SNJB's

KKHA Arts, SMGL Commerce & SPHJ Science College, Chandwad

Department of Electronic Science

Course Name: ELS-271-VSC, Circuit Simulation-II

INDEX

Sr. No.	Title of the Experiment	Date	Remark
1	LED Interfacing with Arduino		
2	LED Array Interfacing with Arduino		
3	Traffic Light Controller		
4	Turn ON/OFF an LED using a push button		
5 & 6	Interfacing 7 segment display and 4x4 Keypad to Arduino		
7	Temperature sensor (LM35) interfacing to Arduino		
8	LCD interfacing to Arduino		
9	Digital Thermometer using LM35 and LCD		
10	Measure distance using Ultrasonic sensor and display it on Serial Monitor or LCD		
11	DC motor speed control using PWM		
12	Interfacing Servo Motor with Arduino		

CERTIFICATE

This is to certify that Mr. /Miss._____ _____ of S.Y.B.Sc. Class, Exam Seat No._____has completed required number of practical in the Electronic Science Laboratory during the academic year 2015– 2026. His / her performance is/not satisfactory**.**

Practical in charge **H.O.D**

Internal **Examiner**

External **Examiner**

Title: LED Interfacing with Arduino

1. Aim

To interface an LED with Arduino and make it blink at regular intervals.

---

2. Apparatus Required

- Arduino Uno board
- LED (Light Emitting Diode)
- Resistor (220Ω or 330Ω)
- Breadboard
- Connecting wires (jumper wires)
- USB cable (for programming & power)

---

3. Theory

An LED (Light Emitting Diode) is a semiconductor device that emits light when current flows through it.

Arduino digital pins can be used to control external components like LEDs. When a digital pin is set to:

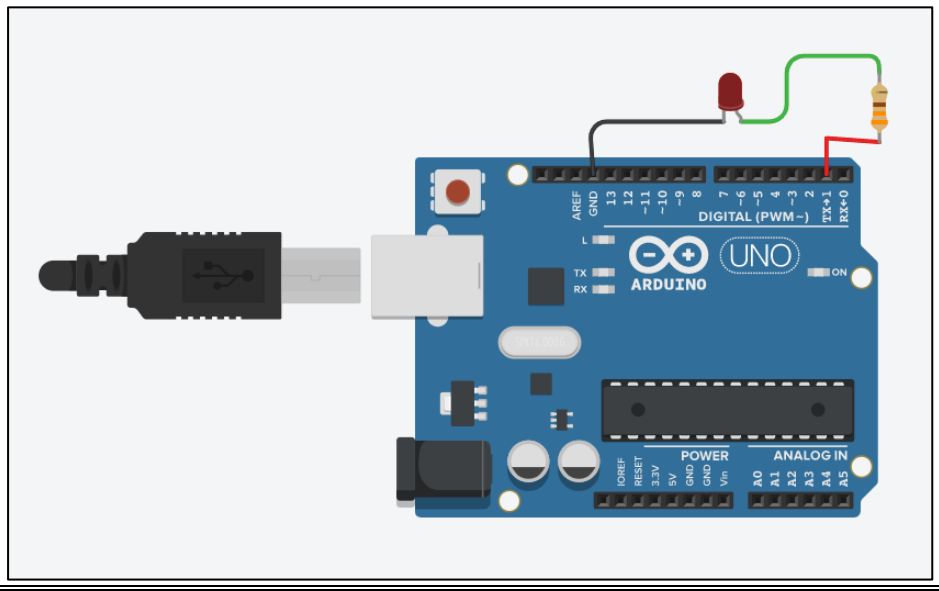
- **HIGH** → 5V is supplied → LED turns ON
- **LOW** → 0V → LED turns OFF

A **current-limiting resistor** is necessary to protect the LED from excessive current.

---

4. Circuit Diagram (Description)

- Connect **LED anode (long leg)** to **pin 1** of Arduino through a resistor.
- Connect **LED cathode (short leg)** to **GND** of Arduino.



5. Program (Code)

```
void setup()
{
  pinMode(1, OUTPUT);
}

void loop()
{
  digitalWrite(1, HIGH);
  delay(5000); // Wait for 5 seconds
  digitalWrite(1, LOW);
  delay(5000); // Wait for 5 seconds
}
```

6. Explanation of Code

- `pinMode(1, OUTPUT);` → Sets pin 1 as output pin
- `digitalWrite(1, HIGH);` → Turns LED ON
- `delay(5000);` → Waits for 5 seconds
- `digitalWrite(1, LOW);` → Turns LED OFF
- Loop repeats continuously

---

7. Procedure

1. Connect the circuit as described.
2. Open Arduino IDE.
3. Write or paste the program.
4. Select the correct board and port.
5. Upload the program to Arduino.
6. Observe the LED behavior.

---

8. Observation

- LED turns ON for 5 seconds
- LED turns OFF for 5 seconds
- This cycle repeats continuously

---

9. Result

The LED was successfully interfaced with Arduino and made to blink at a 5-second interval.

---

10. Precautions

- Always use a resistor with LED
- Check polarity of LED before connecting
- Ensure proper connections on breadboard
- Avoid short circuits

---

11. Applications

- Indicator systems
- Signal lights
- Basic embedded system learning
- Home automation prototypes

Title: LED Array Interfacing with Arduino

1. Aim

To interface an 8-LED array with Arduino and display binary numbers from 1 to 255.

---

2. Apparatus Required

- Arduino Uno board
- 8 LEDs
- 8 Resistors (220Ω or 330Ω)
- Breadboard
- Jumper wires
- USB cable

---

3. Theory

An LED array consists of multiple LEDs connected to different digital pins of Arduino. Each LED represents a **binary bit**:

- LED OFF → 0
- LED ON → 1

The Arduino can display numbers in **binary format** by controlling multiple LEDs simultaneously.

For example:

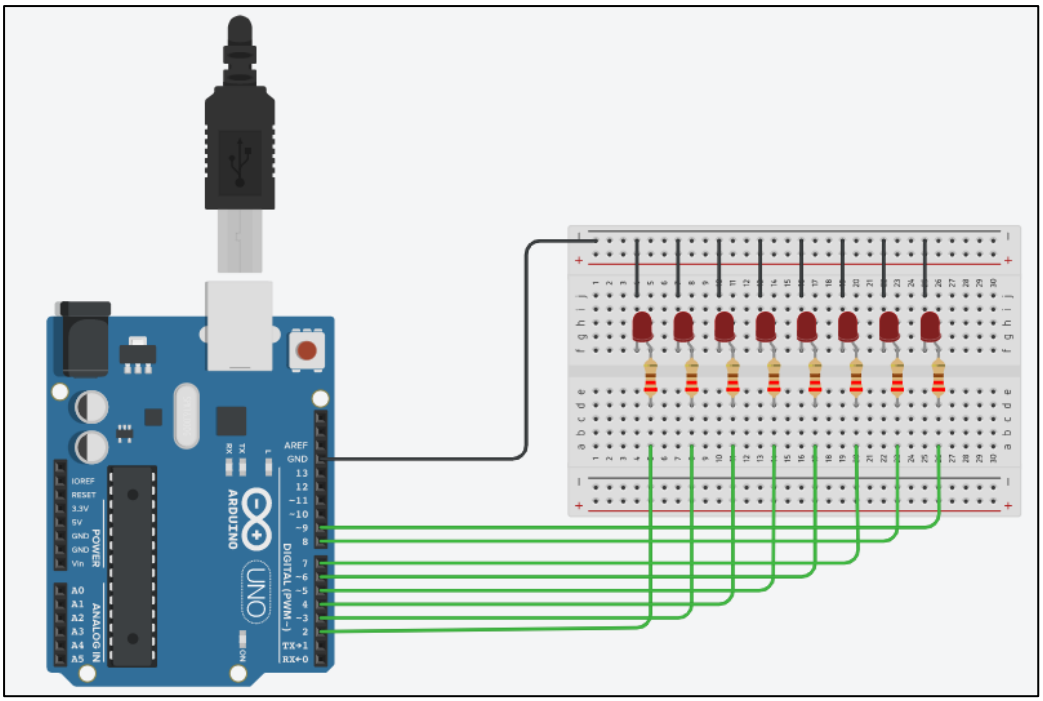
- Decimal 5 → Binary `00000101` → LEDs at positions 0 and 2 ON

The program uses **bitwise operations** (`>>` and `&`) to extract individual bits from a number and display them on LEDs.

---

4. Circuit Diagram (Description)

- Connect 8 LEDs to Arduino pins **2 to 9** respectively.
- Each LED anode → connected to Arduino pin through a resistor
- Each LED cathode → connected to GND



5. Program (Code)

```
int ledPins[] = {2, 3, 4, 5, 6, 7, 8, 9};

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  for (int value = 1; value < 256; value++) {
    displayBinary(value);
    delay(1000);
  }
}

void displayBinary(int number) {
  for (int i = 0; i < 8; i++) {
    int bit = (number >> i) & 1;
    digitalWrite(ledPins[i], bit);
  }
}
```

6. Explanation of Code

- `ledPins[]` → Stores pin numbers for 8 LEDs
- `pinMode()` → Sets all pins as output
- Loop runs from **1 to 255** (8-bit values)
- `displayBinary(value)` → Converts number into binary

- `(number >> i) & 1` → Extracts each bit
- `digitalWrite()` → Turns corresponding LED ON/OFF

---

7. Procedure

1. Connect 8 LEDs to Arduino pins 2–9 using resistors.
2. Connect all LED cathodes to GND.
3. Open Arduino IDE.
4. Enter the given code.
5. Select correct board and port.
6. Upload the program.
7. Observe LED pattern changes every second.

---

8. Observation

- LEDs display binary representation of numbers
- Count increases from 1 to 255
- LEDs change pattern every 1 second

Example:

- 1 → `00000001`
- 2 → `00000010`
- 3 → `00000011`

---

9. Result

The LED array was successfully interfaced with Arduino and used to display binary counting from 1 to 255.

---

10. Precautions

- Use current-limiting resistors with each LED
- Ensure correct polarity of LEDs
- Avoid loose connections
- Verify correct pin connections

---

11. Applications

- Binary counters, Digital display systems, Learning bitwise operations etc.

Title: Traffic Light Controller

1. Aim

To design and implement a traffic light control system for four lanes using Arduino.

---

2. Apparatus Required

- Arduino Uno board
- 12 LEDs (Red, Yellow, Green for each lane)
- 12 Resistors (220Ω or 330Ω)
- Breadboard
- Jumper wires
- USB cable

---

3. Theory

A traffic light system controls vehicle movement using three signals:

- **Red** → Stop
- **Yellow** → Wait
- **Green** → Go

In this project, four lanes are controlled sequentially using Arduino. Each lane has three LEDs representing the traffic signals.

The system works in a loop where:

1. One lane gets **Green**, others stay **Red**
2. Yellow signal provides transition delay
3. Control shifts to next lane

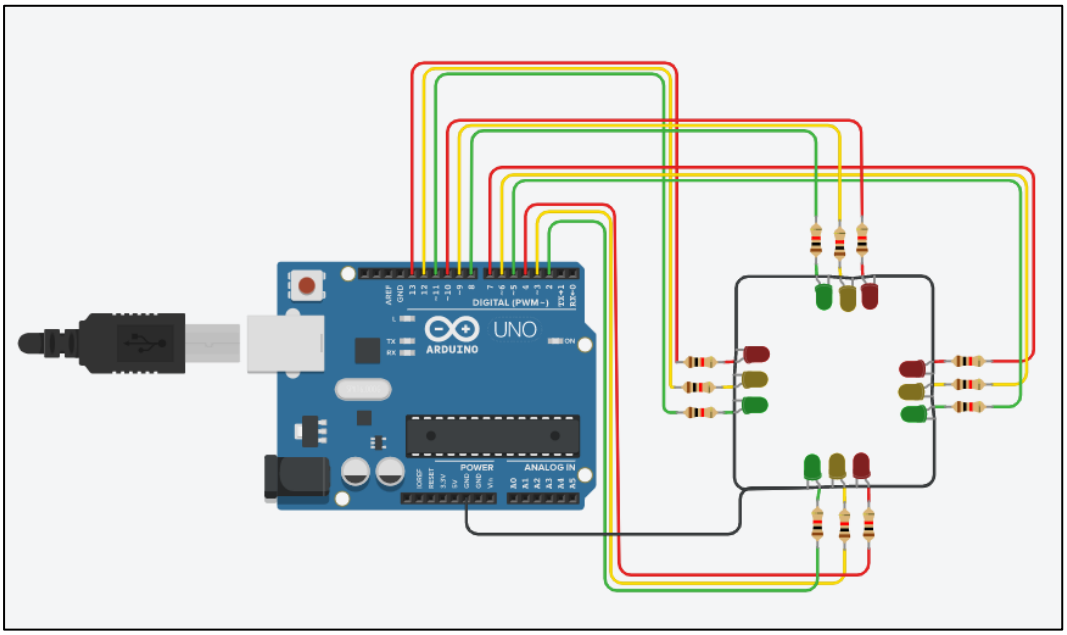
This mimics a real-world traffic junction system.

4. Circuit Diagram (Description)

- Each lane has 3 LEDs:
  - Red, Yellow, Green
- Connections:

Lane	Red	Yellow	Green
Lane 1	Pin 13	Pin 12	Pin 11
Lane 2	Pin 10	Pin 9	Pin 8
Lane 3	Pin 7	Pin 6	Pin 5
Lane 4	Pin 4	Pin 3	Pin 2

- Each LED anode → Arduino pin through resistor
- Each LED cathode → GND



5. Program (Code)

```
int Lane1[] = {13,12,11};
int Lane2[] = {10,9,8};
int Lane3[] = {7,6,5};
int Lane4[] = {4,3,2};

void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(Lane1[i], OUTPUT);
    pinMode(Lane2[i], OUTPUT);
    pinMode(Lane3[i], OUTPUT);
    pinMode(Lane4[i], OUTPUT);
  }

  for (int i = 0; i < 3; i++)
  {
    digitalWrite(Lane1[i], LOW);
    digitalWrite(Lane2[i], LOW);
    digitalWrite(Lane3[i], LOW);
    digitalWrite(Lane4[i], LOW);
```

```
  }
}

void loop()
{
  digitalWrite(Lane1[2], HIGH);
  digitalWrite(Lane2[0], HIGH);
  digitalWrite(Lane3[0], HIGH);
  digitalWrite(Lane4[0], HIGH);

  delay(7000);

  digitalWrite(Lane1[2], LOW);
  digitalWrite(Lane1[1], HIGH);
  delay(3000);

  digitalWrite(Lane1[1], LOW);
  digitalWrite(Lane1[0], HIGH);
  digitalWrite(Lane2[2], HIGH);
  delay(7000);

  digitalWrite(Lane2[2], LOW);
  digitalWrite(Lane2[1], HIGH);
  delay(3000);

  digitalWrite(Lane2[1], LOW);
  digitalWrite(Lane2[0], HIGH);
  digitalWrite(Lane3[2], HIGH);
  delay(7000);

  digitalWrite(Lane3[2], LOW);
  digitalWrite(Lane3[1], HIGH);
  delay(3000);

  digitalWrite(Lane3[1], LOW);
  digitalWrite(Lane3[0], HIGH);
  digitalWrite(Lane4[2], HIGH);
  delay(7000);

  digitalWrite(Lane4[2], LOW);
  digitalWrite(Lane4[1], HIGH);
  delay(3000);

  digitalWrite(Lane4[1], LOW);
  digitalWrite(Lane4[0], HIGH);
  digitalWrite(Lane1[2], HIGH);
}
```

---

6. Explanation of Code

- Arrays `Lane1[]` to `Lane4[]` store pins for Red, Yellow, Green LEDs
- `setup()` initializes all pins as OUTPUT
- Initially, all LEDs are turned OFF
- In `loop()`:
  - One lane gets GREEN signal
  - Others remain RED
  - YELLOW signal acts as transition
  - Each lane operates for:
    - Green → 7 seconds

    - Yellow → 3 seconds
- Cycle repeats continuously

---

7. Procedure

1. Connect LEDs for all four lanes as per pin configuration
2. Use resistors with each LED
3. Open Arduino IDE
4. Enter the given program
5. Select correct board and port
6. Upload the code
7. Observe traffic light sequence

---

8. Observation

- Only one lane shows GREEN at a time
- Other lanes remain RED
- YELLOW signal appears before switching
- Sequence rotates across all four lanes

---

9. Result

The traffic light controller system was successfully implemented using Arduino, controlling four lanes sequentially.

---

10. Precautions

- Ensure correct LED polarity
- Use resistors to avoid damage
- Check pin connections carefully
- Avoid short circuits

---

11. Applications

- Traffic signal systems
- Smart city projects
- Road safety control
- Embedded system simulations

Title: Turn ON/OFF an LED using a push button

1. Aim

To control an LED using a push button with Arduino.

---

2. Apparatus Required

- Arduino Uno board
- LED
- Push button (switch)
- Resistor (220Ω or 330Ω for LED)
- 10kΩ resistor (for pull-down or pull-up) *(optional depending on wiring)*
- Breadboard
- Jumper wires
- USB cable

---

3. Theory

A push button is a digital input device that can be either:

- **HIGH (1)** → Button released (depending on circuit)
- **LOW (0)** → Button pressed

Arduino reads the button state using `digitalRead()`. Based on this input:

- LED can be turned ON or OFF

In this program:

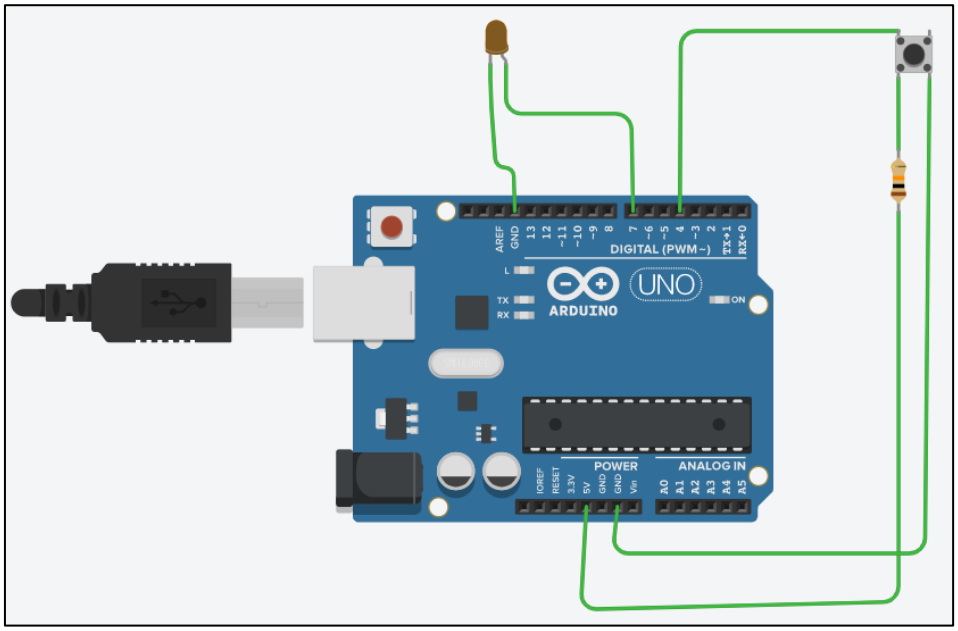
- When button is **pressed → LED turns ON**
- When button is **released → LED turns OFF**

This demonstrates **digital input-output interfacing**.

---

4. Circuit Diagram (Description)

- Connect **LED anode** → Pin 7 through resistor
- Connect **LED cathode** → GND
- Connect one side of push button → Pin 4
- Connect other side → GND (pull-down configuration assumed)



5. Program (Code)

```
int LED = 7;
int buttonpin = 4;

int buttonstate = 0;

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(buttonpin, INPUT);
}

void loop()
{
  buttonstate = digitalRead(buttonpin);

  if (buttonstate == HIGH)
  {
    digitalWrite(LED, LOW);
  }
  else
  {
    digitalWrite(LED, HIGH);
  }
}
```

---

6. Explanation of Code

- `LED = 7` → LED connected to pin 7
- `buttonpin = 4` → Button connected to pin 4

- `digitalRead()` → Reads button state
- If button is **HIGH** → LED OFF
- If button is **LOW** → LED ON
- `digitalWrite()` → Controls LED

---

7. Procedure

1. Connect LED and push button as per circuit
2. Open Arduino IDE
3. Write or paste the program
4. Select correct board and port
5. Upload the code
6. Press and release the button
7. Observe LED behavior

---

8. Observation

- LED turns ON when button is pressed
- LED turns OFF when button is released

---

9. Result

The LED was successfully controlled using a push button with Arduino.

---

10. Precautions

- Use resistor with LED
- Ensure proper button connections
- Avoid loose wiring
- Check polarity of LED

---

11. Applications

- Switching devices
- User input systems
- Home automation
- Control panels

Title: Interfacing 7 segment display and 4x4 Keypad to Arduino

1. Aim

To interface a **4×4 keypad** and a **7-segment display** with Arduino and display the pressed key on the 7-segment display.

---

2. Apparatus Required

- Arduino Uno board
- 4×4 Matrix Keypad
- 7-Segment Display (Common Cathode)
- 7 Resistors (220Ω or 330Ω)
- Breadboard
- Jumper wires
- USB cable

---

3. Theory

A **4×4 keypad** consists of 16 keys arranged in rows and columns. When a key is pressed, a connection is established between a specific row and column, which is detected by the Arduino using scanning.

A **7-segment display** is used to display digits (0–9) and some alphabets (A–F). It consists of 7 LEDs (segments a–g).

Working:

- Keypad sends input to Arduino
- Arduino processes the key
- Corresponding segments are activated to display the character

4. Circuit Diagram (Description)

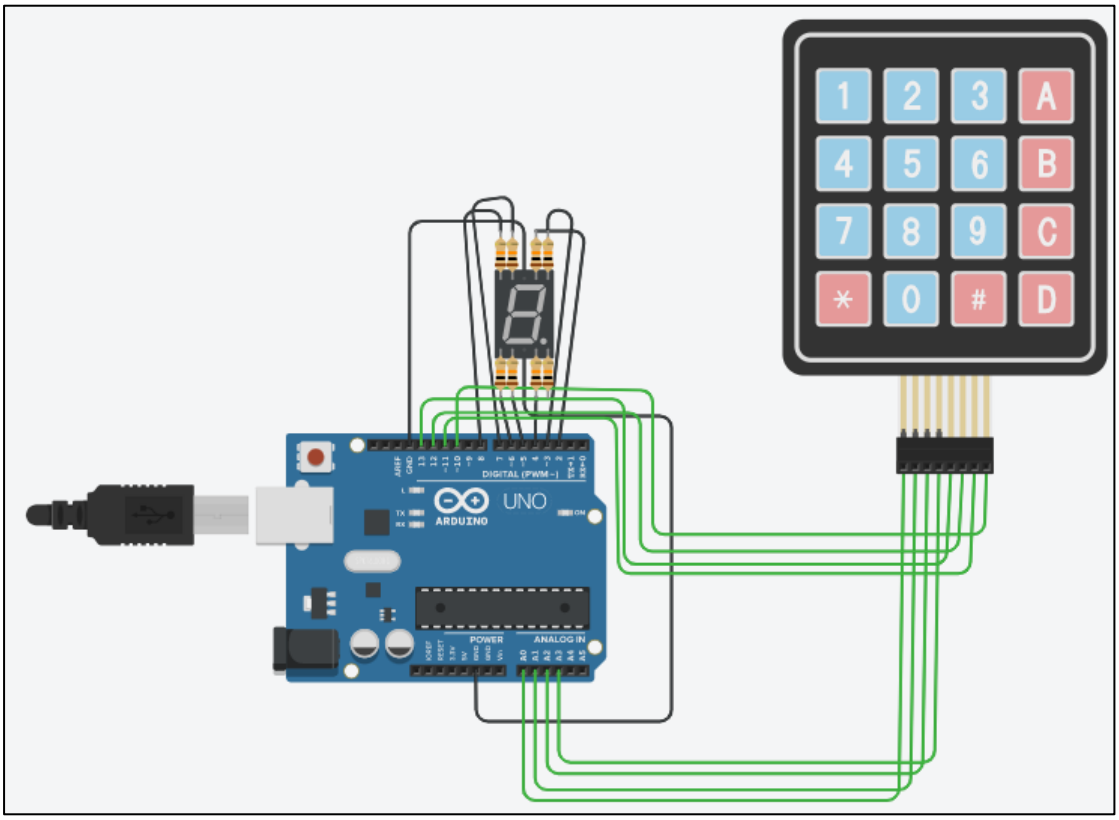
Keypad Connections

- Row pins → A0, A1, A2, A3
- Column pins → 13, 12, 11, 10

7-Segment Connections

Segment	Arduino Pin
a	2
b	3
c	4
d	5
e	6
f	7
g	8

- Common cathode → GND
- Each segment connected through resistor



5. Program (Code)

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
```

```
char hexaKeys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};

byte rowPins[ROWS] = {A0,A1,A2,A3};
byte colPins[COLS] = {13,12,11,10};

Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins,
ROWS, COLS);

void setup()
{
  for(int i=2;i<=8;i++)
    pinMode(i,OUTPUT);

  Serial.begin(9600);
}

// Example function
void one(){
 digitalWrite(2,0); digitalWrite(3,1); digitalWrite(4,1);
 digitalWrite(5,0); digitalWrite(6,0); digitalWrite(7,0);
 digitalWrite(8,0);
}

void loop()
{
  char key = customKeypad.getKey();

  if (key)
  {
    Serial.println(key);

    if(key=='1') one();
    else if(key=='2') two();
    else if(key=='3') three();
    else if(key=='4') four();
    else if(key=='5') five();
    else if(key=='6') six();
    else if(key=='7') seven();
    else if(key=='8') eight();
    else if(key=='9') nine();
    else if(key=='0') zero();
    else if(key=='A') leta();
    else if(key=='B') letb();
    else if(key=='C') letc();
    else if(key=='D') letd();
  }
}
```

(All digit/letter functions are defined in the original program.)

---

6. Explanation of Code

- `Keypad.h` → Library for keypad interfacing
- `hexaKeys[][]` → Stores keypad layout

- `rowPins[]` & `colPins[]` → Define connections
- `getKey()` → Reads pressed key
- Functions like `one()`, `two()` etc. → Control segments
- `digitalWrite()` → Turns segments ON/OFF

---

7. Procedure

1. Connect keypad and 7-segment display as per circuit
2. Use resistors with each segment
3. Open Arduino IDE
4. Install **Keypad library** if not available
5. Enter the given code
6. Select board and port
7. Upload the code
8. Press keys on keypad and observe display

---

8. Observation

- Pressed key is detected by Arduino
- Corresponding number/letter is displayed on 7-segment
- Serial Monitor shows pressed key

---

9. Result

The 4×4 keypad and 7-segment display were successfully interfaced with Arduino, and key presses were displayed correctly.

---

10. Precautions

- Ensure correct wiring of keypad rows and columns
- Use resistors for 7-segment display
- Check common cathode/anode type
- Avoid loose connections

---

11. Applications

- Password-based systems
- Digital keypads
- Electronic voting machines
- Calculator interfaces
- Access control systems

Title: Temperature sensor (LM35) interfacing to Arduino

1. Aim

To interface an **LM35 temperature sensor** with Arduino and measure temperature in Celsius.

---

2. Apparatus Required

- Arduino Uno board
- LM35 temperature sensor
- Breadboard
- Jumper wires
- USB cable

---

3. Theory

The **LM35** is a precision temperature sensor that provides output voltage linearly proportional to temperature.

- Output scale: **10 mV per °C**
- Example:
  - 25°C → 250 mV
  - 30°C → 300 mV

Arduino reads analog voltage using its **ADC (Analog-to-Digital Converter)**:

- Resolution: 10-bit (0–1023)
- Voltage range: 0–5V

Temperature calculation:

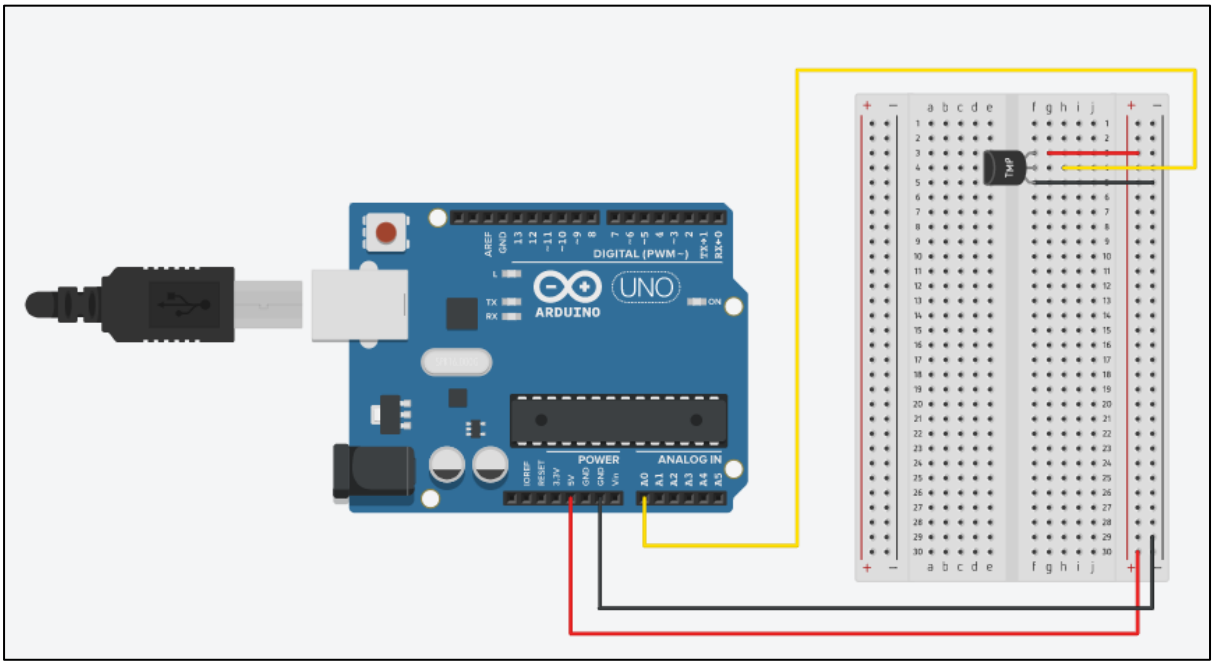
- Convert analog value → voltage
- Convert voltage → temperature

4. Circuit Diagram (Description)

LM35 has 3 pins:

Pin	Connection
VCC	5V (Arduino)
OUT	A0 (Arduino)
GND	GND (Arduino)

(Keep flat side of LM35 facing you: Left = VCC, Middle = OUT, Right = GND)



5. Program (Code)

```
int LM35Pin = A0;

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

void loop()
{
  int val = analogRead(LM35Pin);

  float voltage = val * (5.0 / 1023.0);
  float tempC = voltage * 100.0;

  Serial.print("Temp: ");
  Serial.print(tempC, 2);
  Serial.println(" C");

  delay(500);
}
```

6. Explanation of Code

- `analogRead(A0)` → Reads sensor value (0–1023)
- `voltage = val * (5.0 / 1023.0)` → Converts to voltage
- `tempC = voltage * 100` → Converts to temperature
- `Serial.print()` → Displays temperature on Serial Monitor
- `delay(500)` → Updates every 0.5 seconds

---

7. Procedure

1. Connect LM35 sensor to Arduino as per diagram
2. Open Arduino IDE
3. Enter the given code
4. Select correct board and port
5. Upload the code
6. Open Serial Monitor
7. Observe temperature readings

---

8. Observation

- Temperature is displayed in Celsius
- Readings update every 0.5 seconds
- Temperature changes when sensor is heated/cooled

---

9. Result

The LM35 temperature sensor was successfully interfaced with Arduino, and temperature was measured accurately in Celsius.

---

10. Precautions

- Ensure correct pin connections of LM35
- Do not reverse VCC and GND
- Avoid touching sensor pins directly
- Use stable power supply

---

11. Applications

- Weather monitoring systems
- Temperature control systems
- Industrial automation, Home automation, Medical devices

Title: LCD interfacing to Arduino

1. Aim

To interface a **16×2 LCD display** with Arduino and display messages on it.

---

2. Apparatus Required

- Arduino Uno board
- 16×2 LCD display
- Potentiometer (10kΩ) *(for contrast control)*
- Breadboard
- Jumper wires
- USB cable

---

3. Theory

A **16×2 LCD (Liquid Crystal Display)** is used to display text data. It has:

- 16 columns and 2 rows
- Each character is displayed in a 5×7 pixel matrix

The LCD operates in **4-bit or 8-bit mode**. In this experiment, **4-bit mode** is used to save Arduino pins.

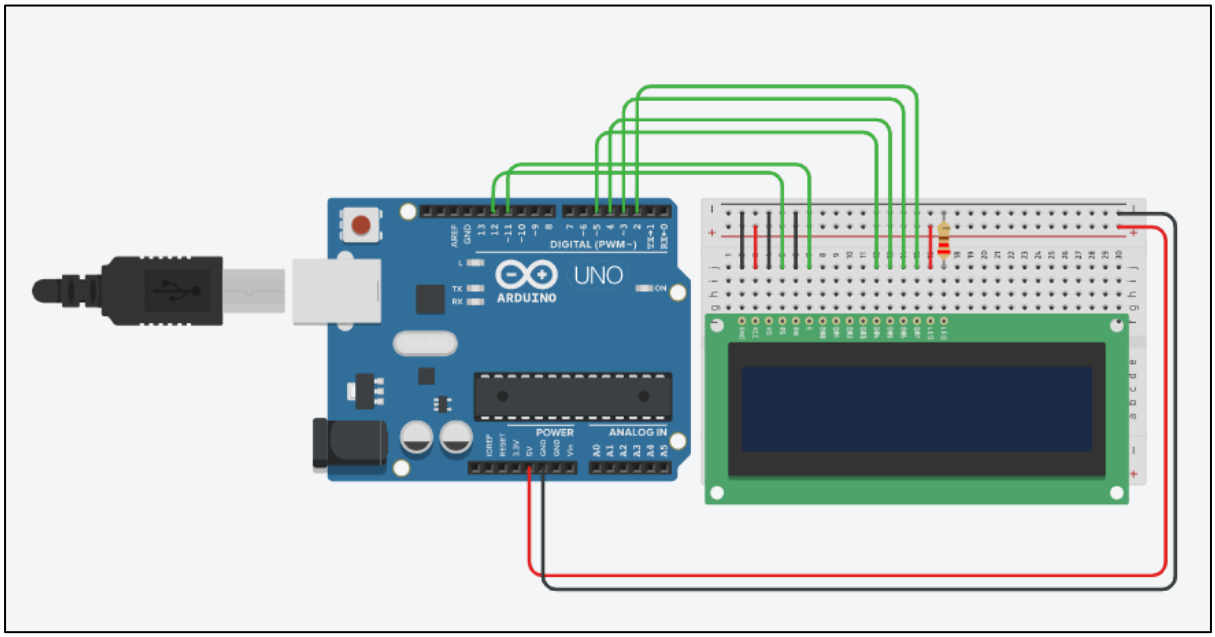
The **LiquidCrystal library** allows easy control of the LCD by sending commands and data.

4. Circuit Diagram (Description)

LCD Pin Connections

LCD Pin	Connection
VSS	GND
VDD	5V
V0	Middle pin of potentiometer
RS	Pin 12
RW	GND
E	Pin 11
D4	Pin 5
D5	Pin 4
D6	Pin 3
D7	Pin 2
A (LED+)	5V
K (LED-)	GND

- Potentiometer ends → 5V and GND
- Used to adjust display contrast



5. Program (Code)

```
#include<LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
 lcd.begin(16, 2);
}

void loop()
{
  lcd.setCursor(0,0);
```

```
  lcd.print("Hello, Students!");

  delay(1000);

  lcd.clear();

  lcd.setCursor(0,0);
  lcd.print("Welcome!");

  lcd.setCursor(0,1);
  lcd.print("To ELECTRONICS");

  delay(1000);

  lcd.clear();
}
```

---

6. Explanation of Code

- `LiquidCrystal lcd(12,11,5,4,3,2);` → Defines control pins
- `lcd.begin(16,2);` → Initializes LCD with 16 columns & 2 rows
- `lcd.setCursor(col,row);` → Sets cursor position
- `lcd.print()` → Displays text
- `lcd.clear()` → Clears screen
- `delay()` → Controls display timing

---

7. Procedure

1. Connect LCD to Arduino as per circuit
2. Connect potentiometer for contrast adjustment
3. Open Arduino IDE
4. Enter the given code
5. Select correct board and port
6. Upload the program
7. Adjust potentiometer to view display
8. Observe messages on LCD

---

8. Observation

- LCD displays "Hello, Students!"
- Then displays:
  - Line 1: "Welcome!"
  - Line 2: "To ELECTRONICS"
- Messages alternate every second

---

9. Result

The 16×2 LCD was successfully interfaced with Arduino and used to display text messages.

---

10. Precautions

- Adjust contrast using potentiometer
- Ensure correct pin connections
- Avoid loose wiring
- Do not reverse power supply pins

---

11. Applications

- Digital displays
- Information systems
- Embedded projects
- Home automation interfaces
- Industrial control panels

Title: Digital Thermometer using LM35 and LCD

1. Aim

To design and implement a **digital thermometer** using an LM35 temperature sensor and a 16×2 LCD with Arduino.

---

2. Apparatus Required

- Arduino Uno board
- LM35 temperature sensor
- 16×2 LCD display
- Potentiometer (10kΩ)
- Breadboard
- Jumper wires
- USB cable

---

3. Theory

The **LM35** temperature sensor provides an output voltage proportional to temperature:

- 10 mV per °C

Arduino reads this analog voltage using its ADC and converts it into temperature.

A **16×2 LCD** is used to display the temperature in:

- Celsius (°C)
- Fahrenheit (°F)

Conversion formulas:

- Celsius → $T(°C)=(Vout−0.5)×100T(°C) = (V_{out} - 0.5) \times 100T(°C)=(Vout −0.5)×100$ *(as used in code)*
- Fahrenheit → $T(°F)=T(°C)×1.8+32T(°F) = T(°C) \times 1.8 + 32T(°F)=T(°C)×1.8+32$

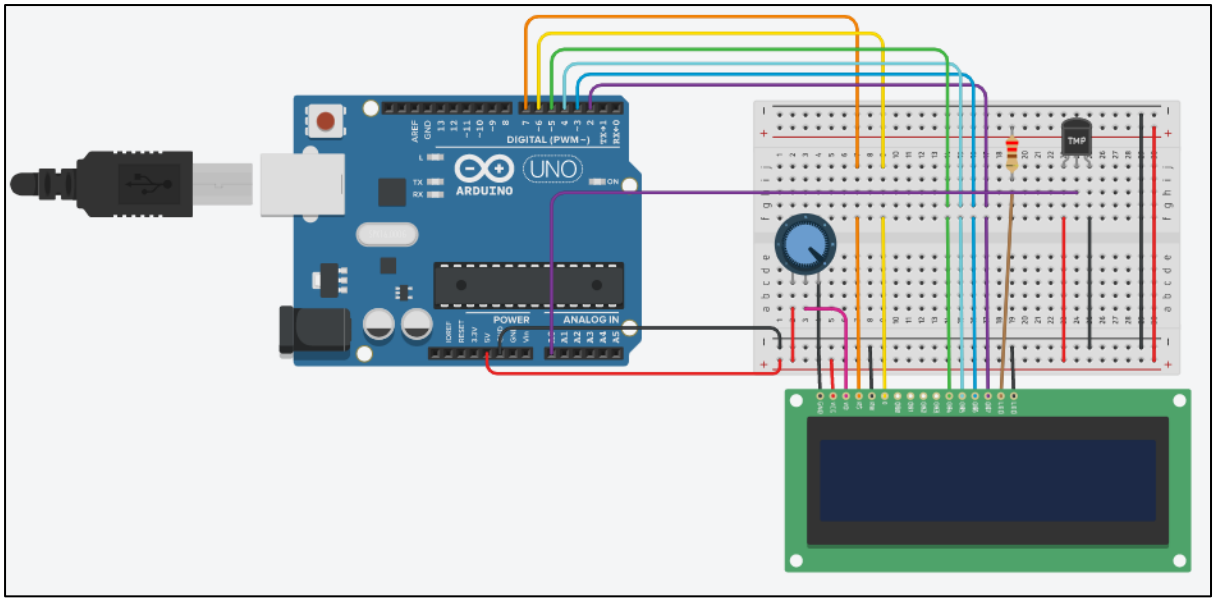
4. Circuit Diagram (Description)

LM35 Connections

Pin	Connection
VCC	5V
OUT	A0
GND	GND

LCD Connections

LCD Pin	Arduino Pin
RS	7
E	6
D4	5
D5	4
D6	3
D7	2
VSS	GND
VDD	5V
V0	Potentiometer



5. Program (Code)

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(7,6,5,4,3,2);
int temp = A0;

void setup() {
  lcd.begin(16, 2);
  pinMode(temp, INPUT);
  lcd.print("Chandwad Temp.");
}
```

```
void loop() {
  int a = analogRead(temp);
  float volt = a * 5.00 / 1023.00;
  float tempc = (volt - 0.5) * 100;
  float tempf = tempc * 1.8 + 32;

  lcd.setCursor(0,1);
  lcd.print(tempc);

  lcd.setCursor(7,1);
  lcd.print("&");

  lcd.setCursor(10,1);
  lcd.print(tempf);
}
```

---

6. Explanation of Code

- `analogRead(A0)` → Reads LM35 output
- `volt = a * 5 / 1023` → Converts to voltage
- `tempc = (volt - 0.5) * 100` → Calculates temperature in °C
- `tempf = tempc * 1.8 + 32` → Converts to °F
- `lcd.print()` → Displays temperature on LCD

---

7. Procedure

1. Connect LM35 sensor to Arduino
2. Connect LCD with potentiometer
3. Open Arduino IDE
4. Enter the code
5. Select correct board and port
6. Upload the program
7. Adjust contrast using potentiometer
8. Observe temperature on LCD

---

8. Observation

- LCD displays temperature in Celsius and Fahrenheit
- Temperature updates continuously
- Values change when sensor is heated or cooled

---

9. Result

The digital thermometer using LM35 and LCD was successfully implemented, displaying temperature in both Celsius and Fahrenheit.

---

10. Precautions

- Ensure correct LM35 pin connections
- Adjust LCD contrast properly
- Avoid loose connections
- Use stable power supply

---

11. Applications

- Digital thermometers
- Weather monitoring systems
- Industrial temperature control
- Home automation
- Medical devices

Title: Measure distance using Ultrasonic sensor and display it on Serial Monitor or LCD

Aim:

To measure the distance using an ultrasonic sensor and display the measured value on the Serial Monitor and LCD display.

Apparatus Required:

- Arduino Uno
- Ultrasonic Sensor (HC-SR04)
- 16x2 LCD Display
- Breadboard
- Connecting wires
- USB cable

Theory:

The ultrasonic sensor (HC-SR04) is used to measure distance by using sound waves. It emits ultrasonic waves and calculates the time taken for the echo to return after reflecting from an object.

Distance is calculated using the formula:

$$\text{Distance} = \frac{\text{Time} \times \text{Speed of Sound}}{2}$$

In Arduino, this is simplified as:

$$\text{Distance (cm)} = \frac{\text{Time}}{58.2}$$

Where:

- Time = duration of echo pulse
- Speed of sound ≈ 343 m/s

The LCD is used to display the measured distance visually.

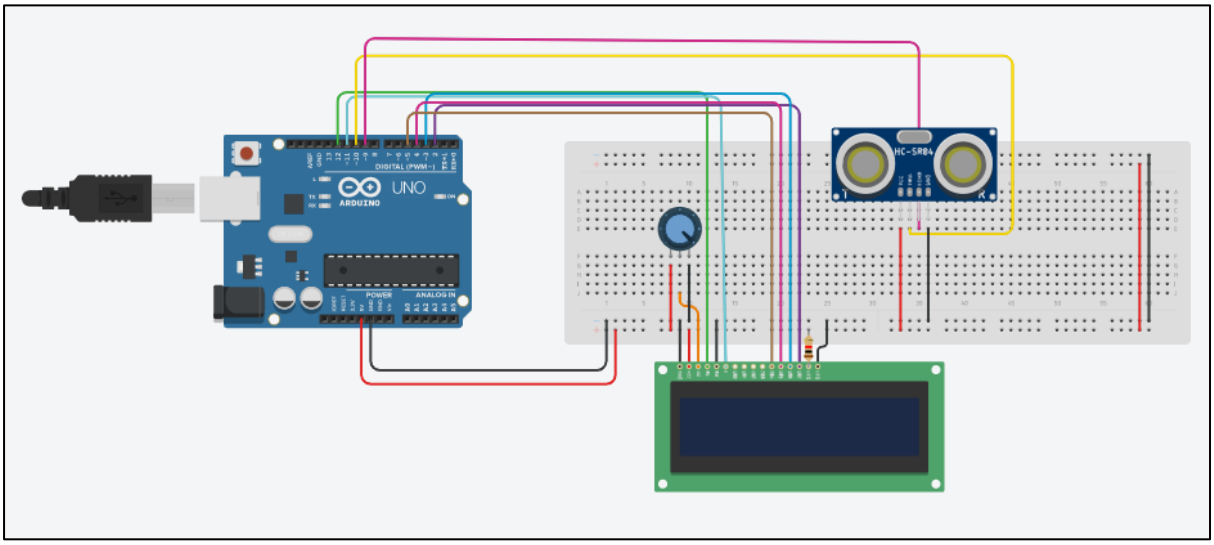
Circuit Connections:

Ultrasonic Sensor (HC-SR04):

- VCC → 5V
- GND → GND
- TRIG → Pin 10
- ECHO → Pin 9

LCD Connections:

- RS → Pin 12
- EN → Pin 11
- D4 → Pin 5
- D5 → Pin 4
- D6 → Pin 3
- D7 → Pin 2
- VSS → GND
- VDD → 5V



Program Code:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int trig = 10;
int echo = 9;
float time;
float height;
float fixed = 190;

void setup()
{
  pinMode(echo, INPUT);
  pinMode(trig ,OUTPUT);
  Serial.begin(9600);
  lcd.begin(16, 2);
```

```
  lcd.print("WELCOME");
  delay(1000);
}

void loop()
{
  lcd.setCursor(0,0);

  digitalWrite(trig , LOW);
  delayMicroseconds(2);
  digitalWrite(trig , HIGH);
  delayMicroseconds(10);
  digitalWrite(trig , LOW);

  time = pulseIn(echo , HIGH);
  height = time / 58.2;
  height = fixed - height;

  lcd.print(height);
  lcd.setCursor(1,0);
  lcd.print(height);
  lcd.print("cm");

  Serial.println(height);

  delay(10);
}
```

Procedure:

1. Connect the ultrasonic sensor and LCD to Arduino as per the circuit diagram.
2. Open Arduino IDE and write/paste the program.
3. Upload the code to Arduino Uno.
4. Open the Serial Monitor.
5. Place an object in front of the ultrasonic sensor.
6. Observe the distance displayed on the LCD and Serial Monitor.

Observations:

Sr. No.	Distance (cm)
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Result:

The distance of the object was successfully measured using the ultrasonic sensor and displayed on the Serial Monitor and LCD.

Precautions:

- Ensure proper connections of the sensor and LCD.
- Avoid obstacles that may distort the ultrasonic signal.
- Keep the sensor stable for accurate readings.
- Do not place objects too close (<2 cm) to the sensor.

Applications:

- Distance measurement systems
- Water level monitoring
- Obstacle detection in robotics
- Parking sensors

Title: DC Motor Speed Control using PWM

1. Aim

To control the speed of a DC motor using Pulse Width Modulation (PWM) with a potentiometer and Arduino.

---

2. Apparatus Required

- Arduino Uno (or compatible board)
- DC motor
- Potentiometer (10kΩ)
- Transistor (e.g., TIP122 or NPN like BC547) / Motor driver
- Diode (1N4007 for back EMF protection)
- Resistor (1kΩ)
- Breadboard
- Connecting wires
- External power supply (if required)

---

3. Theory

PWM (Pulse Width Modulation) is a technique used to control the amount of power delivered to a device by varying the duty cycle of a digital signal.

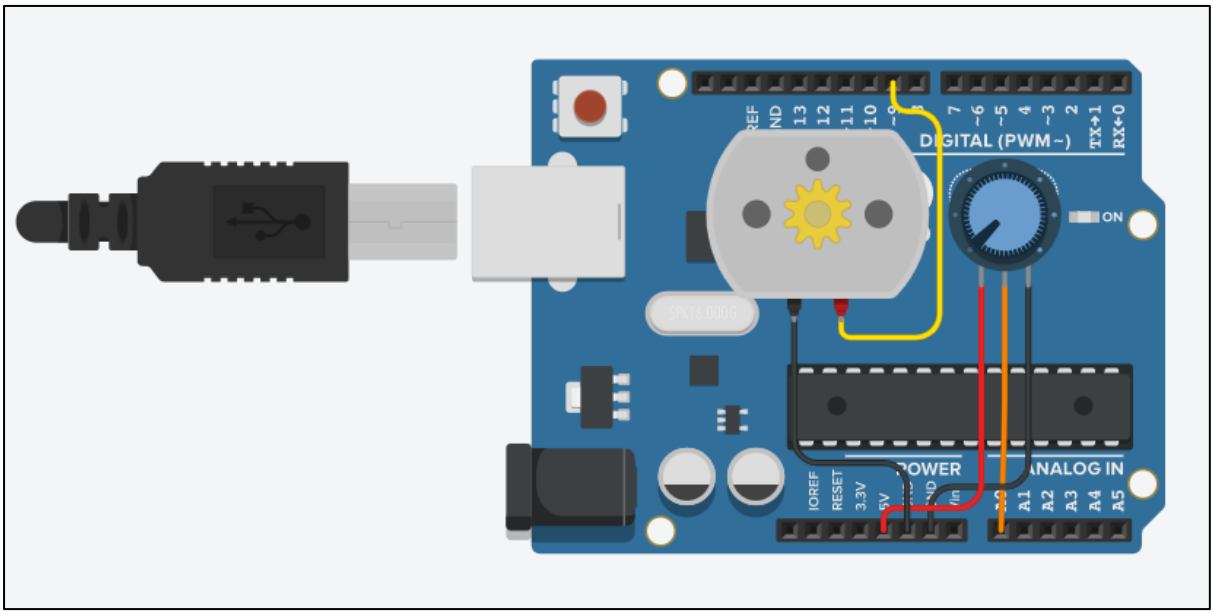
- Arduino provides PWM output using `analogWrite()` function.
- PWM values range from **0 to 255**:
  - 0 → 0% duty cycle (motor OFF)
  - 255 → 100% duty cycle (full speed)

A potentiometer acts as a variable resistor and provides analog input (0–1023), which is mapped to PWM output (0–255) to control motor speed.

---

4. Circuit Diagram (Description)

- Connect potentiometer:
  - One end → 5V
  - Other end → GND
  - Middle pin → A0
- Motor connection:
  - Motor positive → External supply (+)
  - Motor negative → Collector of transistor
  - Emitter → GND
  - Base → Arduino pin 9 via resistor
- Place a diode across the motor terminals (reverse biased) for protection.



5. Program

```
int potPin = A0;  // connect potentiometer to analog input pin A0
int motorPin = 9; // connect motor to digital output pin 9
int speed;        // variable to store motor speed

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

void loop() {
  speed = analogRead(potPin);  // read potentiometer value
  speed = map(speed, 0, 1023, 255, 0); // map potentiometer value to motor
speed range (0-255)
  analogWrite(motorPin, speed); // set motor speed using PWM
  delay(10);
}
```

6. Procedure

1. Assemble the circuit as described.
2. Connect Arduino to the computer using USB cable.
3. Open Arduino IDE and write/paste the program.
4. Select correct board and port.
5. Upload the program to Arduino.
6. Rotate the potentiometer knob slowly.
7. Observe the change in motor speed.

7. Observations

- When potentiometer is rotated:

    - Motor speed changes gradually.
    - Speed decreases when knob is turned in one direction (due to reverse mapping).
- Smooth variation of speed is observed using PWM.

---

8. Result

The speed of the DC motor is successfully controlled using PWM through Arduino and a potentiometer.

---

9. Precautions

- Do not connect motor directly to Arduino pin (use transistor/driver).
- Ensure proper polarity of diode across motor.
- Use external power supply for higher power motors.
- Avoid loose connections.

---

10. Applications

- Fan speed control
- Robotics (wheel speed control)
- Conveyor systems
- Electric vehicles (basic concept)

---

11. Viva Questions

1. What is PWM?
2. What is the range of PWM in Arduino?
3. Why is a transistor used in the circuit?
4. What is the function of the diode across the motor?
5. What does the `map()` function do?

Title: Interfacing Servo Motor with Arduino

1. Aim

To interface a servo motor with Arduino and control its position at different angles.

---

2. Apparatus Required

- Arduino Uno (or compatible board)
- Servo motor (e.g., SG90)
- Connecting wires
- Breadboard (optional)
- Power supply (if required)

---

3. Theory

A **servo motor** is a rotary actuator that allows precise control of angular position.

- It operates typically between **0° to 180°**.
- Controlled using PWM signals from Arduino.
- The Arduino **Servo library** simplifies control using `servo.write(angle)`.

PWM signal determines shaft position:

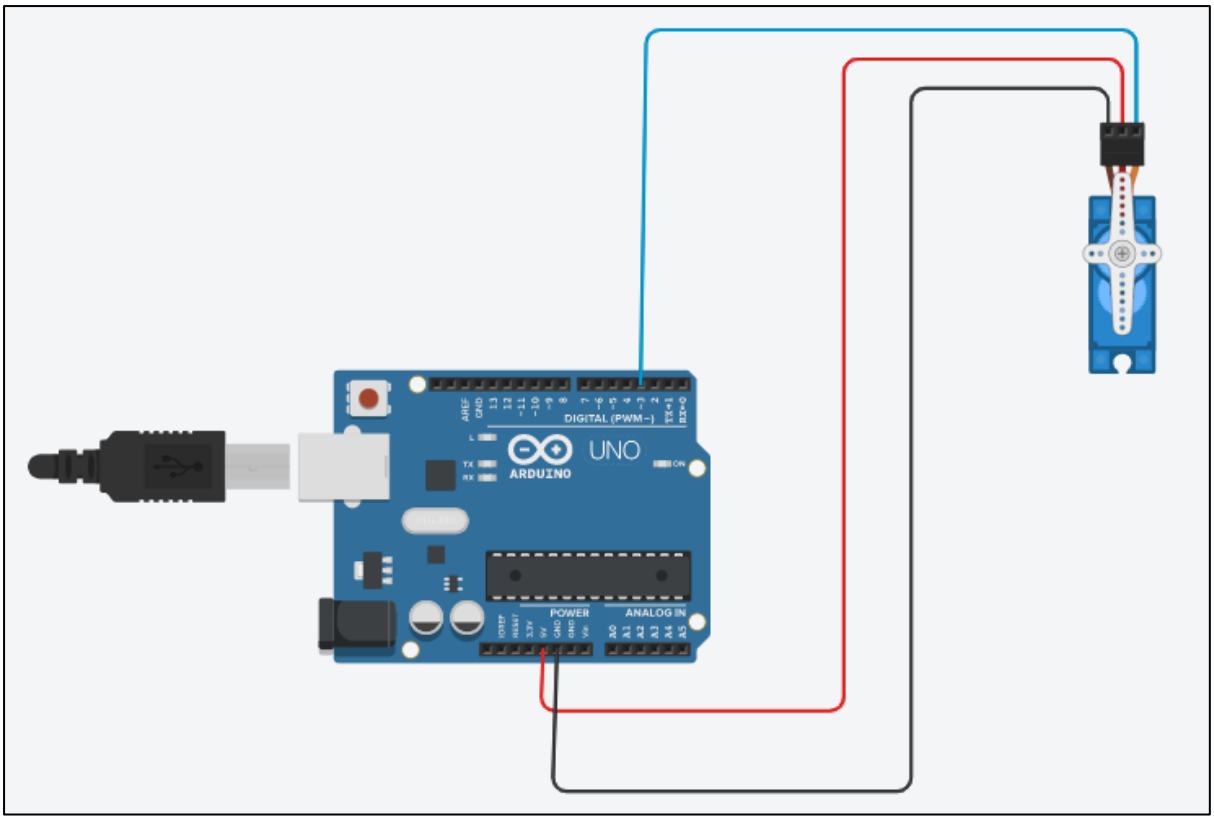
- ~0° → 1 ms pulse
- ~90° → 1.5 ms pulse
- ~180° → 2 ms pulse

---

4. Circuit Diagram (Description)

Servo motor has 3 wires:

- **Red** → 5V (Power)
- **Brown/Black** → GND
- **Yellow/Orange** → Signal pin (connected to Arduino pin 3)



5. Program

```
#include <Servo.h>
int servoPin = 3;
Servo servo;

void setup()
{
  servo.attach(servoPin);
}

void loop()
{
  servo.write(0);
  delay(1000);

  servo.write(90);
  delay(1000);

  servo.write(180);
  delay(1000);

  for (int i=0;i<180;i++)
  {
    servo.write(i);
    delay(50);
  }
}
```

---

6. Procedure

1. Connect the servo motor to Arduino as described.

2. Open Arduino IDE and enter the given program.
3. Select the correct board and port.
4. Upload the program to Arduino.
5. Observe the movement of the servo motor.

---

7. Observations

- Servo moves to:
  - **0° position** and holds for 1 second
  - **90° position** and holds for 1 second
  - **180° position** and holds for 1 second
- Then it rotates **gradually from 0° to 180°** in small steps.

---

8. Result

The servo motor is successfully interfaced with Arduino and its position is controlled at different angles using PWM signals.

---

9. Precautions

- Do not exceed the servo's voltage rating (usually 5V).
- Ensure correct wiring of power, ground, and signal.
- Avoid forcing the servo shaft manually.
- Use external power supply for multiple servos.

---

10. Applications

- Robotic arms
- Automatic door systems
- Camera positioning systems
- CNC and automation systems

---

11. Viva Questions

1. What is a servo motor?
2. What is the range of motion of a typical servo?
3. Which library is used to control servo in Arduino?
4. What does `servo.attach()` do?
5. How does PWM control servo position?