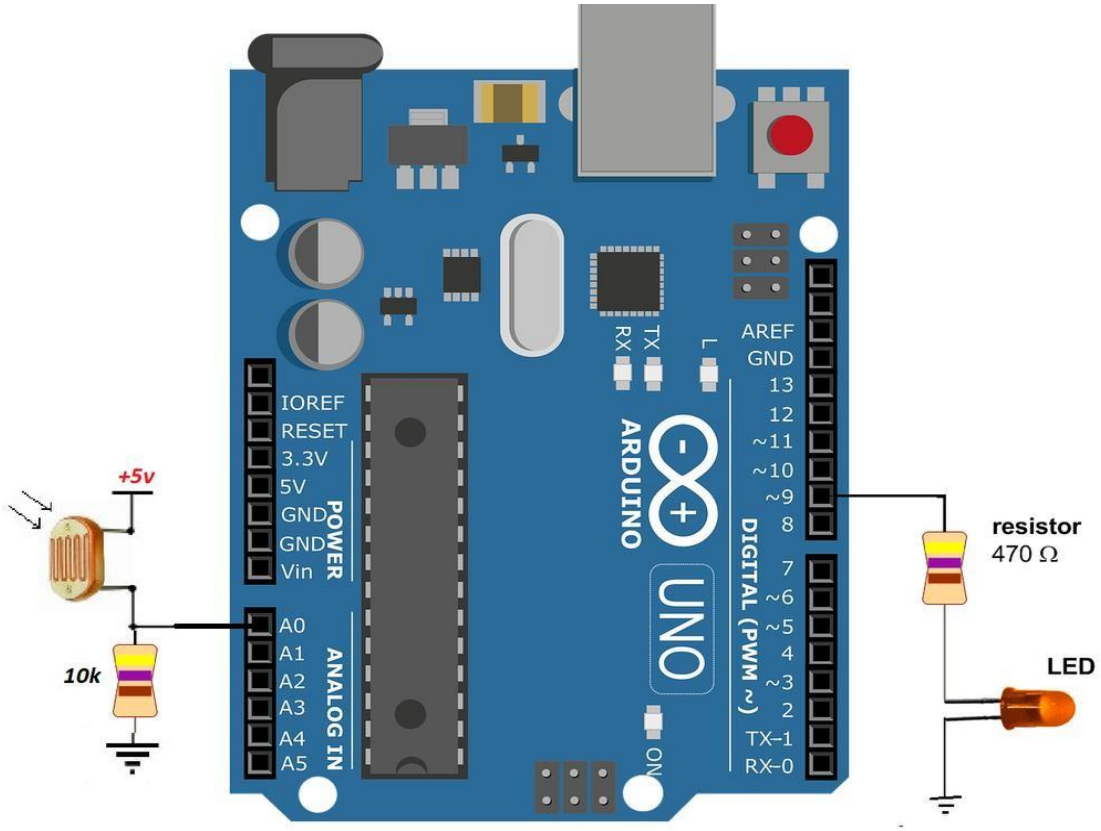
Working Principle

The Automatic Light Control System works by sensing the surrounding light intensity using a Light Dependent Resistor (LDR) and controlling an LED (or a lamp through a relay) with an Arduino Uno.

1. The LDR is connected in a voltage divider circuit with a 10 kΩ resistor.
2. The resistance of the LDR changes with the amount of light falling on it:
    - Bright light: LDR resistance decreases.
    - Darkness: LDR resistance increases.
3. The voltage across the voltage divider is applied to the Arduino's analog input pin (A0).
4. The Arduino continuously reads this voltage using the analogRead() function, which converts it into a digital value ranging from 0 to 1023.
5. The measured value is compared with a predefined threshold in the program.
6. If the light intensity falls below the threshold (dark condition), the Arduino switches the output pin HIGH, turning the LED ON.
7. When the light intensity rises above the threshold (bright condition), the Arduino sets the

output pin LOW, turning the LED OFF.

8. This process repeats continuously, allowing the system to automatically control the light according to ambient lighting conditions.



Connections:

Component	**Arduino Pin**
LDR Terminal 1	5V
LDR Terminal 2	A0
10 kΩ Resistor	A0 to GND
LED Anode (+)	Digital Pin 9
LED Cathode (–)	220 Ω resistor to GND

```
const int ldrPin = A0;
const int ledPin = 9;

int threshold = 500;

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{

  int ldrValue = analogRead(ldrPin);

  Serial.print("LDR Value: ");
  Serial.println(ldrValue);

  if (ldrValue < threshold)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }

  delay(200);
}
```

Observations:

Light Condition	**LDR Value**
Bright Light	700–1023
Normal Light	400–700
Dark	0–400

***Modify the program to automatically control two LEDs to indicate different ambient light intensity levels.**

Group-C	Experiment No.-11	
Title:	**Interfacing ultrasonic / IR sensor for Displacement measurement. (Total Slots = 1)**	
Perform Date:	**Teacher Sign for attendance**	**Sign for completion with grade**

Aim

To interface an Ultrasonic Sensor (HC-SR04) or IR Distance Sensor with an Arduino Uno to measure the displacement (distance) of an object and display the measured value on the Serial Monitor.

Objective:

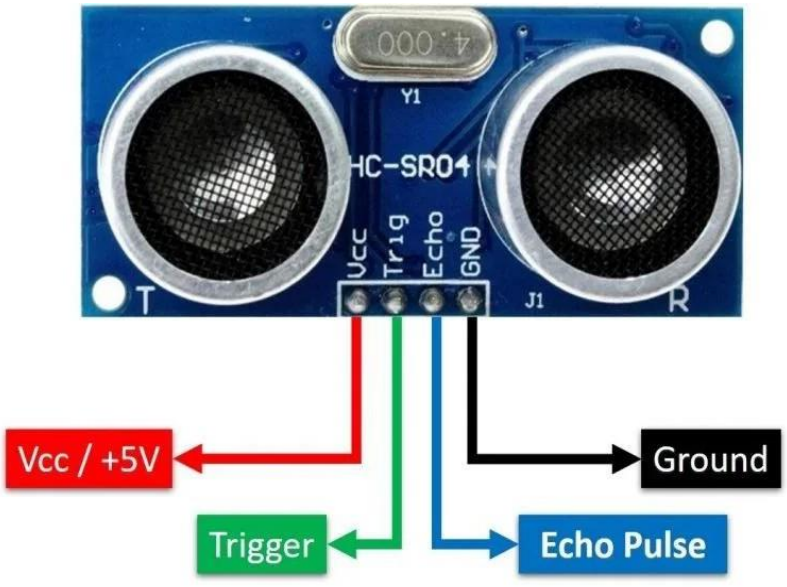
- To study the working principle of **Ultrasonic** and **IR (Infrared)** sensors for displacement measurement.
- To interface an Ultrasonic or IR sensor with an Arduino Uno.
- To measure the displacement (distance) of an object using the selected sensor.
- To acquire sensor data and process it using the Arduino microcontroller.
- To display the measured distance or object detection status on the Arduino Serial Monitor.
- To understand the applications of Ultrasonic and IR sensors in robotics, industrial automation, obstacle detection, and distance measurement systems.

Option A: Displacement measurement using Ultrasonic Sensor (HC-SR04)

Components Required

- Arduino Uno
- HC-SR04 Ultrasonic Sensor
- Breadboard
- Jumper wires
- USB cable

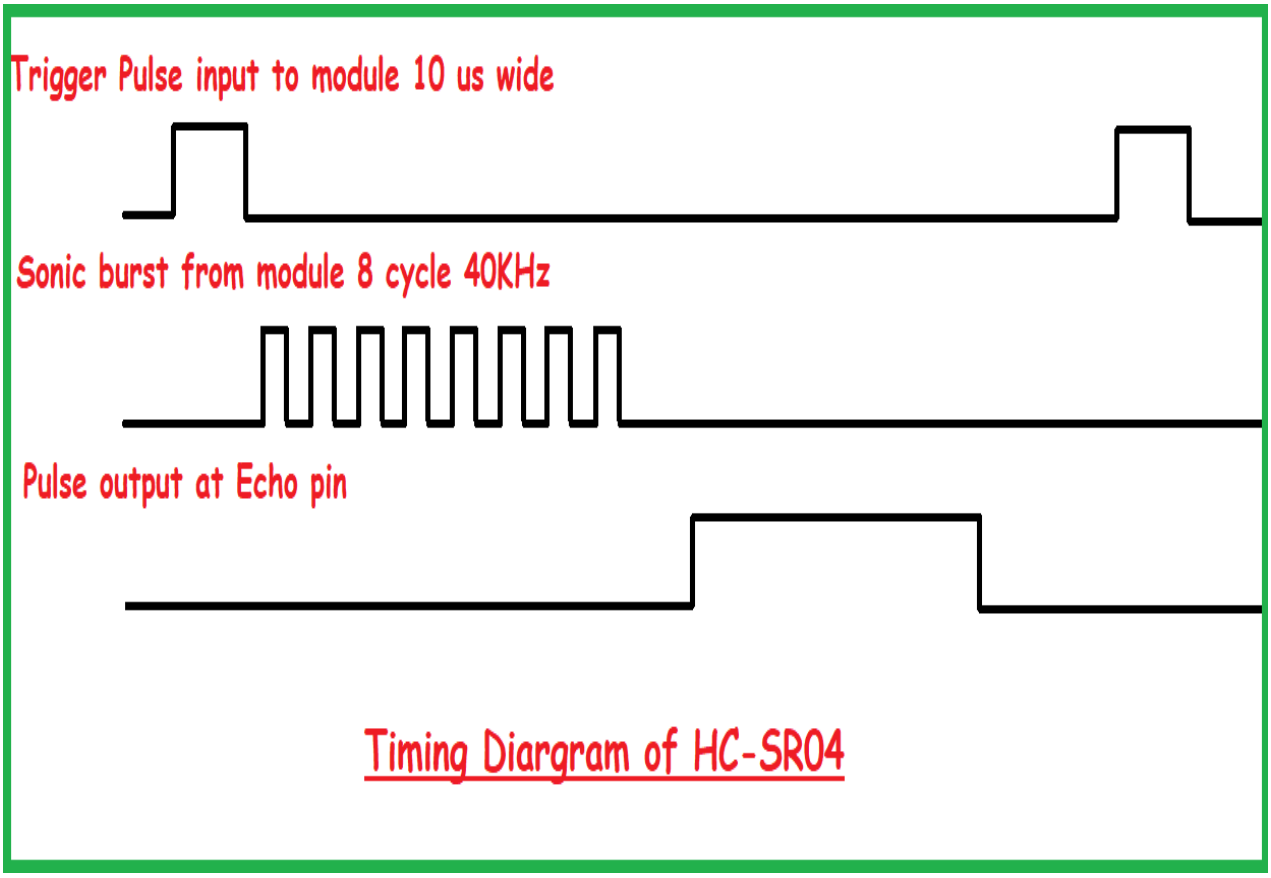
HC-SR04 Module



Pin Name	Function
Vcc	Connect 5 volt supply with this pin.
Trigger	It's an input to sensor. High pulse of **10µs** is applied to this pin. Used to initiate distance ranging
Echo	Provides output pulse signal. It's an output of sensor. High pulse at this pin indicates time taken by sound wave to travel from transmitter to object and object to receiver. In above image left side is transmitting side and right side is receiving side.
Ground	Connect with ground terminal

HC-SR04 Ultrasonic Sensor Working:
The HC-SR04 ultrasonic sensor works based on the principle of echolocation. The sensor sends out a high-frequency sound wave (typically 40 kHz) and measures the time it takes for the wave to bounce off an object and return to the sensor. This time can then be used to calculate the distance between the sensor and the object.

Here is a step-by-step explanation of how the HC-SR04 works:
Triggering the sensor: To start a measurement, a 10µs pulse is sent to the trigger pin of the sensor. This triggers the sensor to emit a burst of ultrasonic sound waves.



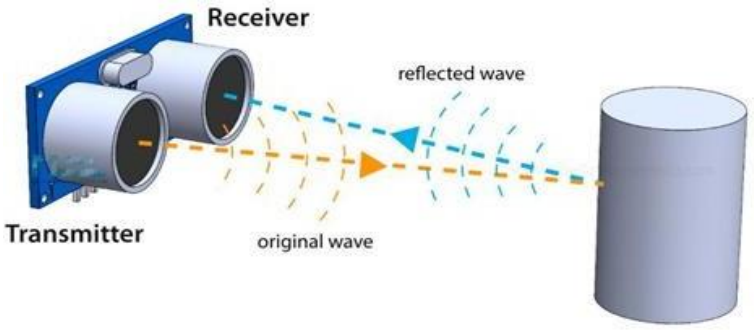
Sending the sound waves: The ultrasonic sound waves travel through the air and bounce off from any objects in their path. The sensor waits for the sound waves to bounce back.
Detecting the echoes: When the sound waves hit an object, they bounce back and are detected by the sensor's receiver. The echo pin goes low when the signal is received. The sensor then generates an echo pulse on the echo pin, which indicates that the sound waves have been received. The width of echo pulse depends on the time taken to receive the signal.

Measuring the time: The time between the trigger pulse and the echo pulse is measured using a timer. This time corresponds to the time it takes for the sound waves to travel to the object and back.
Calculating the distance: Using the time measured in previous step, the distance to the object can be calculated using the following formula:

Distance = (Time taken by the sound wave * Speed of sound) / 2,

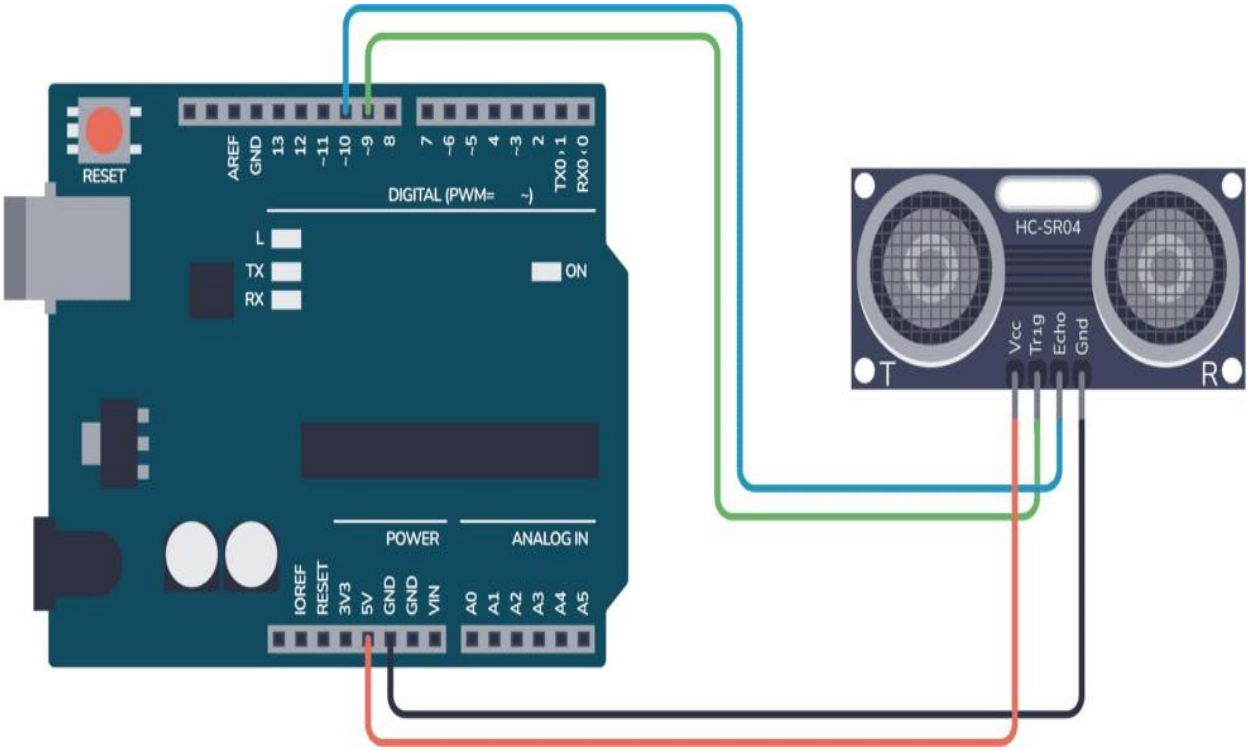
where,
Time taken by the sound wave = Pulse width of the echo pulse,
Speed of sound = 343 meters per second (at room temperature).
It can be also be calculated as:
Distance (in cm) = (Echo pulse time in micro second)/ 58.309

Let's take an example to further clarify the concept. Suppose we have an object in front of the HC-SR04 sensor at an unknown distance, and we receive a pulse of 1000µs width on the Echo pin. To calculate the distance, we have to convert the speed of sound into cm/µs, which is 0.034 cm/µs.
Thus,
Distance = (0.034cm/µs × 1000µs)/2 = 17cm
Therefore, the object is 17cm away from the sensor.



Circuit Connections:

HC-SR04 Pin	Arduino Uno Pin
VCC	5V
GND	GND
TRIG	D9
ECHO	D10



Program:

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

long duration;
float distance;

void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  Serial.begin(9600);
}

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

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);
```

```
  duration = pulseIn(echoPin, HIGH);

  distance = duration * 0.0343 / 2;

  Serial.print("Distance = ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(500);
}
```

Procedure

1. Connect the HC-SR04 sensor to the Arduino Uno as per the circuit diagram.
2. Open the Arduino IDE and create a new sketch.
3. Enter the program and verify it.
4. Upload the program to the Arduino Uno.
5. Open the Serial Monitor and set the baud rate to 9600.
6. Place an object at different distances from the sensor.
7. Observe and record the measured distance displayed on the Serial Monitor.

Observations:

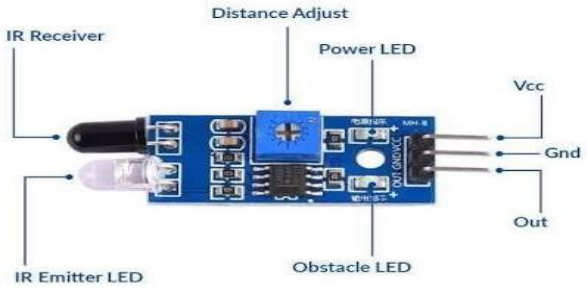
Sr. No.	Actual Distance (cm)	Measured Distance (cm)
1	10	10.2
2	20	20.1
3	30	29.8
4	40	40.3
5	50	49.9

Option B: Displacement measurement using IR Distance Sensor

Components Required

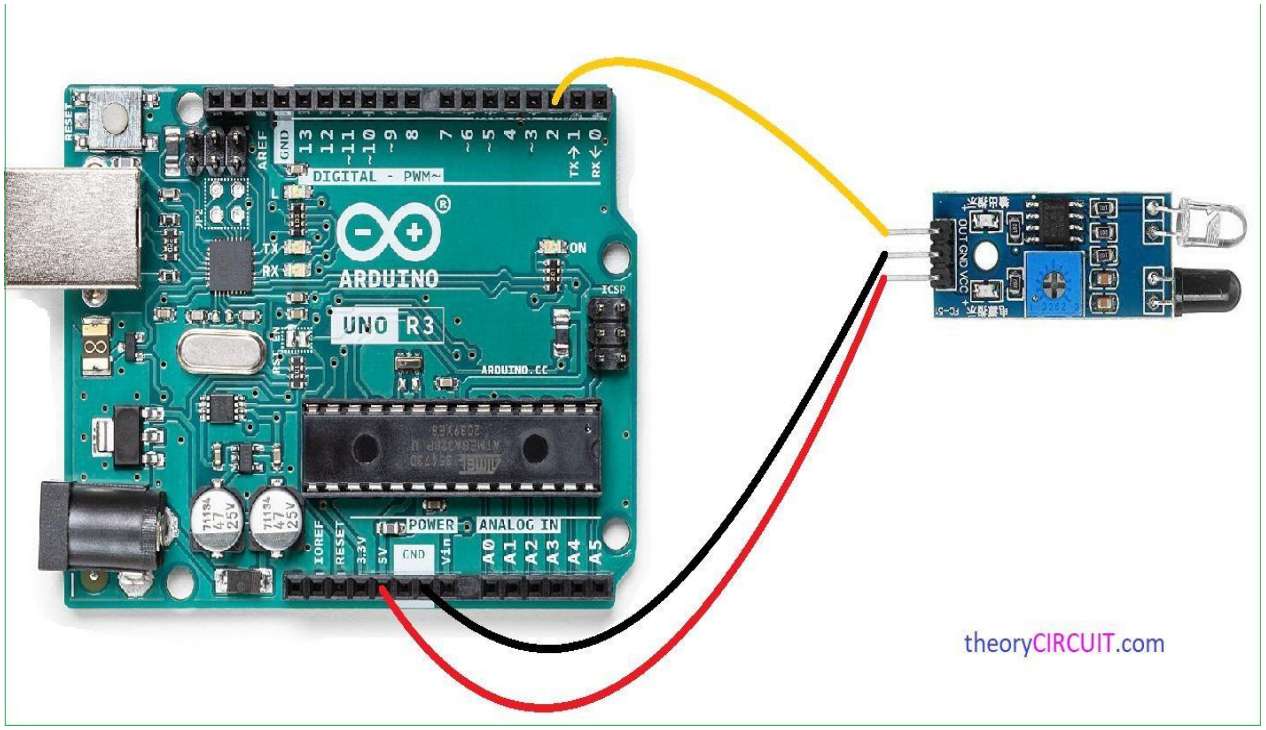
- Arduino Uno
- IR Obstacle/Distance Sensor
- Breadboard
- Jumper wires

IR sensor:



Connections:

IR Sensor Pin	Arduino Pin
VCC	5V
GND	GND
OUT	Digital Pin 2



Program:

```
const int irPin = 2;

void setup()
{
  pinMode(irPin, INPUT);
  Serial.begin(9600);
}
void loop()
{
  int object = digitalRead(irPin);

  if(object == LOW)
  {
    Serial.println("Object Detected");
  }
  else
  {
    Serial.println("No Object");
  }

  delay(500);
}
```

Result:
This program will detect the object whether it is near of far.

*Modify the program to display different messages on the Serial Monitor based on the measured distance using ultrasonic sensor"

Group-C	Experiment No.-12	
Title:	**Interfacing of speakers / buzzers to generate different tones. (Total Slots = 1)**	
Perform Date:	**Teacher Sign for attendance**	**Sign for completion with grade**

Aim:
To interface a speaker/buzzer with a microcontroller and generate different sound tones.

Objective:
To generate different audio tones using a buzzer or speaker connected to a microcontroller.

Components Required

- Arduino Uno board
- Piezo buzzer / speaker
- 220Ω resistor (optional)
- Breadboard
- Jumper wires
- USB cable

Theory:
A buzzer or speaker is an output device used to generate sound signals.
The microcontroller produces sound by generating square wave signals of different frequencies.
Different frequencies produce different tones.

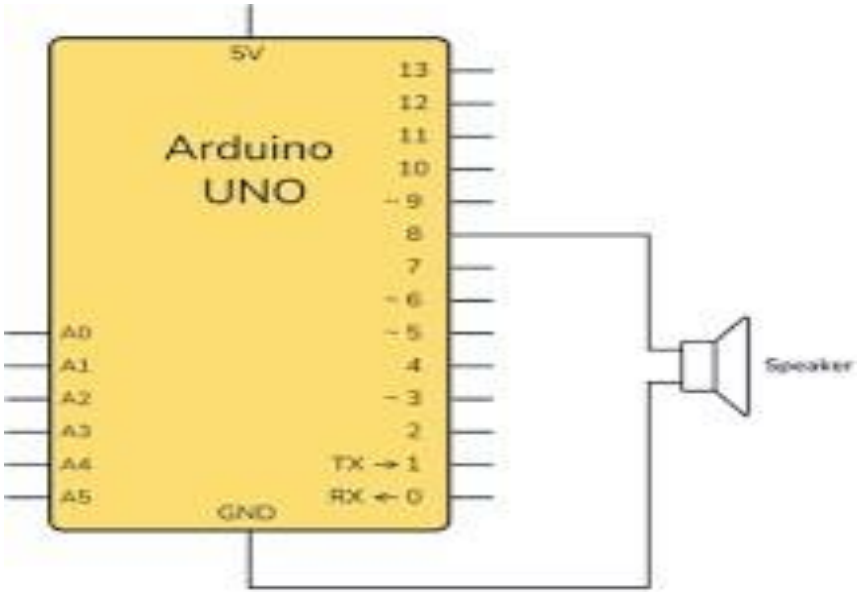
In Arduino programming, the `tone()` function is commonly used:

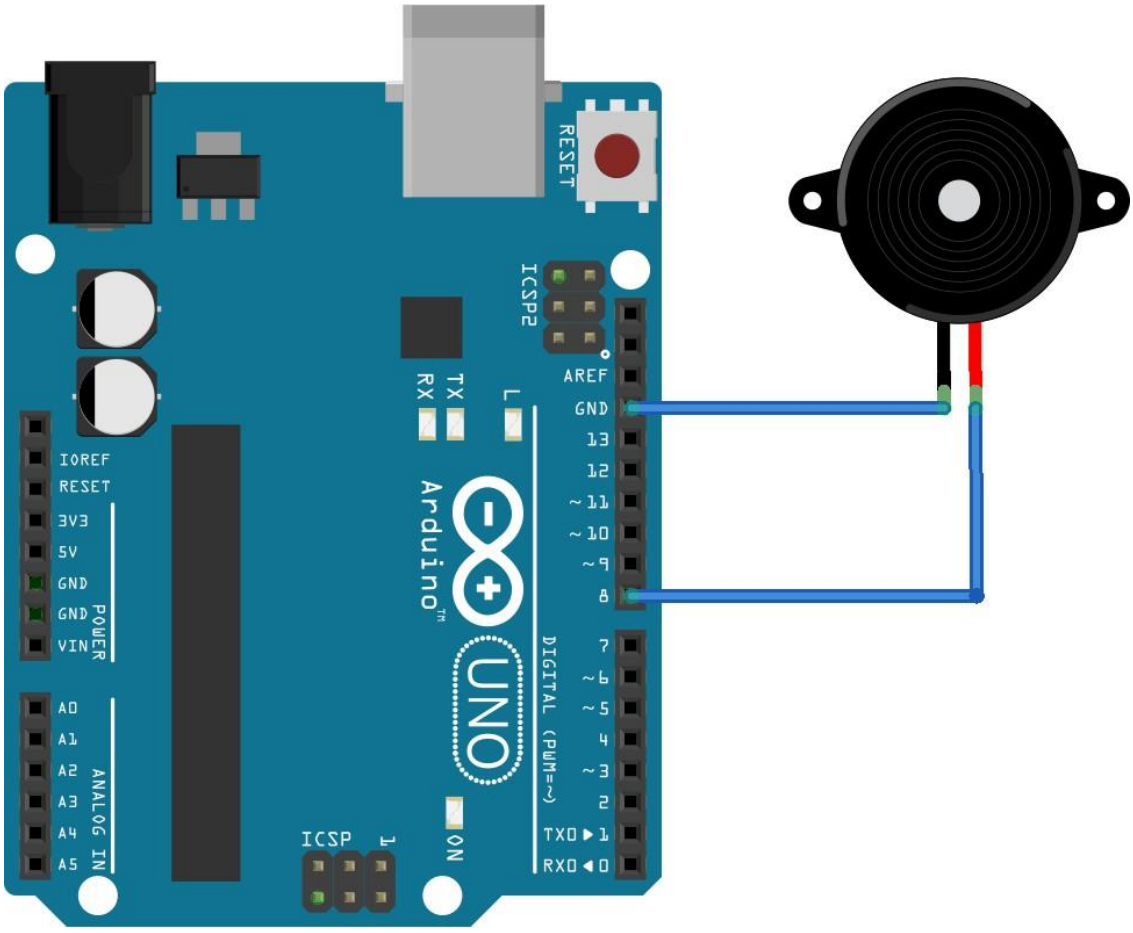
```
tone(pin, frequency);
```

Where:

- pin → buzzer connection pin
- frequency → sound frequency in Hertz (Hz)

Higher frequency → higher pitch sound
Lower frequency → lower pitch sound


5V
Arduino
UNO
13
12
11
10
~9
8
7
~6
~5
4
~3
2
TX →1
RX ←0
A0
A1
A2
A3
A4
A5
GND
Speaker


RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
TX
RX
L
Arduino™
UNO
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN

Program:

```
int buzzer = 8;

void setup()
{
}

void loop()
{
    tone(buzzer, 500);    // 500 Hz tone
    delay(1000);

    tone(buzzer, 1000);  // 1000 Hz tone
    delay(1000);

    tone(buzzer, 1500);  // 1500 Hz tone
    delay(1000);

    noTone(buzzer);       // Stop sound
    delay(1000);
}
```

***Modify the program to generate specific musical tone.**
OR
Modify program to generate different musical tones based on the switch pressed.

SECTION - I

Group-D

Actuators and Communication

(Total Slots = 1)

Introduction

In an embedded system, **sensors** collect information from the environment, while **actuators** perform physical actions based on commands from the microcontroller. Communication interfaces enable the Arduino to exchange data with computers, sensors, displays, and other embedded devices.

The **Arduino Uno** supports various actuators and communication protocols, making it suitable for automation, robotics, industrial control, and Internet of Things (IoT) applications.

Objectives

After studying this topic, students will be able to:

- Understand the concept of actuators.
- Differentiate between sensors and actuators.
- Interface common actuators with Arduino.
- Understand various communication protocols used in Arduino.
- Develop simple actuator- and communication-based applications.

What is an Actuator?

An **actuator** is an output device that converts electrical signals into physical actions such as motion, sound, light, or switching. The Arduino controls actuators by sending digital or PWM signals.

Examples of Actuators

- LED
- Buzzer
- DC Motor
- Servo Motor
- Stepper Motor
- Relay
- Solenoid Valve

Types of Actuators

1. Electrical Actuators

These actuators produce electrical outputs such as light or sound.

Examples:

- LED
- Buzzer
- Relay

2. Mechanical Actuators

These actuators produce mechanical motion.

Examples:

- DC Motor
- Servo Motor
- Stepper Motor
- Solenoid

Common Actuators Used with Arduino

Actuator	Function	Arduino Control
LED	Visual indication	Digital Output/PWM
Buzzer	Sound generation	Digital Output or tone()
DC Motor	Continuous rotation	PWM through a motor driver
Servo Motor	Angular position control	Servo library
Stepper Motor	Precise rotational movement	Stepper driver/library
Relay	Switching AC/DC loads	Digital Output

Actuator Interfacing

The Arduino controls actuators by configuring digital pins as **OUTPUT** using the `pinMode()` function.

Example: LED

```
const int led = 13;

void setup()
{
pinMode(led, OUTPUT);
}
void loop()
{
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
}
```

Example: Buzzer

```
const int buzzer = 8;

void setup()
{
}

void loop()
{
  tone(buzzer, 1000);
  delay(1000);

  noTone(buzzer);
  delay(1000);
}
```

Communication in Arduino

Communication enables the Arduino to exchange data with external devices such as computers, sensors, displays, and other microcontrollers.

The Arduino Uno supports three main communication protocols:

- UART (Serial Communication)
- I²C (Inter-Integrated Circuit)
- SPI (Serial Peripheral Interface)

1. UART (Serial Communication)

UART is used for communication between the Arduino and a computer or another serial device.

Pins

Pin	Function
D0	RX (Receive)
D1	TX (Transmit)

Common Functions

```
Serial.begin(9600);

Serial.print("Hello");

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

Applications

- Serial Monitor
- GPS modules
- Bluetooth modules
- GSM modules

2. I²C Communication

I²C is a two-wire communication protocol that allows multiple devices to communicate using only two signal lines.

Pins

Pin	Function
A4	SDA (Data)
A5	SCL (Clock)

Common Devices

- LCD with I²C Module
- DS3231 RTC
- EEPROM
- Temperature Sensors

Example

```
#include <Wire.h>

void setup()
{
Wire.begin();
}
```

3. SPI Communication

SPI is a high-speed communication protocol commonly used for memory cards, displays, and wireless modules.

SPI Pins

Pin	Function
D10	SS (Slave Select)
D11	MOSI
D12	MISO
D13	SCK

Applications

- SD Card Module
- TFT Display
- RFID Reader
- Wireless Modules

Comparison of Communication Protocols

Feature	UART	I²C	SPI
Number of Wires	2	2	4
Speed	Medium	Medium	High
Multiple Devices	Limited	Yes	Yes
Complexity	Low	Medium	Medium

Feature	UART	I²C	SPI
Typical Applications	PC, Bluetooth, GPS	RTC, LCD, Sensors	SD Cards, Displays, RFID

Applications of Actuators and Communication

- Home automation
- Industrial automation
- Robotics
- Smart irrigation systems
- Data logging
- IoT systems
- Security systems
- Medical equipment
- Smart energy management

Group-D	Experiment No.-13	
Title:	Interfacing relay module for AC/DC load control (demonstration-level) (Total Slots = 1)	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim

To interface a relay module with a microcontroller for controlling AC/DC loads.

Objective

To control an external electrical load using a relay module interfaced with a microcontroller.

Components Required

- Arduino Uno board
- 5V Relay module
- LED/DC bulb (for safe demonstration)
- External DC power supply
- Breadboard
- Jumper wires
- USB cable

Theory

A relay is an electrically operated switch used to control high-voltage or high-current devices using low-voltage microcontroller signals.

The relay module acts as an interface between:

- Low-power control circuit (Arduino)
- High-power load circuit (AC/DC load)

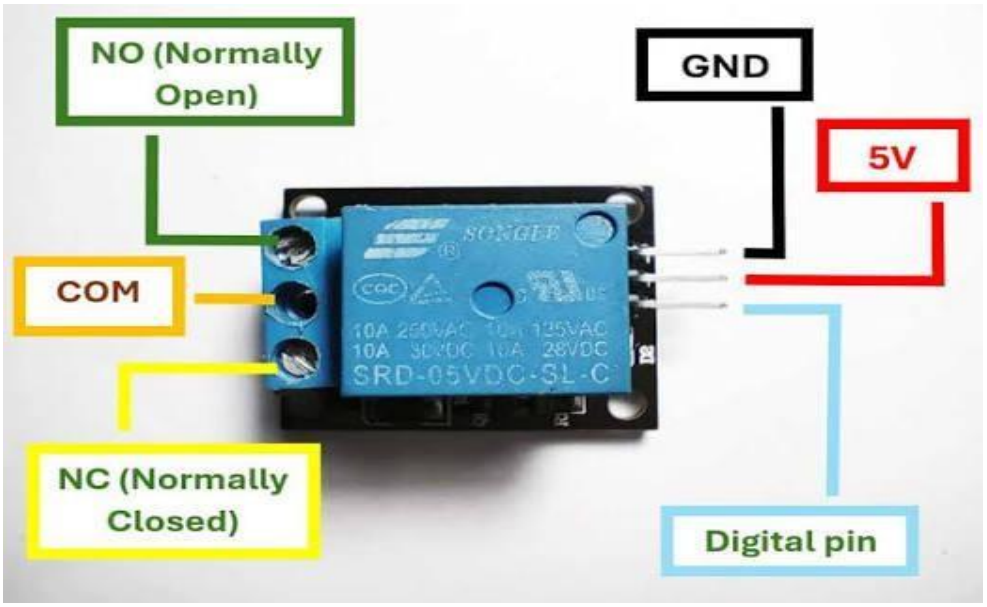
When the microcontroller sends a signal:

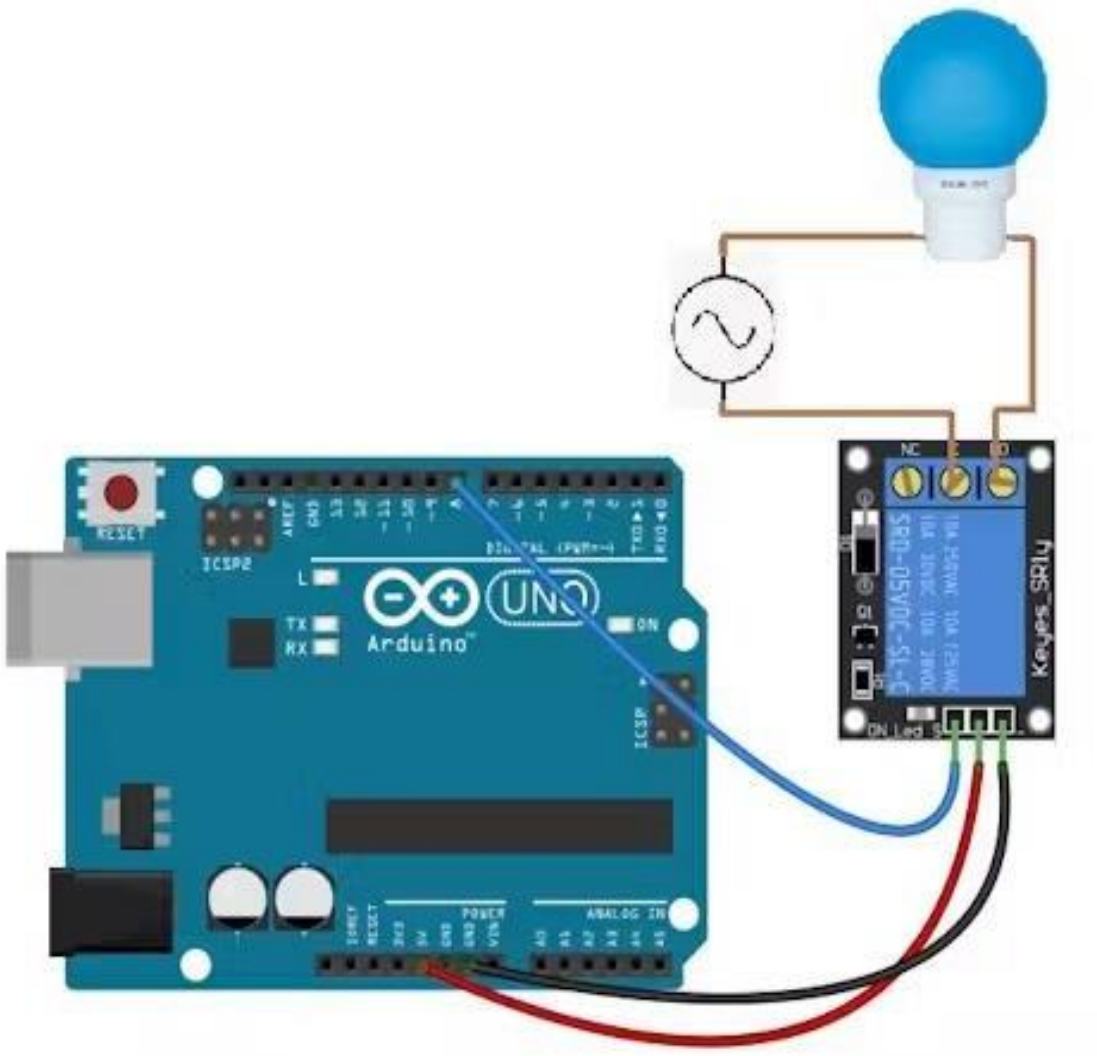
- Relay coil energizes
- Internal switch changes position
- Load turns ON or OFF

Relay terminals:

- **COM** → Common terminal
- **NO** → Normally Open
- **NC** → Normally Closed

For demonstration purposes, a low-voltage DC load such as an LED or DC bulb is used instead of high-voltage AC appliances.


NO (Normally Open)
GND
5V
COM
NC (Normally Closed)
Digital pin
SRD-05VDC-SL-C


Arduino
UNO
Keyes_SR1y

Program:

```
int relayPin = 8;

void setup()
{
    pinMode(relayPin, OUTPUT);
}

void loop()
{
    digitalWrite(relayPin, HIGH);   // Relay ON

    delay(2000);

    digitalWrite(relayPin, LOW);    // Relay OFF

    delay(2000);
}
```

Working:

- HIGH signal activates the relay.
- Relay contacts close.
- Connected load turns ON.
- LOW signal deactivates the relay.
- Load turns OFF.

Program-2:

The modified Arduino program uses a push button to toggle a relay ON and OFF hence external AC/DC load connected to relay.

For this connect push button to Arduino pin-2 and relay to Arduino pin-7.

```
const int relayPin = 7;
const int buttonPin = 2;

bool relayState = false;
bool lastButtonState = HIGH;

void setup()
{
    pinMode(relayPin, OUTPUT);

    pinMode(buttonPin, INPUT_PULLUP);

    digitalWrite(relayPin, LOW);
}
```

```
void loop()
{
    bool currentButtonState = digitalRead(buttonPin);

    // Detect button press
    if (lastButtonState == HIGH && currentButtonState == LOW)
    {
        relayState = !relayState;

        digitalWrite(relayPin, relayState);

        delay(200); // Debounce delay
    }

    lastButtonState = currentButtonState;
}
```

Working Principle:

1. The push button is connected to Digital Pin 2 using the Arduino's internal pull-up resistor (INPUT_PULLUP).
2. When the button is not pressed, the input pin reads HIGH.
3. When the button is pressed, the input pin reads LOW.
4. The Arduino detects the transition from HIGH to LOW (button press).
5. Each button press toggles the relay state:
    - First press → Relay ON
    - Second press → Relay OFF
    - Third press → Relay ON, and so on.
6. A 200 ms debounce delay prevents multiple toggles caused by switch bouncing.

Result:

The relay module was successfully interfaced with the microcontroller, and the connected AC/DC load was controlled successfully.

Conclusion:

The experiment demonstrated successful interfacing of a relay module with a microcontroller for controlling external AC/DC loads safely and effectively.

***Modify the program so that the AC/DC load connected to the relay toggles (ON and OFF continuously) only while a push button (switch) is pressed.**

Group-D	Experiment No.-14	
Title:	Interfacing DC motor using motor driver (L293D / L298) to control speed using Potentiometer. (Total Slots = 1)	
Perform Date:	Teacher Sign for attendance	Sign for completion with grade

Aim:
To interface a DC motor with an Arduino Uno using an L293D/L298N motor driver and control its speed using a potentiometer.

Objectives:
- To understand the working of a DC motor and motor driver.
- To interface a DC motor with Arduino using the L293D/L298N motor driver.
- To control the speed of the DC motor using a potentiometer.
- To learn PWM (Pulse Width Modulation) for speed control.

Apparatus Required:
- Arduino Uno
- L298N Motor Driver Module (or L293D Motor Driver IC)
- DC Motor
- 10 kΩ Potentiometer
- External 9–12 V DC power supply
- Breadboard (if using L293D IC)
- Jumper wires
- USB cable
- Computer with Arduino IDE

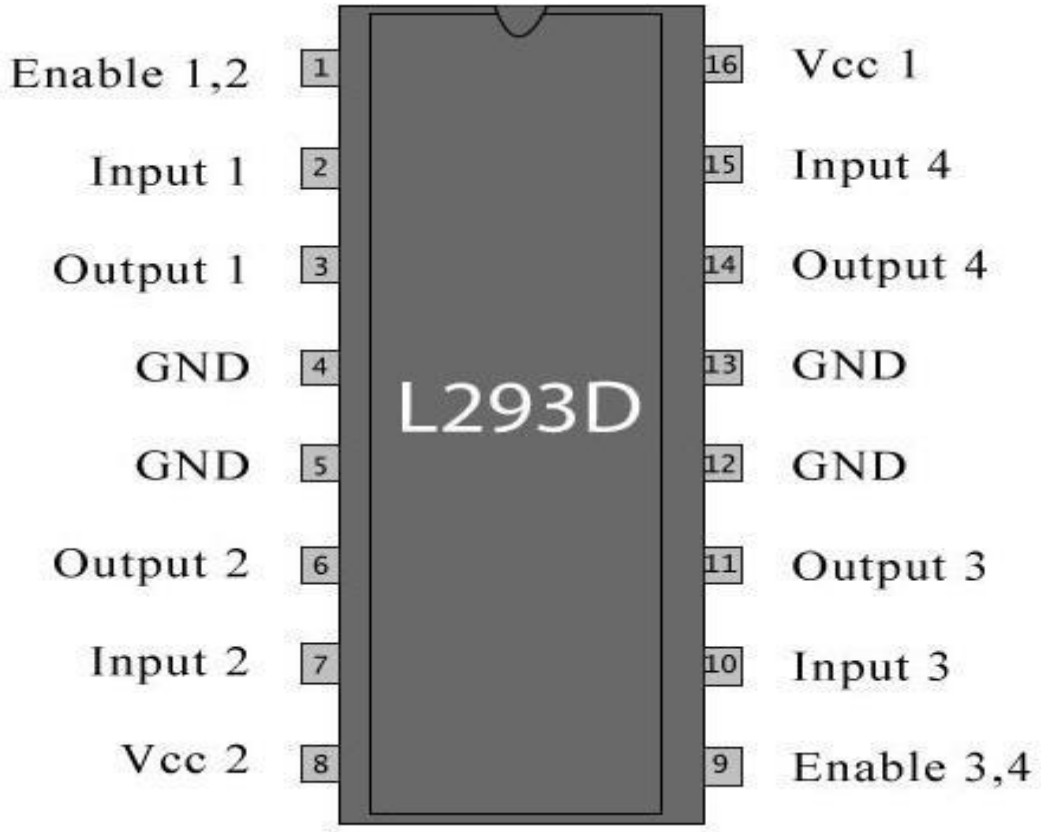
Theory:

A **DC motor** converts electrical energy into mechanical rotational motion. Since the Arduino cannot supply sufficient current to drive a motor directly, a **motor driver** such as the **L293D** or **L298N** is used. The motor driver acts as an interface between the Arduino and the motor, providing the required current and protecting the Arduino from high current.

The **potentiometer** acts as a variable resistor. As its knob is rotated, the voltage at the Arduino's analog input changes from **0 V to 5 V**. The Arduino reads this voltage using the `analogRead()` function and converts it into a value between **0 and 1023**.

This value is mapped to a PWM (Pulse Width Modulation) range of **0 to 255** using the `map()` function. The PWM signal is then applied to the **Enable (ENA)** pin of the motor driver using the `analogWrite()` function. By varying the PWM duty cycle, the average voltage applied to the motor changes, thereby controlling its speed.

	L293D	L298N
		
Max Supply Voltage	36V	40V
Peak Output Current	1.2A	2.5A
Continuous Output Current	0.6A	2A
Power Dissipation	5W	25W

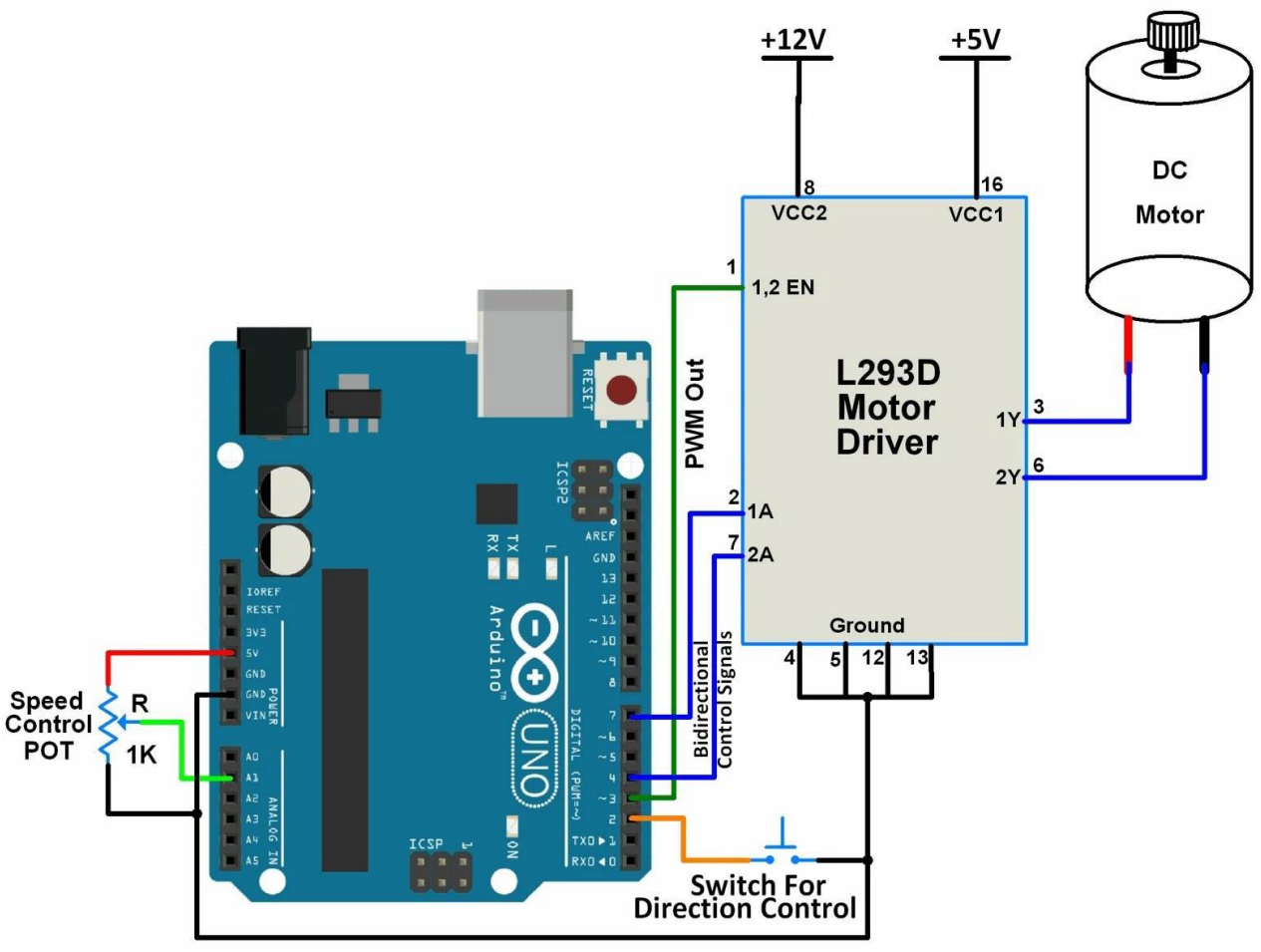


Circuit Connections:

L293D Pin	Function	Arduino Connection
Pin 1	Enable 1 (EN1)	D3 (PWM)
Pin 2	Input 1 (IN1)	D7
Pin 7	Input 2 (IN2)	D4
Pin 3	Output 1	Motor Terminal 1
Pin 6	Output 2	Motor Terminal 2
Pin 8	Motor Supply (Vcc2)	6–12 V External Supply
Pin 16	Logic Supply (Vcc1)	5 V
Pins 4, 5, 12, 13	Ground	GND

Potentiometer:

Potentiometer Pin	Arduino Connection
Left Terminal	5 V
Middle (Wiper)	A0
Right Terminal	GND



Program:

```
// L293D Motor Driver - Speed Control Using Potentiometer

const int potPin = A0;     // Potentiometer connected to A0
const int enablePin = 3;   // L293D Enable pin (PWM)
const int in1 = 7;         // Input 1(1A)
const int in2 = 4;         // Input 2(2A)

void setup()
{
  pinMode(enablePin, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  // Set motor direction (Forward)
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
}

void loop()
{
  // Read potentiometer value
  int potValue = analogRead(potPin);

  // Convert 0–1023 to 0–255 for PWM
  int motorSpeed = map(potValue, 0, 1023, 0, 255);

  // Control motor speed
  analogWrite(enablePin, motorSpeed);
}
```

This program only controls the speed of the motor. The motor direction is fixed in the forward direction by setting:

```
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
```

Rotating the potentiometer changes the PWM duty cycle, causing the motor speed to vary smoothly from 0% (stopped) to 100% (maximum speed).

How statement work?

```
int motorSpeed = map(potValue, 0, 1023, 0, 255);
```

This statement uses the Arduino **`map()`** function to convert the potentiometer reading into a PWM value suitable for motor speed control.

Syntax:

```
map(value, fromLow, fromHigh, toLow, toHigh);
```

Explanation of Parameters:

Parameter	Value	Description
`value`	potValue	Analog value read from the potentiometer
`fromLow`	0	Minimum analog input value
`fromHigh`	1023	Maximum analog input value
`toLow`	0	Minimum PWM output value
`toHigh`	255	Maximum PWM output value

Why is `map()` Used?

- The Arduino's **`analogRead()`** function returns values from **0 to 1023** (10-bit ADC).
- The **`analogWrite()`** function accepts PWM values from **0 to 255** (8-bit PWM).
- Therefore, the `map()` function converts the analog input range to the PWM output range.

Potentiometer Reading (potValue)	PWM Value (motorSpeed)	Motor Speed
0	0	Motor OFF
256	64	Low Speed
512	128	Medium Speed
768	191	High Speed
1023	255	Maximum Speed

Working:
Suppose the potentiometer is at the middle position:

```
potValue = 512;
```

After mapping:

```
motorSpeed = map(512, 0, 1023, 0, 255);
```

The result is approximately:

```
motorSpeed = 128;
```

then

```
analogWrite(enablePin, motorSpeed);
```

it sends a PWM signal with a duty cycle of about **50%**, causing the motor to run at approximately half speed.

*Modify the program to display motor speed on Serial monitor.

Group-D	Experiment No.-15	
Title:	**Controlling an RGB LED Using the BlinkM Module (I2C communication) (Total Slots = 1)**	
Perform Date:	**Teacher Sign for attendance**	**Sign for completion with grade**

Aim:

To interface the BlinkM RGB LED Module with an Arduino Uno using the I²C communication protocol and control the LED to display different colors.

Objectives:

- To understand the I²C communication protocol.
- To interface the BlinkM RGB LED module with the Arduino Uno.
- To control the RGB LED by generating different colors.
- To develop an Arduino program for I²C communication.

Apparatus Required:

- Arduino Uno
- BlinkM RGB LED Module
- Breadboard
- Jumper wires
- USB cable
- Computer with Arduino IDE

Theory:

The **BlinkM** is an intelligent RGB LED module that contains a microcontroller and an RGB LED. It communicates with the Arduino using the **I²C (Inter-Integrated Circuit)** protocol.

Unlike a standard RGB LED, the BlinkM module can receive commands over the I²C bus to change its color, brightness, and lighting effects without requiring separate PWM control for each LED.
The Arduino Uno communicates with the BlinkM through the **Wire library**, which implements the I²C protocol.

I²C Pins on Arduino Uno

Arduino Pin	Function
A4	SDA (Serial Data)
A5	SCL (Serial Clock)

The default I²C address of the BlinkM module is **0x09**.

Circuit Connections

BlinkM Pin	Arduino Uno
VCC	5V
GND	GND
SDA	A4
SCL	A5

About Wire Library:

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

Explanation:

- The #include directive tells the Arduino compiler to include the **Wire library** in the program.
- The **Wire library** is the standard Arduino library used for **I2C (Inter-Integrated Circuit)** communication.
- It allows the Arduino to communicate with I2C devices such as the **BlinkM RGB LED module**, LCD displays, EEPROMs, RTC modules, sensors, and many other peripherals.

Why is #include <Wire.h> Required?

- Enables I2C communication.
- Provides functions to send and receive data over the I2C bus.
- Simplifies communication with I2C devices.
- Supports communication with multiple devices using unique I2C addresses.

Function	Description
`Wire.begin();`	Initializes the I2C bus as a master device.
`Wire.beginTransmission(address);`	Starts communication with the I2C device at the specified address.
`Wire.write(data);`	Sends a byte or data to the I2C device.
`Wire.endTransmission();`	Ends the communication and transmits the data.
`Wire.requestFrom(address, bytes);`	Requests data from an I2C device.
`Wire.read();`	Reads a byte of data received from the I2C device.

Arduino Program:

```
#include <Wire.h>


#define BLINKM_ADDR 0x09


void setColor(byte r, byte g, byte b)
{
  Wire.beginTransmission(BLINKM_ADDR);
  Wire.write('n');      // Set RGB color immediately
  Wire.write(r);
  Wire.write(g);
  Wire.write(b);
  Wire.endTransmission();
}


void setup()
{
  Wire.begin();
}
```

```
void loop()
{
  setColor(255, 0, 0);     // Red
  delay(1000);

  setColor(0, 255, 0);     // Green
  delay(1000);

  setColor(0, 0, 255);     // Blue
  delay(1000);

  setColor(255, 255, 0);   // Yellow
  delay(1000);

  setColor(0, 255, 255);   // Cyan
  delay(1000);

  setColor(255, 0, 255);   // Magenta
  delay(1000);

  setColor(255, 255, 255); // White
  delay(1000);

  setColor(0, 0, 0);       // OFF
  delay(1000);
}
```

Working Principle:

1. The Arduino initializes the I²C communication using Wire.begin().
2. The BlinkM module is connected to the Arduino through the **SDA** and **SCL** lines.
3. The setColor() function sends RGB values to the BlinkM using the I²C protocol.
4. The BlinkM receives the command and changes the RGB LED color accordingly.
5. The program continuously cycles through different colors with a delay of one second.

Observation Table:

RGB Values	Displayed Color
(255, 0, 0)	Red
(0, 255, 0)	Green
(0, 0, 255)	Blue
(255, 255, 0)	Yellow
(0, 255, 255)	Cyan
(255, 0, 255)	Magenta
(255, 255, 255)	White
(0, 0, 0)	OFF

Result:

The BlinkM RGB LED module was successfully interfaced with the Arduino Uno using the I²C communication protocol. Different colors were displayed by transmitting RGB values over the I²C bus.

Applications

- Decorative lighting
- Mood lighting systems
- Status indication
- Home automation
- Robotics
- Smart lighting systems
- Interactive electronic projects

***Modify the program to program to display the color names on serial monitor.**

Group-D	Experiment No.-16	
Title:	**Serial communication using USB: data transmission between Arduino and PC (Total Slots = 1)**	
Perform Date:	**Teacher Sign for attendance**	**Sign for completion with grade**

Aim:
To establish serial communication between an Arduino board and a PC using USB cable and transmit/receive data successfully.

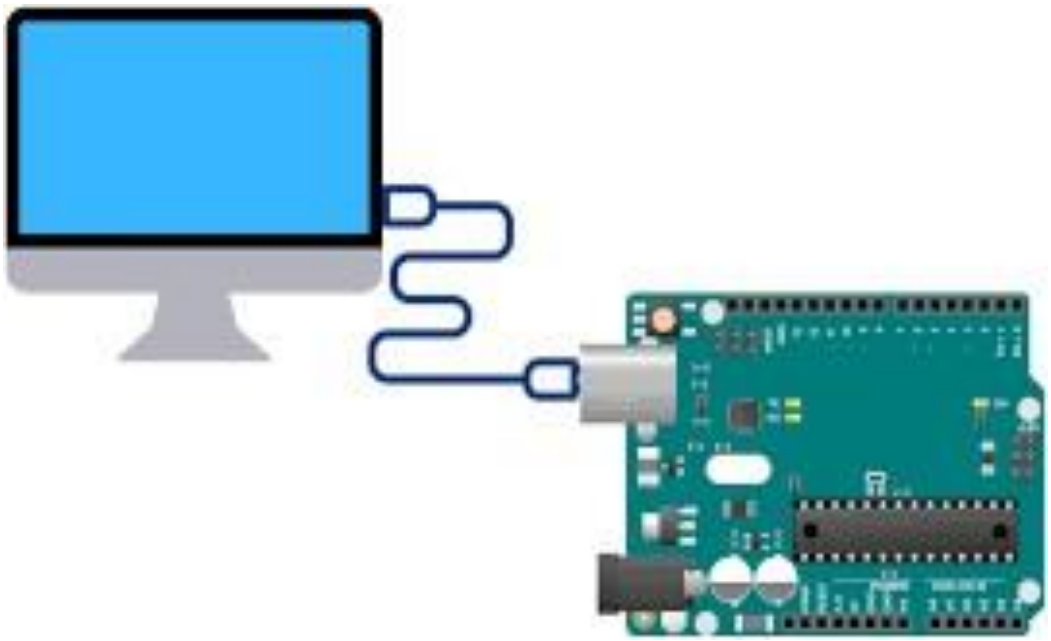
Objectives:
- To understand USB serial communication
- To send data from Arduino to PC
- To send data from PC to Arduino
- To observe serial data using Serial Monitor

Components Required
- Arduino Uno board
- LED (optional)
- PC/Laptop
- USB cable

Theory
Serial communication is a method of transferring data one bit at a time between devices. In Arduino, serial communication occurs through the USB interface using UART (Universal Asynchronous Receiver Transmitter).
The USB cable:
- Supplies power to the Arduino board.
- Transfers serial data between Arduino and the computer.

Communication takes place at a specific **baud rate** (e.g., 9600 bps). The Arduino IDE provides a **Serial Monitor** to view and send data.

Functions used	Descriptions
`Serial.begin(9600)`	Starts serial communication
`Serial.print() / Serial.println()`	Sends data to PC
`Serial.available()`	Checks incoming data
`Serial.read()`	Reads received data



Program 1: Sending Data from Arduino to PC

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

void loop()
{
  Serial.println("Hello from Arduino!");
 delay(1000);       // Wait for 1 second
}
```

Procedure

1. Connect Arduino to PC using USB cable.
2. Open the Arduino IDE.
3. Select the correct:
   - Board
   - COM Port
4. Upload the program.
5. Open **Serial Monitor**.
6. Set baud rate to **9600**.
7. Observe the message displayed every second.

The Serial Monitor continuously displays: `Hello from Arduino!`

Program 2: Receiving Data from PC

```
char data;

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

void loop()
{

  if (Serial.available() > 0)
 {
    data = Serial.read();

    Serial.print("Received: ");
    Serial.println(data);
  }
}
```

Procedure

1. Upload the program to Arduino.
2. Open Serial Monitor.
3. Type any character in the input box.
4. Press Enter/Send.
5. Observe the received character displayed back.

Expected Output

```
Received: A
Received: B
Received: 5
```

Result

Serial communication between the Arduino and PC using USB was successfully established, and data transmission/reception was observed through the Serial Monitor.

*Modify the program to send and receive the data between Arduino and PC.

SECTION - II

Activity

Student has to perform **any one Activity** which is equivalent to three experiments.

Activity-1	
Electronics Hobby Project	**Design and develop a small electronics-based project and submit a report.**

The report should include the following sections:

1. Title of the Hobby Project
2. Aim
3. Objectives
4. Components Required
5. Block Diagram
6. Circuit Diagram
7. Working Principle
8. Arduino Program
9. Results and Observations
10. Applications
11. Future Scope
12. Conclusion
13. References

OR

Activity-2	
Field Visit Report	**Visit an electronics industry and submit a detailed report**

Report Format

1. Name of the Industry
2. Date of Visit
3. Location
4. Objectives of the Visit
5. Brief Introduction of the Industry
6. Products Manufactured
7. Manufacturing Process
8. Equipment and Instruments Observed
9. Embedded Systems and Automation Use
10. Testing and Quality Control
11. Safety Practices Followed
12. Key Learning Outcomes
13. Challenges Observed
14. Suggestions and Recommendations
15. Conclusion
16. Photographs Paste photographs of the industry visit, manufacturing facilities, equipment, and group activities (where permitted).