

## Practical - 1

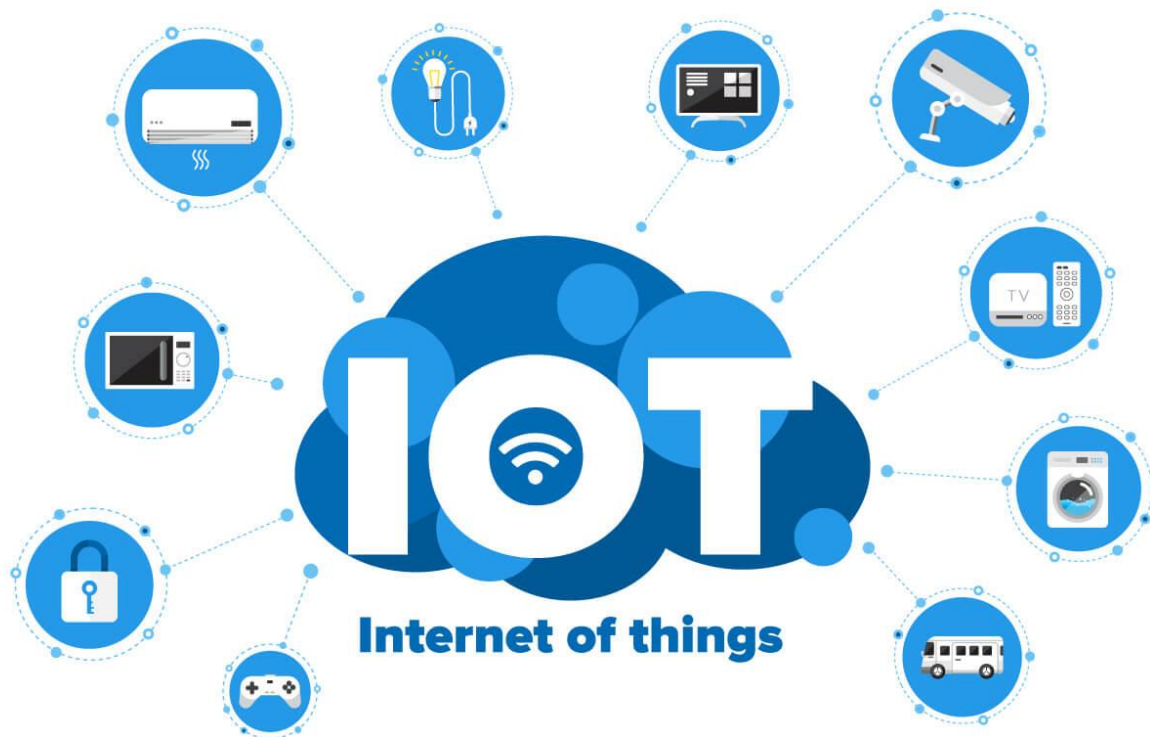
**Aim:** To study IoT and Arduino UNO in detail.

### Introduction

IoT stands for Internet of Things. It refers to the interconnectedness of physical devices, such as appliances and vehicles, that are embedded with software, sensors, and connectivity which enables these objects to connect and exchange data. This technology allows for the collection and sharing of data from a vast network of devices, creating opportunities for more efficient and automated systems.

**Internet of Things (IoT)** is the networking of physical objects that contain electronics embedded within their architecture in order to communicate and sense interactions amongst each other or with respect to the external environment. In the upcoming years, IoT-based technology will offer advanced levels of services and practically change the way people lead their daily lives. Advancements in medicine, power, gene therapies, agriculture, smart cities, and smart homes are just a few of the categorical examples where IoT is strongly established.

IOT is a system of interrelated things, computing devices, mechanical and digital machines, objects, animals, or people that are provided with unique identifiers. And the ability to transfer the data over a network requiring human-to-human or human-to-computer interaction.



## **History of IoT**

- 1982- Vending machine
- 1990- Toaster
- 1999- IOT (Kevin Ashton)
- 2000- LG Smart Fridge
- 2004- Smart Watch
- 2007- Smart Phone
- 2009- Car Testing
- 2011- Smart TV
- 2013- Google Lens
- 2014- Echo
- 2015- Tesla autopilot

## **Key Components of IoT**

- Device or sensor
- Connectivity
- Data processing
- Interface

**IoT is network of interconnected computing devices which are embedded in everyday objects, enabling them to send and receive data.**

Over 9 billion things (physical objects) are currently connected to the Internet, as of now. In the near future, this number is expected to rise to a whopping 20 billion.

## **Main Components used in IoT**

- **Low-power embedded systems:** Less battery consumption, high performance are the inverse factors that play a significant role during the design of electronic systems.
- **Sensors:** Sensors are the major part of any IoT application. It is a physical device that measures and detects certain physical quantities and converts it into signal which can be provided as an input to processing or control unit for analysis purpose.

## **Characteristics of IoT**

The Internet of Things (IoT) is a network of interconnected devices that communicate and exchange data with each other over the internet. Following are some of the key characteristics of IoT:

1. **Connectivity:** IoT devices are connected to the internet or to each other, enabling them to transmit and receive data. This connectivity can be wired or wireless, such as Wi-Fi, Bluetooth, Zigbee, or cellular networks.
2. **Sensing and Actuating:** IoT devices are equipped with sensors to collect data from their surroundings and actuators to perform actions based on the collected data. These sensors can measure various parameters like temperature, humidity, light, motion, etc.
3. **Data Processing:** IoT devices often have some degree of processing capability to analyse the data they collect locally or to preprocess it before sending it to a central server or cloud platform. This helps in reducing latency and bandwidth usage.
4. **Interoperability:** IoT devices and systems should be designed to work together seamlessly, regardless of their manufacturer or underlying technology. Standardized communication protocols and data formats facilitate interoperability.
5. **Scalability:** IoT networks can scale from a few devices to millions of devices without significant infrastructure changes. This scalability is crucial for accommodating the growing number of connected devices in various applications.
6. **Security:** Security is a critical aspect of IoT, given the sensitive nature of the data collected and transmitted by these devices. IoT systems need to implement robust security measures to protect against unauthorized access, data breaches, and cyber-attacks.
7. **Remote Monitoring and Control:** IoT enables remote monitoring and control of devices and systems from anywhere with an internet connection. This capability allows for real-time tracking, management, and automation of processes and equipment.
8. **Data Analytics and Insights:** IoT generates vast amounts of data, which can be analysed to derive valuable insights for decision-making, optimization, and prediction. Advanced analytics techniques, including machine learning and artificial intelligence, are often used to extract actionable intelligence from IoT data.
9. **Energy Efficiency:** Many IoT devices are designed to be energy-efficient to prolong battery life or reduce power consumption when connected to mains power. Energy-efficient designs are essential for IoT devices deployed in remote or battery-powered environments.
10. **Ubiquitous Presence:** IoT technology can be found in various industries and applications, including smart homes, healthcare, transportation, agriculture, industrial automation, and environmental monitoring. Its ubiquitous presence contributes to the concept of a connected world where everyday objects are part of a vast network of interconnected devices.

### **Communication in IoT**

Communication in the Internet of Things (IoT) is essential for enabling devices to exchange data and interact with each other. Several communication protocols and technologies are used to facilitate this data exchange. Here are some key aspects of communication in IoT:

1. **Wireless Communication:** Many IoT devices communicate wirelessly due to their often remote or distributed nature. Wireless technologies such as Wi-Fi, Bluetooth, Zigbee, Z-Wave, LoRaWAN, and cellular (e.g., 3G, 4G, and 5G) are commonly employed for IoT communication. Each of these wireless technologies has its own advantages and is suitable for different IoT use cases based on factors like range, data rate, power consumption, and cost.
2. **Low-Power Protocols:** Many IoT devices are battery-powered or operate on low power, requiring communication protocols optimized for energy efficiency. Examples include

Bluetooth Low Energy (BLE), Zigbee, and LoRaWAN. These protocols allow devices to conserve power while maintaining communication capabilities, enabling long battery life or energy-efficient operation.

3. **Short-Range Communication:** Short-range communication protocols like Bluetooth and Zigbee are suitable for IoT applications within a limited range, such as smart home devices, wearable technology, and industrial sensor networks. These protocols typically offer low-power consumption and high data transfer rates over short distances.
4. **Long-Range Communication:** Long-range communication protocols like LoRaWAN and cellular networks (3G, 4G, and 5G) enable IoT devices to communicate over greater distances, making them suitable for applications like asset tracking, smart agriculture, and smart city infrastructure. These protocols provide extended coverage and support for devices deployed across large areas.
5. **Mesh Networking:** Mesh networking allows IoT devices to form decentralized networks where each device can communicate with other nearby devices, even if they are out of range of a central hub or gateway. Zigbee and Thread are examples of mesh networking protocols commonly used in IoT applications. Mesh networks offer robustness, scalability, and self-healing capabilities, making them suitable for large-scale deployments and environments with obstacles or interference.
6. **MQTT (Message Queuing Telemetry Transport):** MQTT is a lightweight publish-subscribe messaging protocol commonly used in IoT applications. It is designed for constrained devices and unreliable networks, making it well-suited for IoT scenarios where bandwidth and reliability are concerns. MQTT facilitates efficient communication between IoT devices and cloud platforms or edge gateways.
7. **CoAP (Constrained Application Protocol):** CoAP is another lightweight protocol designed for constrained devices and low-power networks, such as those found in IoT deployments. CoAP enables devices to exchange data using RESTful web services over UDP or SMS, making it suitable for resource-constrained environments like sensor networks and IoT-enabled smart objects.
8. **HTTP and RESTful APIs:** In addition to specialized IoT protocols, HTTP and RESTful APIs are also used for communication between IoT devices, gateways, and cloud platforms. RESTful APIs provide a standard way for devices to interact with cloud services, enabling integration with web-based applications and services.

Overall, communication in IoT is diverse, encompassing a range of wireless technologies, protocols, and standards tailored to different use cases, requirements, and constraints. Effective communication is essential for enabling IoT devices to collect, transmit, and act upon data, facilitating the creation of interconnected and intelligent IoT systems.

### **Applications of IoT**

The Internet of Things (IoT) has a wide range of applications across various industries and sectors. Here are some notable examples:

1. **Smart Home Automation:** IoT enables the automation and control of home appliances, lighting, heating, air conditioning, security cameras, door locks, and more. Smart home devices can be remotely monitored and controlled using smartphones or voice assistants, improving convenience, energy efficiency, and home security.
2. **Healthcare Monitoring:** IoT devices such as wearable fitness trackers, smart medical devices, and remote patient monitoring systems enable continuous monitoring of vital

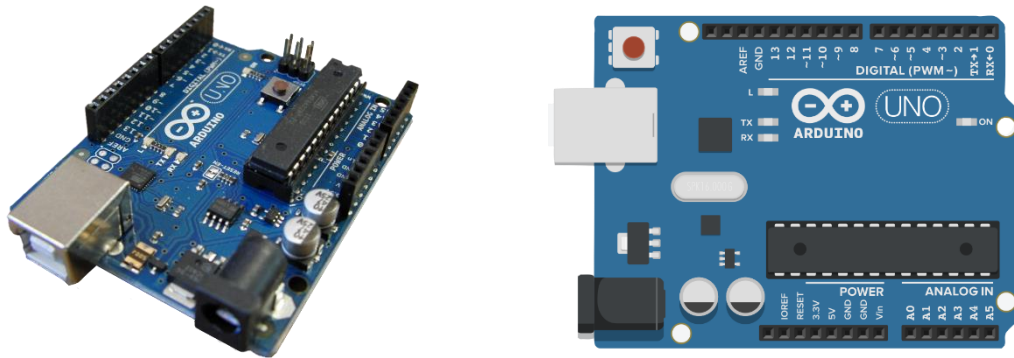
signs, medication adherence, and health parameters. These devices facilitate remote healthcare delivery, personalized treatment plans, and early detection of health issues.

3. **Industrial IoT (IoT):** IoT involves the use of IoT devices, sensors, and connectivity technologies to optimize industrial processes, monitor equipment health, and improve operational efficiency. Examples include predictive maintenance, asset tracking, supply chain optimization, and real-time monitoring of production lines.
4. **Smart Cities:** IoT technologies are used to create smart city solutions for urban planning, infrastructure management, and public services. Examples include smart traffic management, intelligent transportation systems, waste management, environmental monitoring, smart street lighting, and public safety and surveillance systems.
5. **Agriculture and Precision Farming:** IoT devices and sensors are deployed in agriculture to monitor soil moisture, temperature, humidity, and other environmental factors. This data is used to optimize irrigation, fertilization, and pest control, leading to increased crop yields, resource efficiency, and sustainability.
6. **Retail and Supply Chain Management:** IoT enables retailers to track inventory levels, monitor product shipments, and analyse customer behaviour in real time. RFID tags, beacons, and sensors are used for asset tracking, inventory management, and personalized marketing, improving supply chain efficiency and enhancing the customer shopping experience.
7. **Smart Energy Management:** IoT technologies are employed for smart grid systems, energy monitoring, and demand response programs. Smart meters, sensors, and connected devices enable real-time monitoring of energy consumption, optimization of energy distribution, and integration of renewable energy sources, leading to energy savings and reduced carbon emissions.
8. **Environmental Monitoring:** IoT devices are used for environmental monitoring and conservation efforts. Sensors and monitoring systems track air quality, water quality, pollution levels, and biodiversity in urban and natural environments, providing valuable data for environmental research, policy-making, and sustainability initiatives.
9. **Smart Transportation:** IoT applications in transportation include intelligent traffic management, vehicle tracking, fleet management, and smart parking solutions. Connected vehicles, traffic sensors, and predictive analytics optimize traffic flow, reduce congestion, and enhance safety and efficiency on roads and public transportation systems.
10. **Emergency Response and Disaster Management:** IoT technologies are utilized for disaster preparedness, early warning systems, and emergency response coordination. IoT sensors and communication networks provide real-time data on natural disasters, accidents, and public safety incidents, enabling faster response times and more effective disaster recovery efforts.

These are just a few examples of the diverse applications of IoT across different industries and domains. The continued advancement of IoT technology is expected to drive further innovation and transformation in various sectors, improving efficiency, productivity, and quality of life.

### **Introduction to Arduino**

Arduino is an open-source electronics platform that consists of both hardware and software components. It is designed for building digital devices and interactive objects that can sense and control physical devices in the real world. Arduino is popular among hobbyists, students, artists, and professionals due to its simplicity, versatility, and affordability.



The key components and concepts on Arduino are:

1. **Arduino Board:** The Arduino board serves as the core of the platform. It is a small, programmable microcontroller board that contains a microcontroller chip, input/output pins, voltage regulator, and other essential components. There are various types of Arduino boards available, each with different features and specifications.
2. **Microcontroller:** At the heart of the Arduino board is a microcontroller chip, which is responsible for executing the code uploaded to the board. The most commonly used microcontroller on Arduino boards is the Atmel AVR series, although other microcontroller architectures like ARM are also used in some Arduino-compatible boards.
3. **Integrated Development Environment (IDE):** The Arduino IDE is a software application used to write, compile, and upload code to the Arduino board. It provides a simple and beginner-friendly interface for writing programs (sketches) in the Arduino programming language, which is based on C and C++.
4. **Arduino Programming Language:** Arduino sketches are written in a simplified version of C/C++ programming language. The Arduino language includes built-in functions and libraries that simplify common tasks such as reading sensor data, controlling actuators, and communicating with other devices.
5. **Input/Output Pins:** Arduino boards feature a set of digital and analog input/output (I/O) pins that allow the board to interact with external sensors, actuators, and other electronic components. Digital pins can be used to read or write digital signals (on/off), while analog pins can measure analog signals (voltage levels).
6. **Shields:** Arduino shields are add-on boards that can be plugged into the Arduino board to extend its capabilities. Shields come in various types, including motor drivers, wireless communication modules (Wi-Fi, Bluetooth), sensor modules, display modules, and more.
7. **Power Supply:** Arduino boards can be powered through USB connection, battery, or external power supply. They typically have built-in voltage regulators that allow them to accept a wide range of input voltages.
8. **Community and Documentation:** Arduino has a large and active community of users, developers, and enthusiasts who contribute to its development and share knowledge, projects, and resources online. There are numerous tutorials, guides, forums, and project examples available to help beginners get started with Arduino.

Overall, Arduino provides a user-friendly platform for experimenting with electronics and building interactive projects, making it an excellent choice for beginners and experienced makers alike. It offers a versatile and accessible platform for bringing your ideas to life.

## **Arduino UNO**

The Arduino Uno is a popular microcontroller board that is widely used for prototyping and building electronics projects. It is part of the Arduino family of boards and is known for its simplicity, versatility, and ease of use. Here are some key features of the Arduino Uno:

1. **Microcontroller:** The Arduino Uno is based on the Atmega328P microcontroller from Atmel (now Microchip). It operates at 16 MHz and has 32 KB of flash memory (program memory), 2 KB of SRAM (temporary memory), and 1 KB of EEPROM (non-volatile memory).
2. **Input/Output Pins:** The Uno has a total of 14 digital input/output (I/O) pins, of which 6 can be used as PWM (Pulse Width Modulation) outputs. Additionally, it has 6 analog input pins. These pins can be used to interface with various sensors, actuators, LEDs, and other electronic components.
3. **USB Interface:** The Arduino Uno features a built-in USB interface (USB Type-B connector) that allows it to connect to a computer for programming and serial communication. It uses the USB-to-serial converter chip (ATmega16U2) to facilitate communication between the board and the computer.
4. **Power Supply:** The Uno can be powered via USB from a computer or an external power supply connected to the DC power jack. It has a built-in voltage regulator that allows it to accept a wide range of input voltages (7-20V).
5. **Reset Button:** The Uno has a reset button that can be used to restart the microcontroller and reinitialize the program running on the board.
6. **ICSP Header:** The Uno has an In-Circuit Serial Programming (ICSP) header for programming the microcontroller using an external programmer, such as the AVRISP mkII or USBasp.
7. **Operating Voltage:** The Uno operates at 5V, which means that its digital and analog I/O pins are also 5V tolerant. However, it also has a 3.3V output pin that can be used to power 3.3V devices.
8. **Compatibility:** The Arduino Uno is compatible with the Arduino Integrated Development Environment (IDE) and supports the Arduino programming language, which is based on C/C++. It can be programmed using a simple and beginner-friendly programming interface.

Overall, the Arduino Uno is a versatile and beginner-friendly microcontroller board that is suitable for a wide range of electronics projects, including robotics, home automation, wearable technology, and more. Its ease of use, large community support, and extensive documentation make it an excellent choice for both beginners and experienced makers.

## **Features of Arduino**

Arduino is a popular open-source electronics platform known for its simplicity and versatility. Below are some of the key features that make Arduino a favoured choice for hobbyists, students, artists, and professionals alike:

1. **Microcontroller-based:** Arduino boards are built around microcontrollers, which are small computer chips that can be programmed to perform various tasks. The most commonly used microcontroller on Arduino boards is the Atmel AVR series, although other microcontroller architectures like ARM are also used in some Arduino-compatible boards.

2. **Open Source:** Arduino hardware designs, software, and documentation are open source, meaning they are freely available for anyone to use, modify, and distribute. This fosters a large and active community of users and contributors who share knowledge, projects, and resources online.
3. **Easy to Use:** Arduino is designed to be beginner-friendly, with a simple and intuitive programming environment (Arduino IDE) and a straightforward hardware interface. The IDE provides a user-friendly interface for writing, compiling, and uploading code to Arduino boards, making it accessible even to those with little or no programming experience.
4. **Versatile:** Arduino boards can be used for a wide range of projects and applications, including robotics, home automation, wearable technology, art installations, scientific experiments, and more. They can interface with various sensors, actuators, displays, and communication modules, making them suitable for diverse projects.
5. **Low Cost:** Arduino boards are relatively inexpensive compared to other microcontroller platforms, making them accessible to hobbyists, students, and enthusiasts with limited budgets. Additionally, Arduino-compatible boards and components are widely available from multiple vendors, further reducing costs.
6. **Extensible:** Arduino boards can be easily extended and customized using add-on boards called shields. Shields plug into the Arduino board's headers and provide additional functionality, such as motor control, wireless communication, sensor interfaces, and more. There are hundreds of shields available, covering a wide range of applications.
7. **Community Support:** Arduino has a large and active community of users, developers, and enthusiasts who contribute to its development and share knowledge, projects, and resources online. There are numerous tutorials, guides, forums, and project examples available to help beginners get started with Arduino and to support more advanced users in their projects.
8. **Cross-platform Compatibility:** The Arduino IDE and software libraries are available for Windows, macOS, and Linux, making Arduino a cross-platform development platform. This allows users to develop and program Arduino projects on their preferred operating system.
9. **Real-world Applications:** Arduino is used in various real-world applications and industries, including education, prototyping, research, product development, and commercial products. Its versatility, ease of use, and affordability make it a popular choice for both prototyping and production purposes.

Overall, Arduino's simplicity, versatility, affordability, and strong community support make it an ideal platform for experimenting with electronics, learning programming, and building innovative projects.

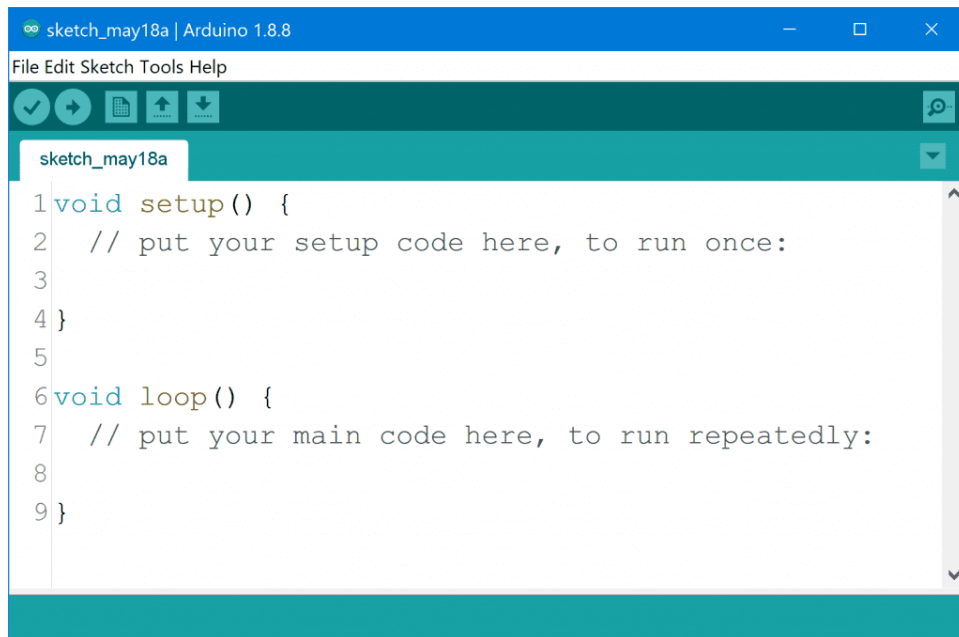


## Practical - 2

**Aim:** To study and install IDE of Arduino.

### Arduino IDE

The Arduino Integrated Development Environment (*IDE*) is a software application used to write, compile, and upload code to Arduino boards. It provides a user-friendly interface for programming Arduino microcontrollers and developing electronics projects.



The core components/ elements of the Arduino IDE are:

1. **Graphical Interface:** The Arduino IDE features a simple and intuitive graphical interface that is easy to navigate, making it suitable for beginners and experienced users alike. The main window of the IDE is divided into several sections, including the code editor, toolbar, serial monitor, and message console.
2. **Code Editor:** The code editor is where users write their Arduino sketches (programs). It supports syntax highlighting, auto-indentation, and auto-completion, which help streamline the coding process and improve code readability. The code editor also provides features such as find and replace, undo and redo, and code folding.
3. **Sketch Structure:** An Arduino sketch typically consists of two essential functions: `setup()` and `loop()`. The `setup()` function is called once when the Arduino board is powered on or reset and is used to initialize variables, configure pins, and perform other setup tasks. The `loop()` function is called repeatedly in a continuous loop and is where the main program logic is written.
4. **Library Manager:** The Arduino IDE includes a library manager that allows users to easily add, manage, and update libraries (collections of pre-written code) for various sensors, actuators, communication protocols, and other functionalities. The library manager simplifies the process of integrating third-party code into Arduino projects, saving time and effort.
5. **Board Manager:** The Arduino IDE's board manager enables users to select the target Arduino board and its corresponding microcontroller variant. It provides a convenient

interface for installing board support packages (BSPs) for different Arduino boards, including official Arduino boards and third-party compatible boards.

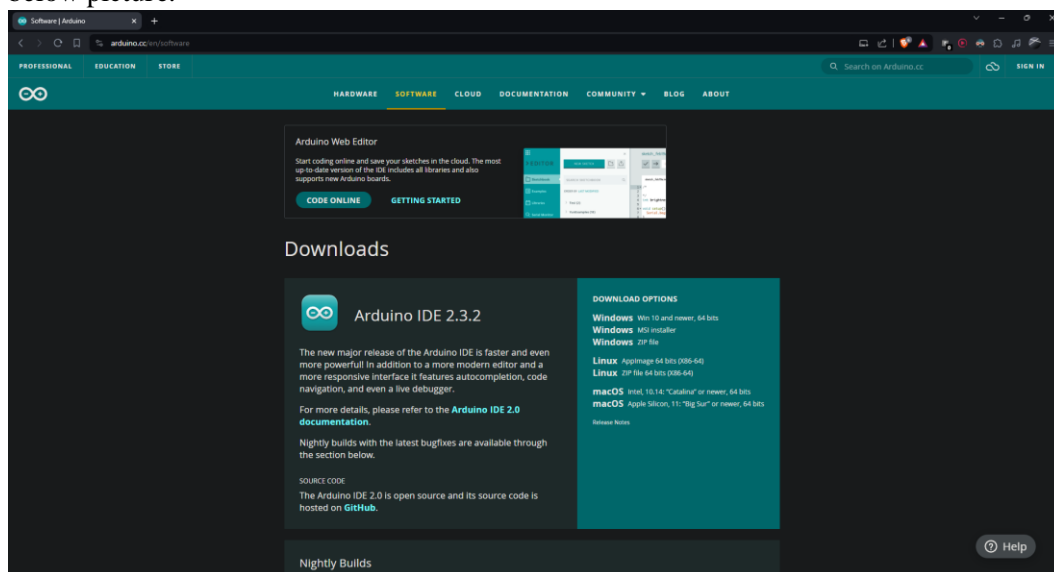
6. **Serial Monitor:** The serial monitor is a built-in tool in the Arduino IDE that allows users to communicate with the Arduino board via the serial port. It provides a text-based interface for sending and receiving data between the Arduino board and the computer, which is useful for debugging, troubleshooting, and displaying sensor readings or other output messages.
7. **Compilation and Upload:** Once the code is written, users can compile and upload it to the Arduino board directly from the IDE. The IDE invokes the Arduino compiler (avr-gcc) to compile the code into machine-readable binary format (HEX file) and then uses the bootloader or external programmer to upload the compiled code to the Arduino board via USB or serial connection.
8. **Tools and Preferences:** The Arduino IDE provides various tools and preferences that allow users to customize their development environment according to their preferences and requirements. This includes options for setting the serial port, baud rate, programmer type, and other configuration settings.
9. **Cross-Platform Compatibility:** The Arduino IDE is compatible with multiple operating systems, including Windows, macOS, and Linux, making it accessible to users regardless of their preferred platform.

Overall, the Arduino IDE is a powerful and user-friendly software tool that simplifies the process of programming Arduino boards and developing electronics projects. Its intuitive interface, built-in tools, and extensive documentation make it an ideal choice for beginners and experienced makers alike.

## **Installation of Arduino IDE**

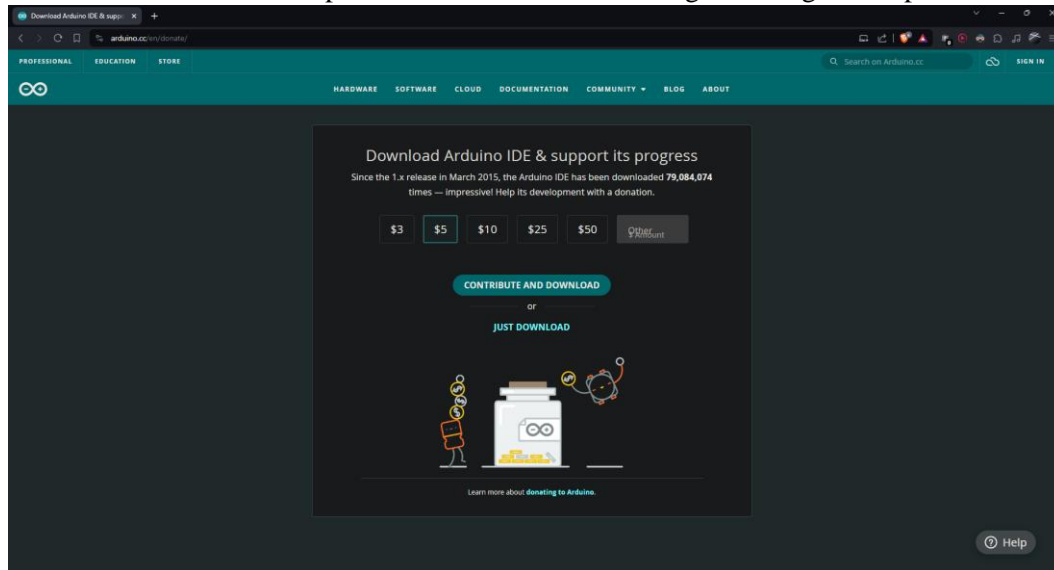
Following are the steps to install Arduino IDE on your PC:ar

1. Go to <https://www.arduino.cc/en/software>. The page that opens up will look something like the given below picture:

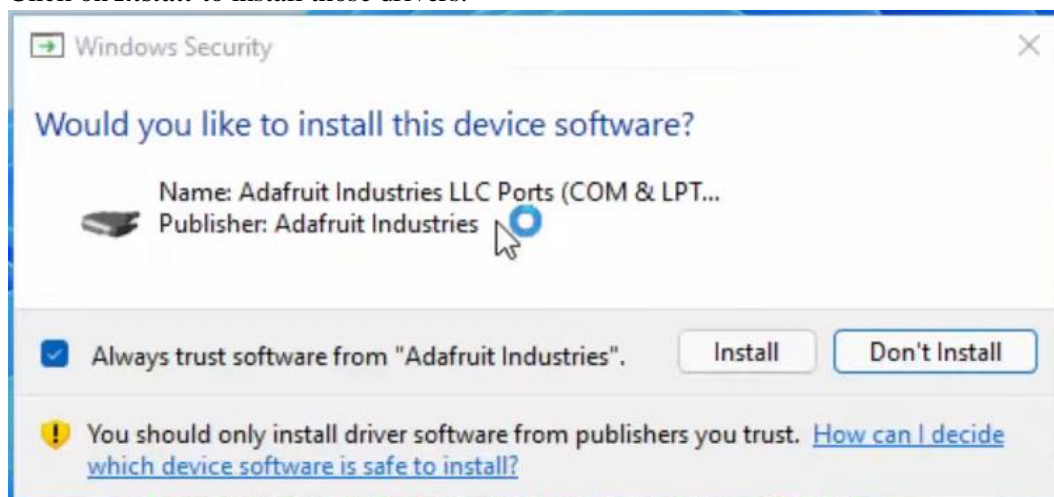


2. In the Download Options section, you will find various links according to your Operating System. Download the version according to your OS. Download .exe or .msi, if you want to setup the software, and download .zip file, if you want a portable version of the software. However, it is always

recommended to use a setup file. A screen like the below given image will open:



3. Click on **Just Download** button, to start downloading.
4. Once the downloading gets finished, run the setup, and provide any administrator privileges, if asked for.
5. The setup will show the EULA. Read all the terms and conditions properly and carefully. If you do not have any objections/ problems with the EULA policy, click on **I Agree** and accept it, to proceed further.
6. Next screen will show all the components that you can install, along with the main application. Leave it as it is, and click on **Next** to proceed.
7. Next screen will show the location where your software will get installed. By default, it will be "**C:\Program Files (x86)\Arduino**". You can change the destination folder, if you want. Click on **Install** to begin the installation process.
8. Before the successful completion of installation, the setup will prompt to download some additional drivers, to make the IDE work properly. The prompt will look something like the image given below. Click on **Install** to install those drivers.



9. Once the setup is done, click on **Close** to finish the installation.
10. You are done. You can now directly run the software.

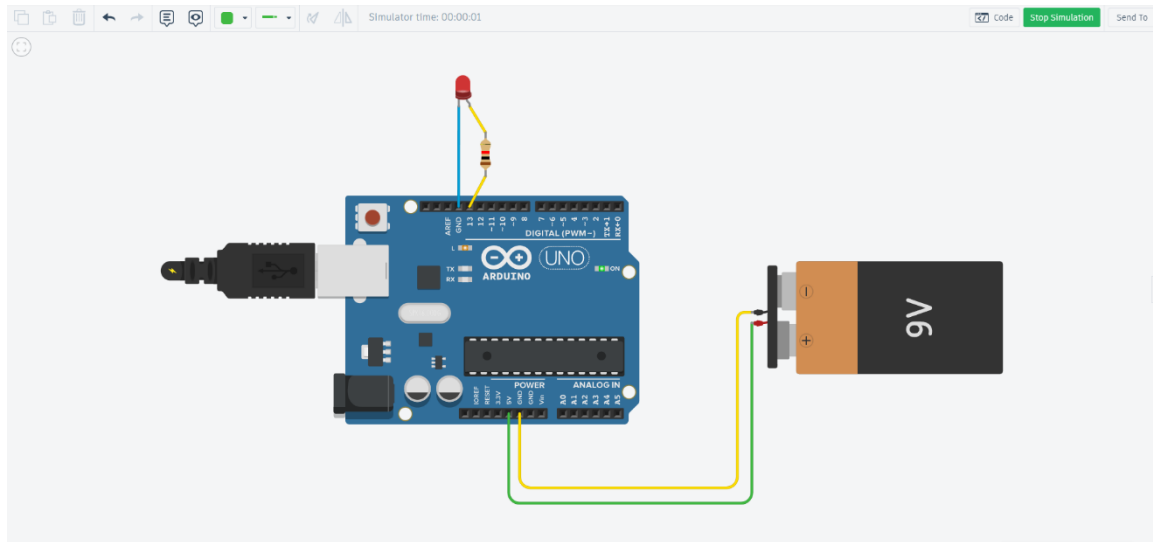
## Practical - 3

**Aim:** To blink an LED using Arduino.

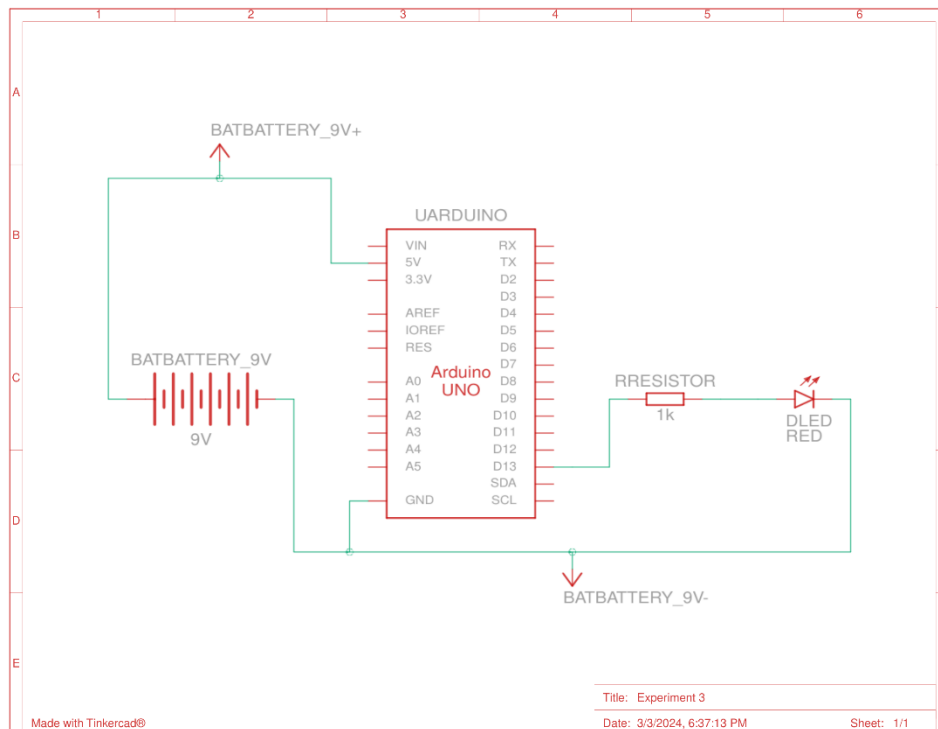
### **Introduction:**

This experiment aims at demonstrating basic Arduino functionality by creating a simple circuit to blink an LED, and to create a functional circuit that demonstrates digital output control and serves as a foundational introduction to Arduino programming and hardware interaction for beginners.

### **Circuit Diagram:**



### **Schematic View:**



## Key Components:

Component List			Download CSV
Name	Quantity	Component	
UArduino	1	Arduino Uno R3	
BATBattery 9V	1	9V Battery	
DLED	1	Red LED	
RResistor	1	1 kΩ Resistor	

## Code:

```
1 void setup() {
2   pinMode(LED_BUILTIN, OUTPUT);
3 }
4
5 void loop() {
6   digitalWrite(LED_BUILTIN, HIGH);
7   delay(1000); // Wait for 1000 millisecond(s)
8   digitalWrite(LED_BUILTIN, LOW);
9   delay(1000); // Wait for 1000 millisecond(s)
10 }
11
12 |
```

## Steps of Working:

1. Connect the negative terminal (*Cathode*) of the LED with the ground (*GND*) on Arduino.
2. Connect the positive terminal (*Anode*) of the LED with D13 point on Arduino.
3. Connect the negative terminal of the 9V battery with the ground (*GND*) on Arduino.
4. Connect the positive terminal of the 9V battery with 5V point of power on Arduino.

## Precautions:

- To avoid excess current from entering into the LED, which may damage the LED if it is left running for a long time, we need to connect the Anode of the LED to one of the terminals of a resistor (220Ω), and then connect the other terminal of the resistor to the D13 point on Arduino.

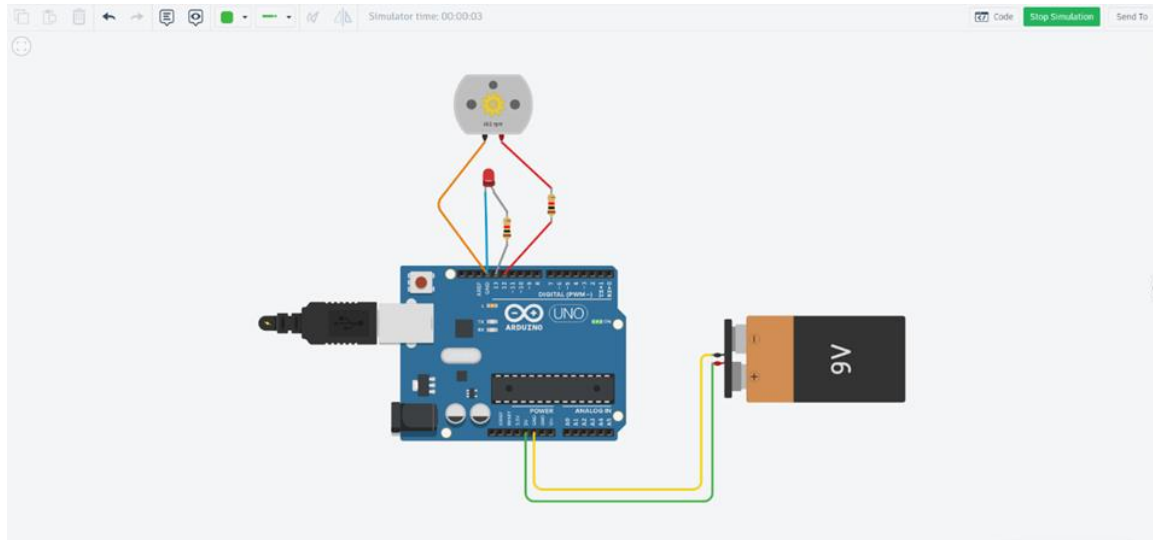
## Practical - 4

**Aim:** To demonstrate the working of a DC motor to an Arduino circuit.

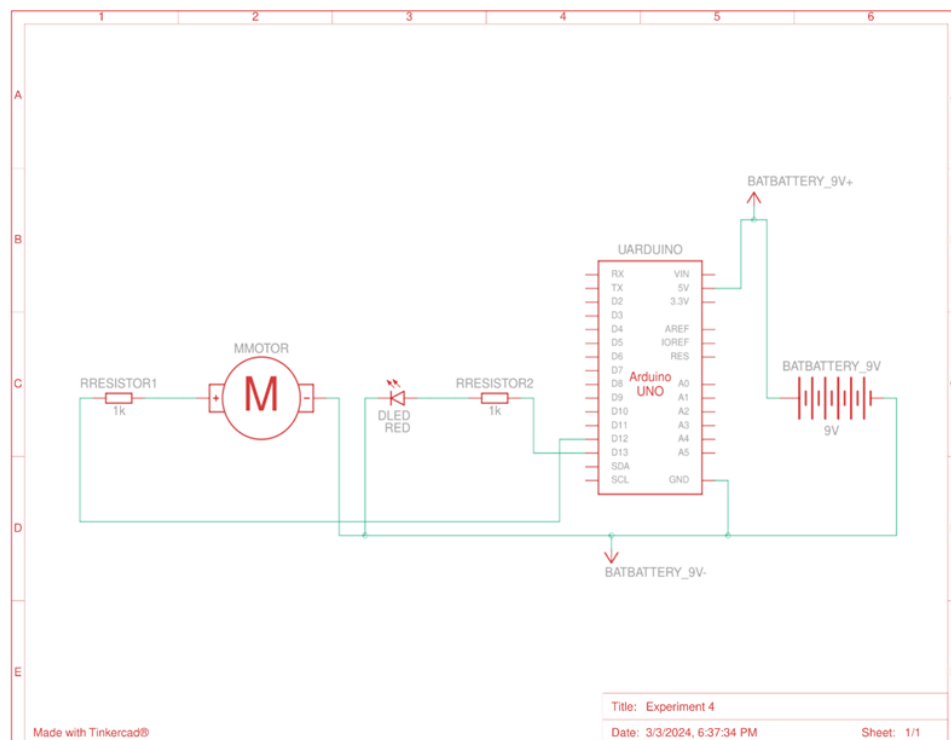
### **Introduction:**

This experiment aims at demonstrating the functioning of a DC motor, when it is connected to an Arduino, at the non-default point (***D12 here, considering D13 as the default point***). DC motor is integrated into an Arduino circuit, establishing a functional connection between the motor and the microcontroller at the point.

### **Circuit Diagram:**



### **Schematic View:**



## Key Components:

Component List			Download CSV
Name	Quantity	Component	
UArduino	1	Arduino Uno R3	
MMotor	1	DC Motor	
RResistor1 RResistor2	2	1 kΩ Resistor	
DLED	1	Red LED	
BATBattery 9V	1	9V Battery	

## Code:

```
1 void setup()
2 {
3     pinMode(LED_BUILTIN, OUTPUT);
4     pinMode(12, OUTPUT);
5 }
6
7 void loop()
8 {
9     digitalWrite(LED_BUILTIN, HIGH);
10    digitalWrite(12, OUTPUT);
11 }
12
13
```

## Steps of Working:

1. Connect the negative terminal (*Cathode*) of the DC motor with the ground (*GND*) on Arduino.
2. Connect the positive terminal (*Anode*) of the DC motor with D12 point on Arduino.
3. Connect the negative terminal of the 9V battery with the ground (*GND*) on Arduino.
4. Connect the positive terminal of the 9V battery with 5V point of power on Arduino.
5. To make this setup functional, we need to set the *pinMode()* function in *setup()* and *digitalWrite()* in *loop()*.

## Precautions:

- To avoid excess current from reaching the DC motor, which may damage the motor if it is kept running for a long time, we need to connect the Anode of the motor to one of the terminals of a resistor, and then connect the other terminal of the motor to the D13 point on Arduino. Before connecting the motor to a resistor, motor would have extremely high RPM. But after connecting the motor to a resistor, a cap is put on the RPM of motor.

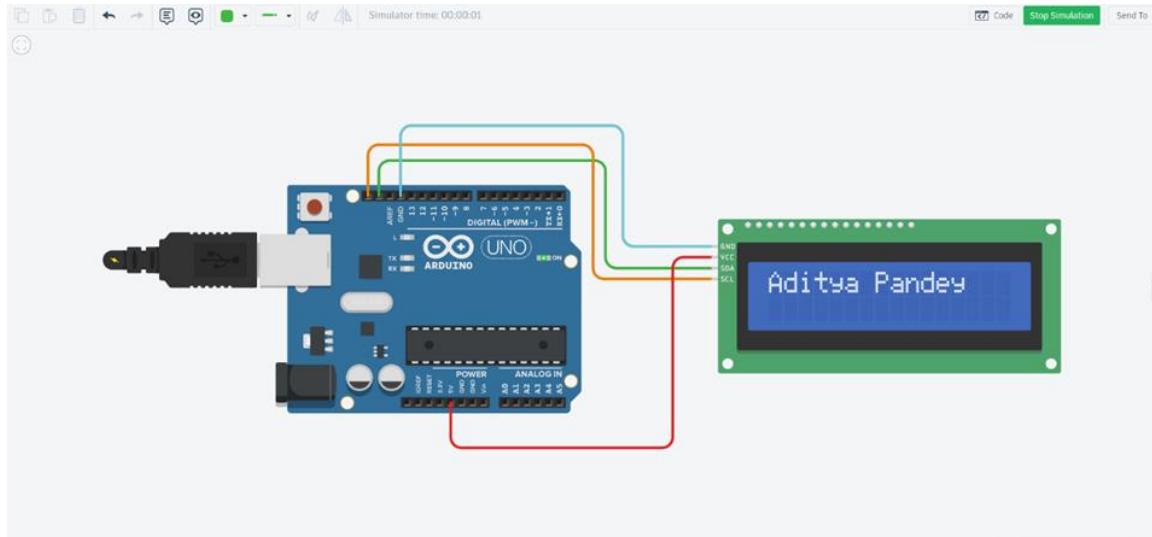
## Practical - 5

**Aim:** To display your name on an LCD screen using Arduino.

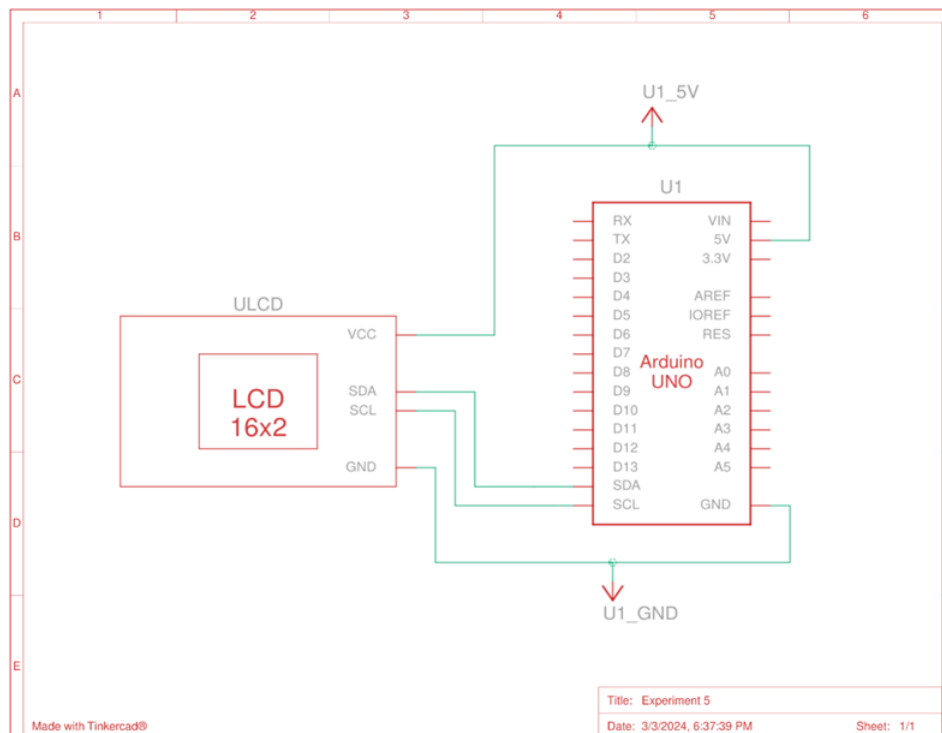
### **Introduction:**

This project aims at demonstrating the working of a Liquid Crystal Display (*LCD*) screen and displaying the student's name on it using Arduino. The students learn to connect and configure an LCD screen, interpret and execute code for customized text message, and comprehend the basics of character-based interfaces.

### **Circuit Diagram:**



### **Schematic View:**





## Key Components:

Component List			Download CSV
Name	Quantity	Component	
U1	1	Arduino Uno R3	
ULCD	1	MCP23008-based, 32 LCD 16 x 2 (I2C)	

## Code:

```
1 #include <Adafruit_LiquidCrystal.h>
2
3 Adafruit_LiquidCrystal lcd(0);
4
5 void setup() {
6     lcd.begin(16, 2);
7     lcd.print("Aditya Pandey");
8 }
9
10 void loop() {
11     lcd.setCursor(0, 1);
12     lcd.setBacklight(1);
13 }
14
15
```

## Steps of Working:

1. Connect GND of the LCD screen to the GND on Arduino.
2. Connect VCC of the LCD screen to the 5V point of power on Arduino.
3. Connect SDA of the LCD screen to the 2<sup>nd</sup> unnamed terminal on Arduino.
4. Connect SCL of the LCD screen to the 1<sup>st</sup> unnamed terminal on Arduino.
5. To make this setup functional, we use *Adafruit\_LiquidCrystal.h* library in the code.

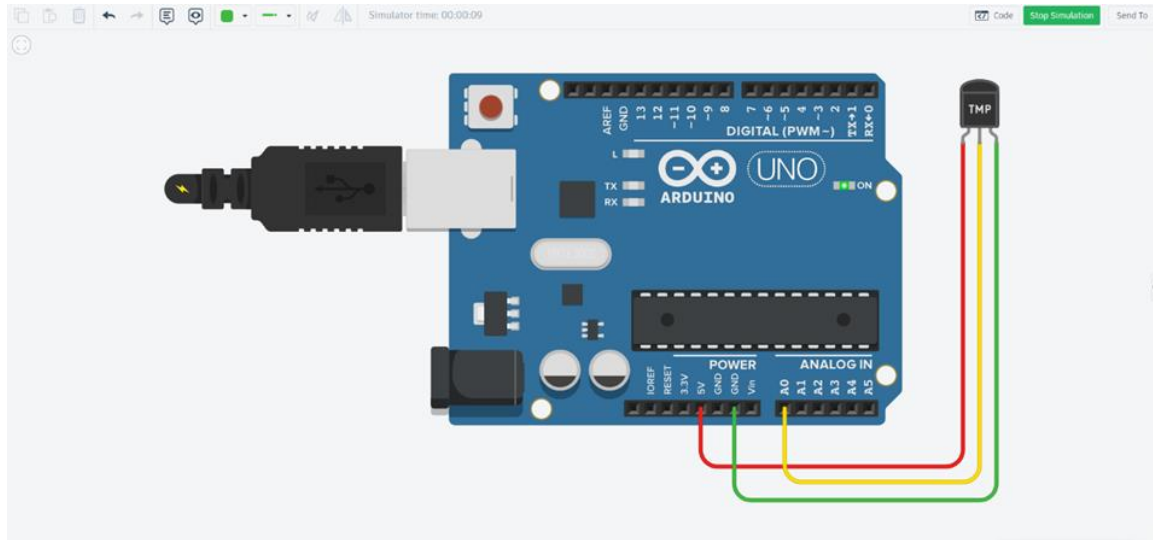
## Practical - 6

**Aim:** To show the working of temperature sensor using Arduino.

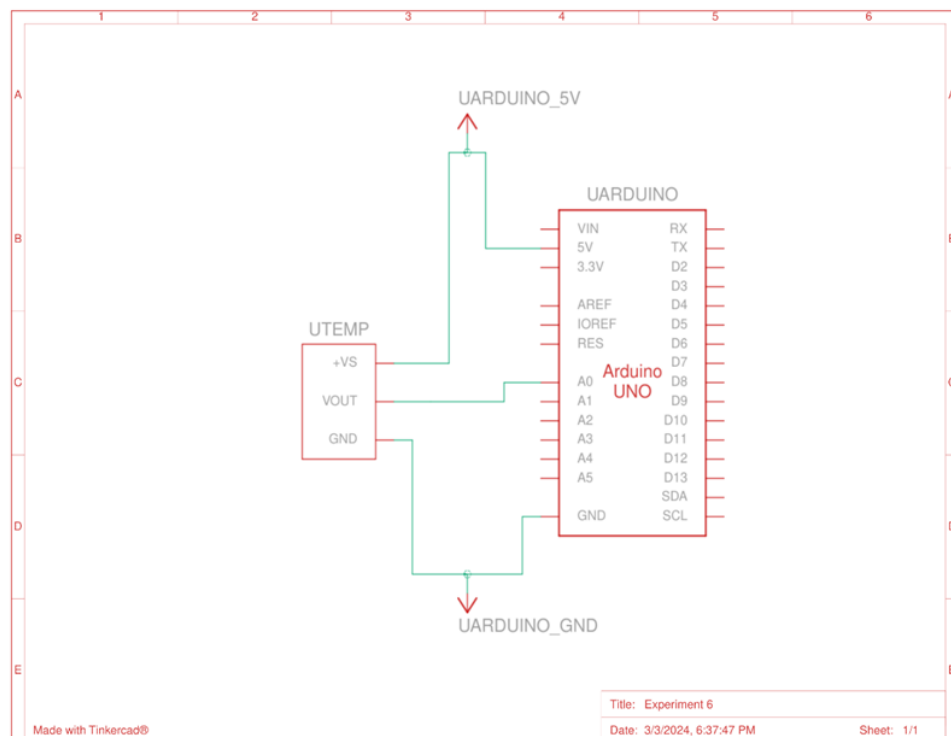
### **Introduction:**

This project aims at creating a basic understanding of how the temperature sensor works at different temperatures, when connected to Arduino. This experiment involves connecting the temperature sensor to the Arduino board, interpreting the analog signals from the sensor, and presenting temperature readings through appropriate coding.

### **Circuit Diagram:**



### **Schematic View:**

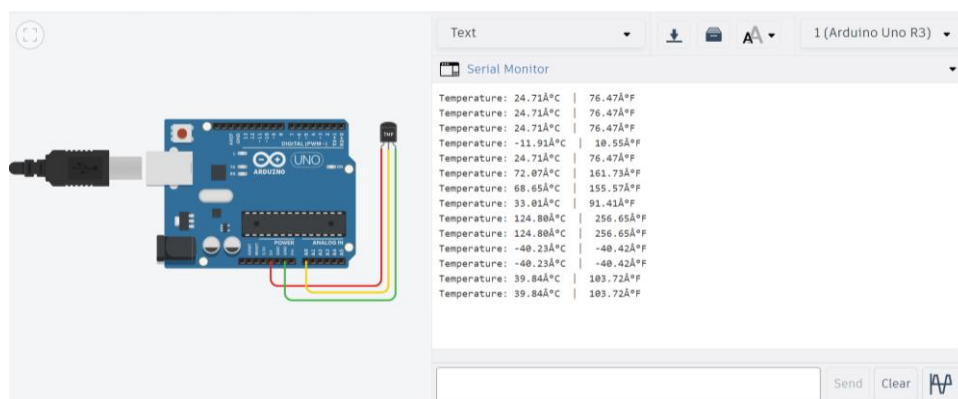


## Key Components:

Component List			Download CSV
Name	Quantity	Component	
UArduino	1	Arduino Uno R3	
UTemp	1	Temperature Sensor [TMP36]	

## Code:

```
1 #define sensorPin A0
2
3 void setup() {
4   Serial.begin(9600);
5 }
6
7 void loop() {
8   int reading = analogRead(sensorPin);
9   float voltage = reading * (5.0 / 1024.0);
10  float temperatureC = (voltage - 0.5) * 100;
11
12  Serial.print("Temperature: ");
13  Serial.print(temperatureC);
14  Serial.print("\xC2\xB0"); // Shows degree symbol
15  Serial.print("C | ");
16
17  float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
18  Serial.print(temperatureF);
19  Serial.print("\xC2\xB0"); // Shows degree symbol
20  Serial.println("F");
21
22  delay(1000); // Wait a second between readings
23 }
24
25
```



## Steps of Working:

1. Connect *Power* node of temperature sensor to 5V of *Power* point on Arduino.
2. Connect *GND* node of temperature sensor to GND on Arduino.
3. Connect the *Vout* node of temperature sensor to analog point (*here, A0*) on Arduino.
4. Define this sensor pin A0 in your code.

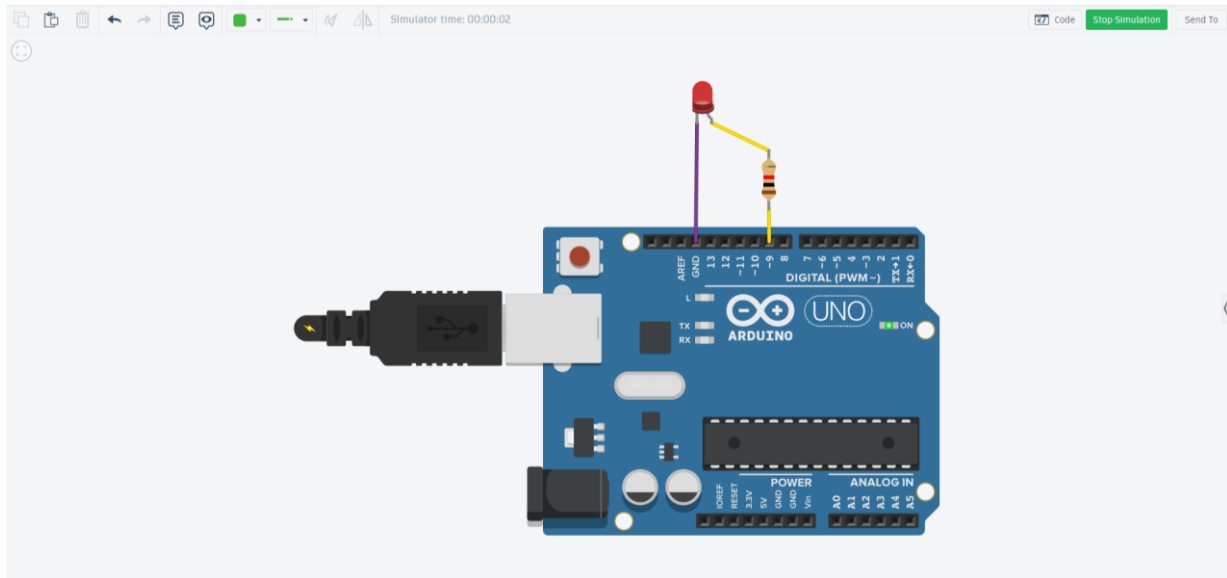
## Practical - 7

**Aim:** To show how to fade an LED using analogWrite() function.

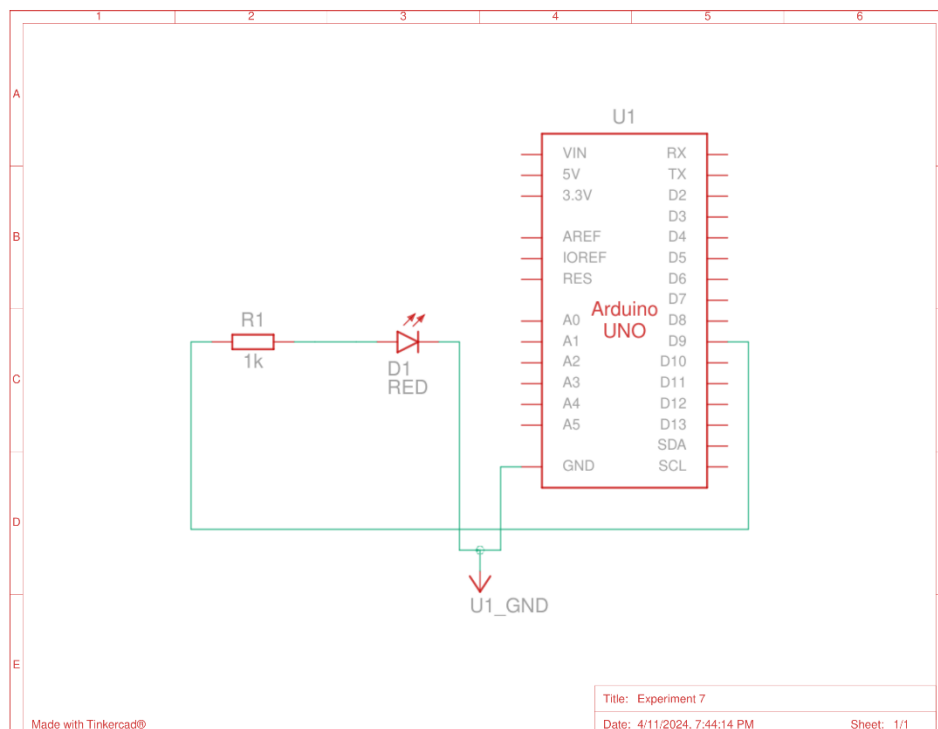
### **Introduction:**

This experiment aims at demonstrating basic Arduino functionality by creating a simple circuit to fade an LED but instead of regular digital command, it used analog commands. The purpose of this program is to demonstrate the usage of analogWrite() function.

### **Circuit Diagram:**



### **Schematic View:**



## Key Components:

Component List			Download CSV
Name	Quantity	Component	
U1	1	Arduino Uno R3	
D1	1	Red LED	
R1	1	1 kΩ Resistor	

## Code:

```
1 const int pin = 9;
2
3 void setup() {
4   pinMode(pin, OUTPUT);
5 }
6
7 void loop() {
8   for (int i=0 ; i<=255 ; i++) {
9     analogWrite(pin, i);
10    delay(1);
11  } delay(1000);
12
13  for (int i=255 ; i>=0 ; i--) {
14    analogWrite(pin, i);
15    delay(1);
16  } delay(1000);
17 }
18
19
```

## Steps of Working:

5. Connect the negative terminal (*Cathode*) of the LED with the ground (*GND*) on Arduino.
6. Connect the positive terminal (*Anode*) of the LED with D9 point on Arduino.
7. Define this D9 pin in your code.
8. Using `analogWrite()` function, make the LED glow.

## Precautions:

- To avoid excess current from entering into the LED, which may damage the LED if it is left running for a long time, we need to connect the Anode of the LED to one of the terminals of a resistor (220Ω), and then connect the other terminal of the resistor to the D13 point on Arduino.

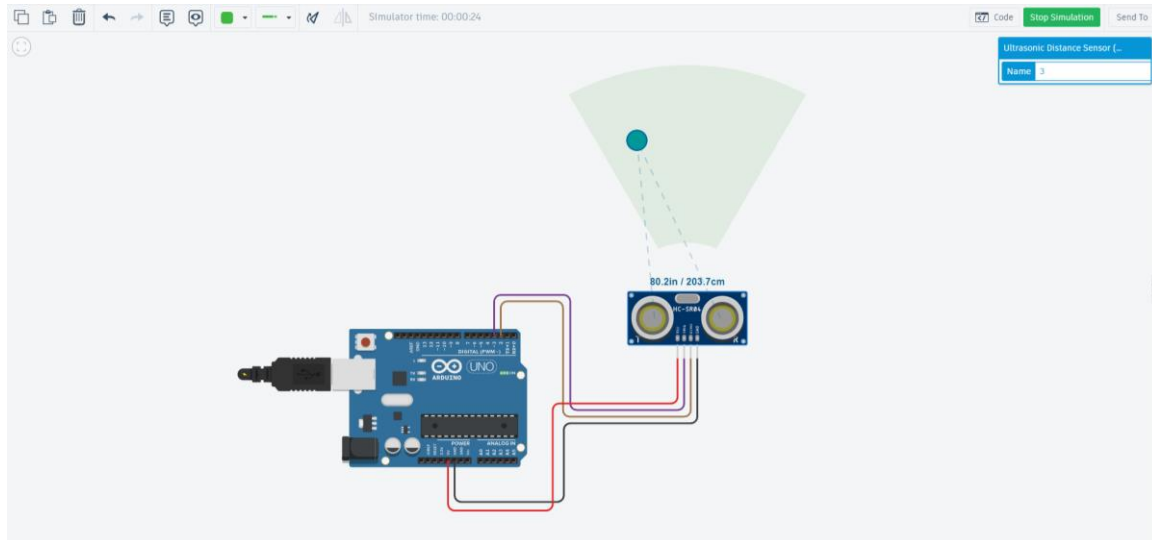
## Practical - 8

**Aim:** To measure distance using HC-SR04 sensor.

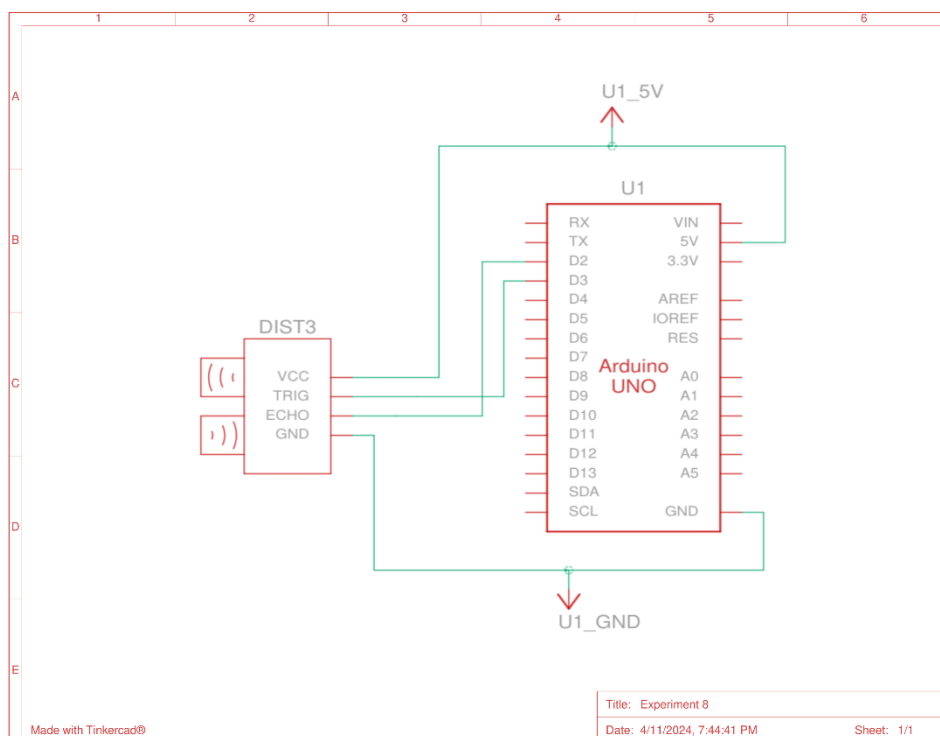
### **Introduction:**

This program demonstrates a practical application of IoT, where Arduino devices can be used for distance measurements. The HC-SR04 ultrasonic sensor is commonly used for this purpose. The sensor works by emitting ultrasonic pulses and then measuring the time it takes for the pulses to bounce back after hitting an object. This time measurement is directly proportional to the distance between the sensor and the object.

### **Circuit Diagram:**



### **Schematic View:**



## Key Components:

Component List			 Download CSV
Name	Quantity	Component	
U1	1	Arduino Uno R3	
DIST3	1	Ultrasonic Distance Sensor (4-pin)	

## Code:

```
1 #define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
2 #define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04
3
4 // defines variables
5 long duration; // variable for the duration of sound wave travel
6 int distance; // variable for the distance measurement
7
8 void setup() {
9   pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
10  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
11  Serial.begin(9600); // // Serial Communication is starting with 9600 of baudrate speed
12 }
13 void loop() {
14   // Clears the trigPin condition
15   digitalWrite(trigPin, LOW);
16   delayMicroseconds(2);
17   // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
18   digitalWrite(trigPin, HIGH);
19   delayMicroseconds(10);
20   digitalWrite(trigPin, LOW);
21   // Reads the echoPin, returns the sound wave travel time in microseconds
22   duration = pulseIn(echoPin, HIGH);
23   // Calculating the distance
24   distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)
25   // Displays the distance on the Serial Monitor
26   Serial.print("Distance: ");
27   Serial.print(distance);
28   Serial.println(" <0xa0>m");
29   delay(1000);
30 }
31
32
```

## Steps of Working:

6. Connect VCC of the sensor to 5V point of power on the Arduino.
7. Connect TRIG (Trigger) of the sensor to D3 point on the Arduino.
8. Connect ECHO (Echo) of the sensor to D2 point on the Arduino.
9. Connect GND (Ground) of the sensor with the ground (GND) on the Arduino.
10. Define the echo-pin D2 and trigger-pin D3 in your code.
11. Move the object and observe the change in distance.

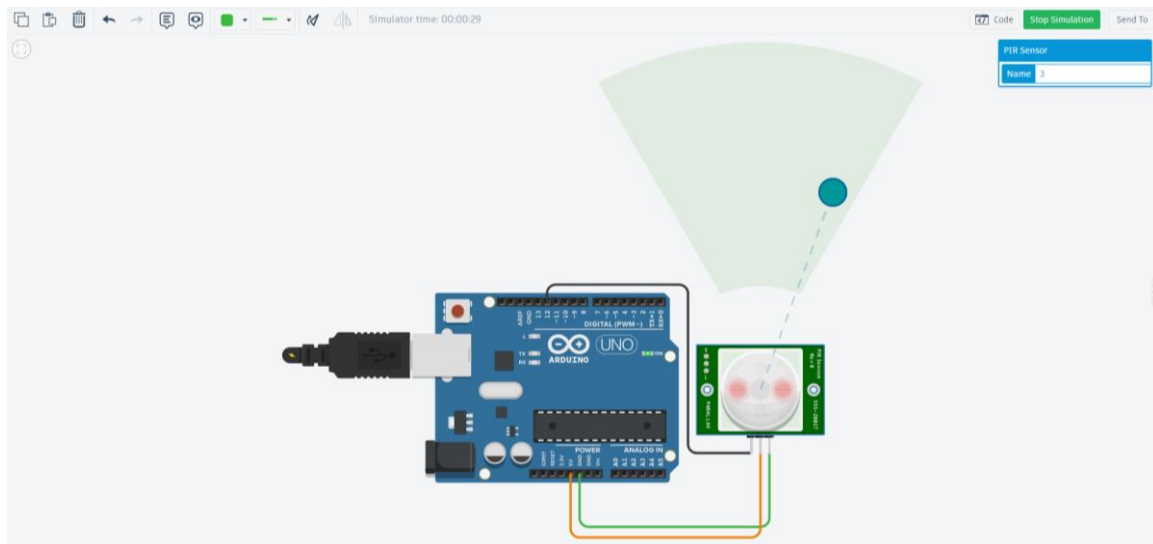
## Practical - 9

**Aim:** To make a motion detector using Passive Infrared Sensor (PIR) and Arduino.

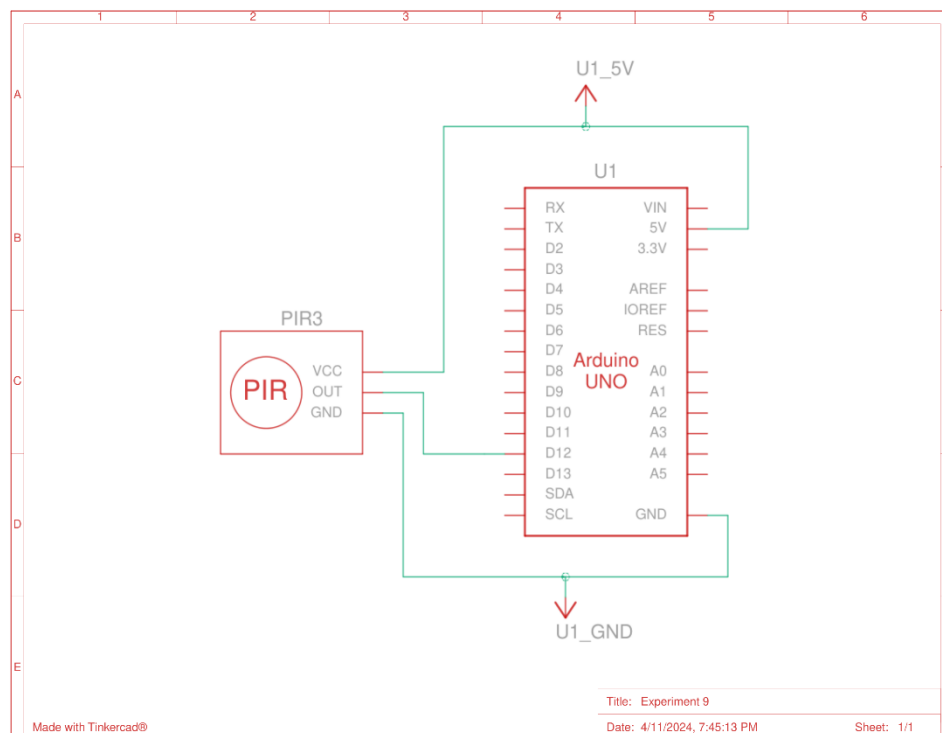
### **Introduction:**

A motion detector using a PIR sensor and Arduino is a useful project for detecting human or animal movement within a certain range. The PIR sensor detects changes in infrared radiation emitted by objects in its field of view, allowing it to detect motion without direct contact.

### **Circuit Diagram:**



### **Schematic View:**





## Key Components:

Component List			 Download CSV
Name	Quantity	Component	
U1	1	Arduino Uno R3	
PIR3	1	PIR Sensor	

## Code:

```
1 int pirPin = 12;
2
3 void setup() {
4   Serial.begin(9600);
5   pinMode(pirPin, INPUT);
6   Serial.println(pirPin);
7 }
8
9 void loop() {
10  int pirValue = digitalRead(pirPin);
11
12  if (pirValue == HIGH) {
13    Serial.println("Motion Detected");
14    delay(1000);
15  } else {
16    Serial.println("No motion detected.");
17    delay(1000);
18  }
19 }
20
21
```

## Steps of Working:

1. Connect Ground of the PIR sensor with the ground (GND) on the Arduino.
2. Connect Power of the PIR sensor to 5V point of power on the Arduino.
3. Connect Signal of the PIR sensor to the D12 points on Arduino.
4. Define the D12 pin in your code.
5. Move the object and motion will be detected in the sensor.

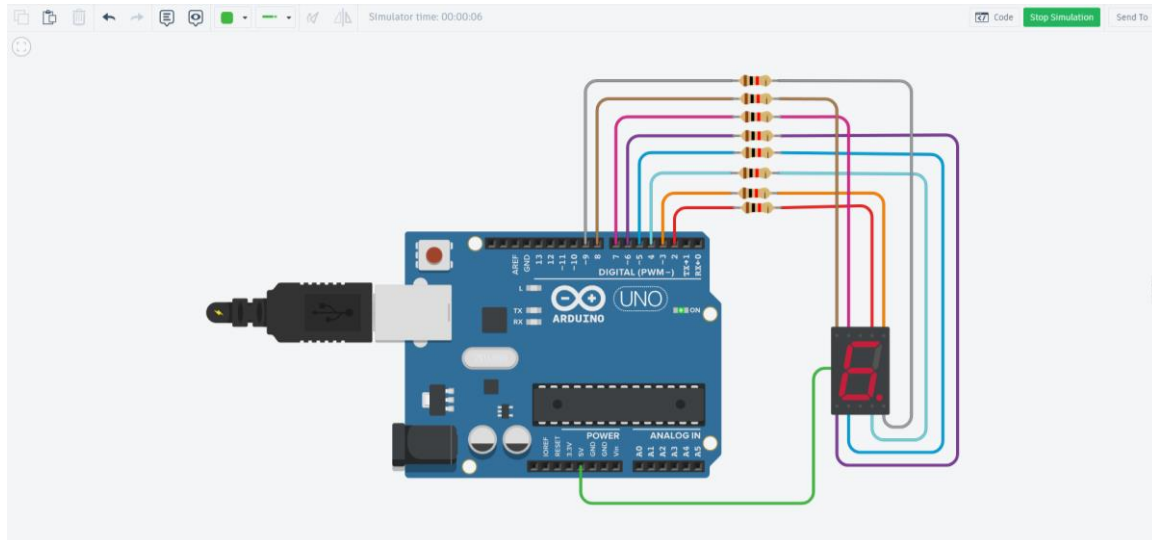
## Practical - 10

**Aim:** To interface 1-Digit 7-Segment display with Arduino.

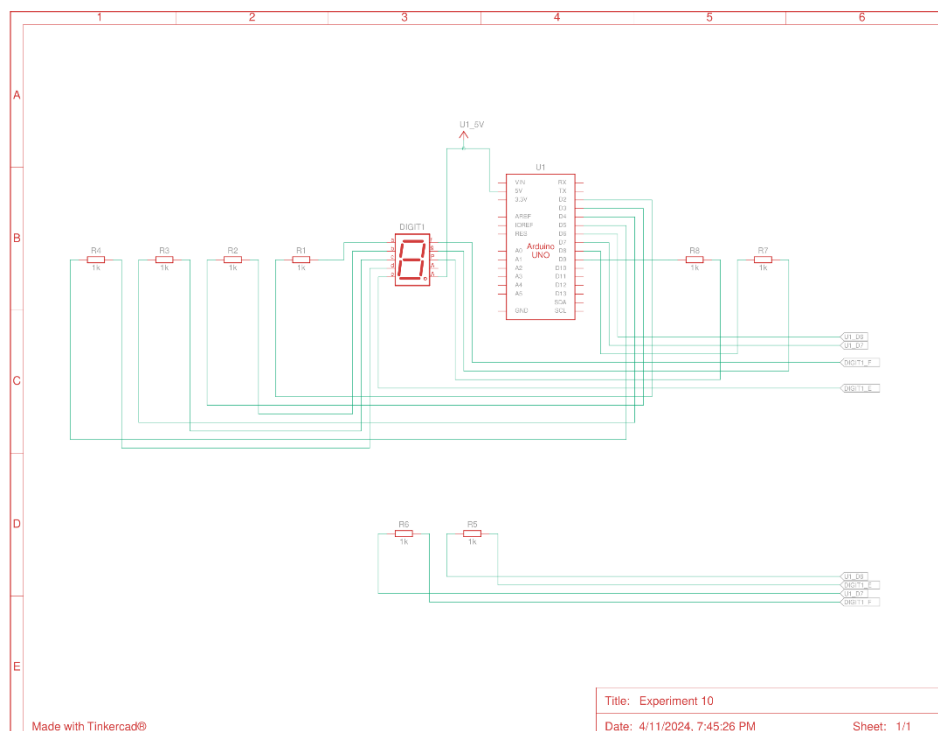
## Introduction:

A 7-segment display is a simple and commonly used component for displaying numerical digits. Each digit is composed of seven individually controllable segments, arranged in a specific pattern to represent numbers from 0 to 9. By cycling through different numbers and displaying them sequentially, you can create various visual effects or display numerical values obtained from sensors or user input.

### Circuit Diagram:



### Schematic View:



## Key Components:

Component List			Download CSV
Name	Quantity	Component	
U1	1	Arduino Uno R3	
Digit1	1	Anode 7 Segment Display	
R1 R2 R3 R4 R5 R6 R7 R8	8	1 kΩ Resistor	

## Code:

```
1 String numbers[10] = {
2   "1111101", // 0
3   "0110000", // 1
4   "1101101", // 2
5   "1111001", // 3
6   "0110011", // 4
7   "1011011", // 5
8   "1011111", // 6
9   "1110000", // 7
10  "1111111", // 8
11  "1111011", // 9
12 };
13
14 void setup() {
15   for (int i = 2; i < 10; i++) {
16     pinMode(i, OUTPUT);
17   }
18 }
19
20 void loop() {
21   for (int i = 0; i < 10; i++) {
22     displayDigit(i);
23     delay(1000);
24   }
25 }
26
27 void displayDigit(int digit) {
28   for (int i = 0; i < 8; i++) {
29     if (numbers[digit][i] == '1') {
30       digitalWrite(i+2, LOW);
31     } else {
32       digitalWrite(i+2, HIGH);
33     }
34   }
35 }
36
37
```

## Steps of Working:

1. Connect the 7 points (A, B, C, D, E, F, G) to the Digital pins of the Arduino.
2. Connect any one of the Common ports to 5V point of power on the Arduino.
3. You may also connect DP (for decimal), if the project demands the use of decimal.
4. Define all these ports in your code. You may use a loop, since number of ports is high.
5. Define the 8-bit patterns for all the digits.

## Precautions:

- To avoid excess current from entering into the 7-segment, which may damage the device if it is left running for a long time, we need to connect the 7-segment to the Arduino through a resistor (220Ω).

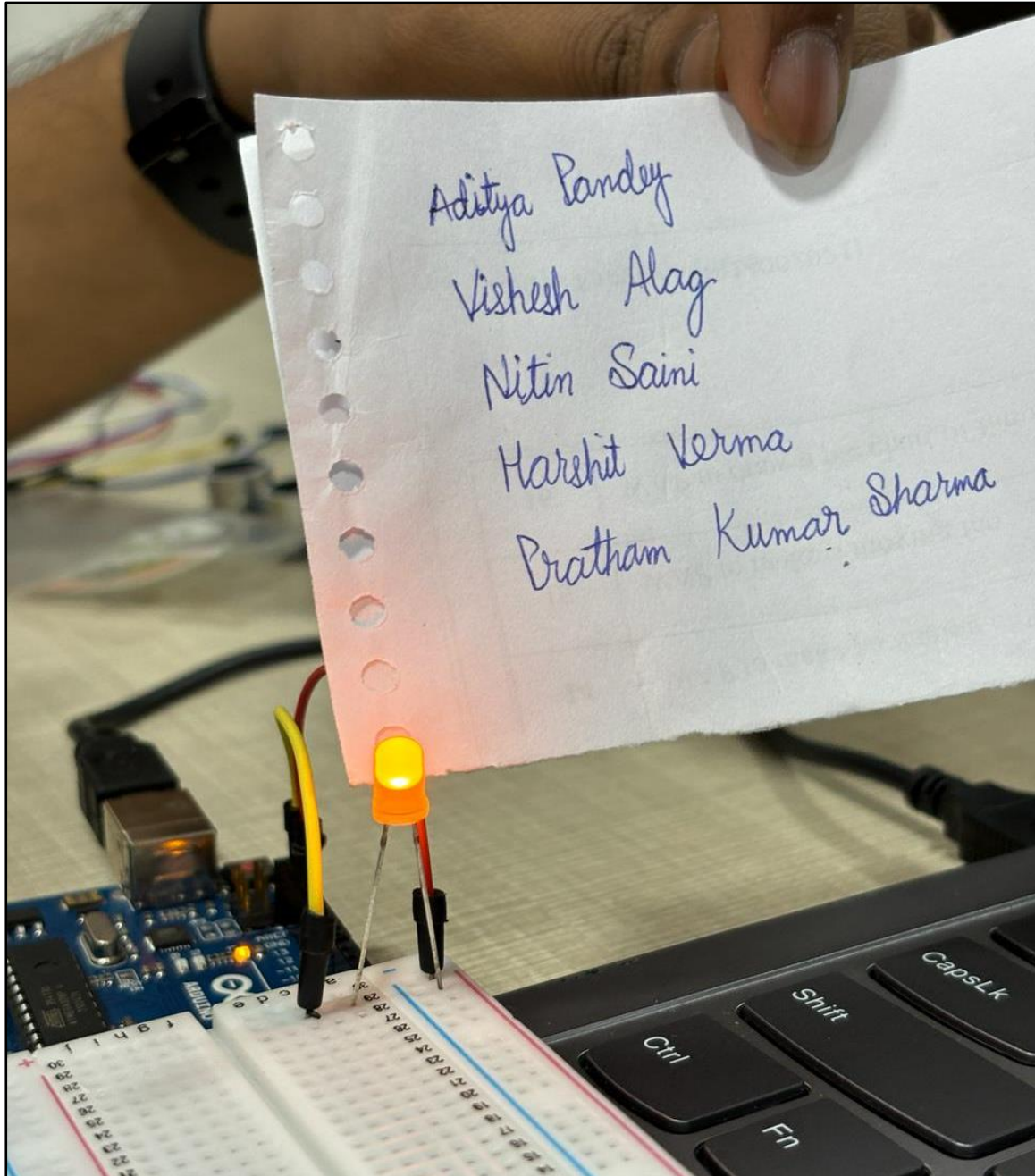
## Practical - 11

**Aim:** To demonstrate the working of several IoT devices on Arduino UNO using actual hardware.

### **Introduction:**

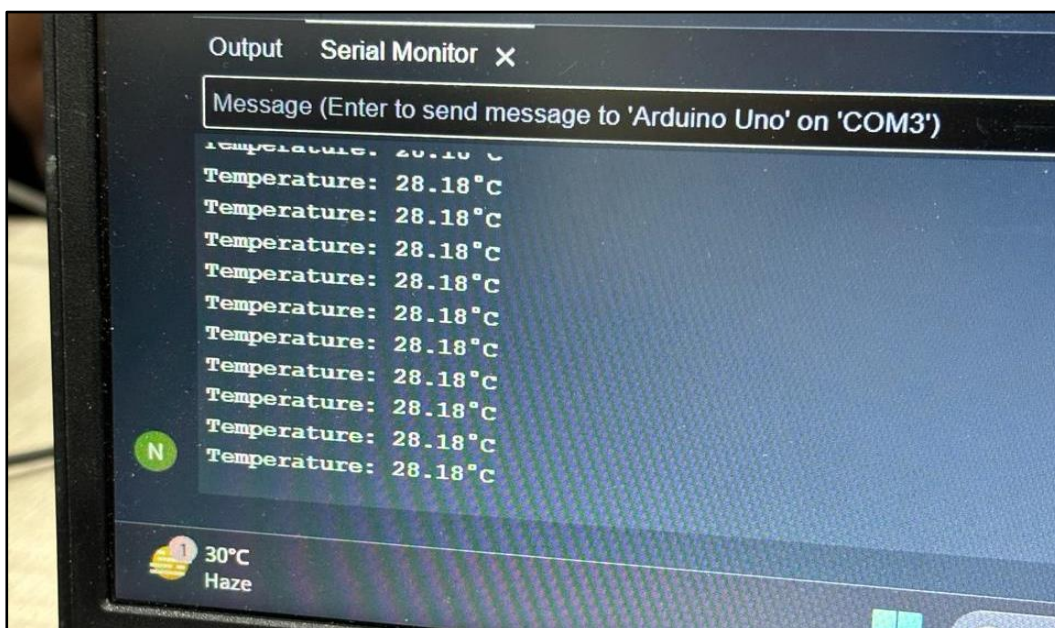
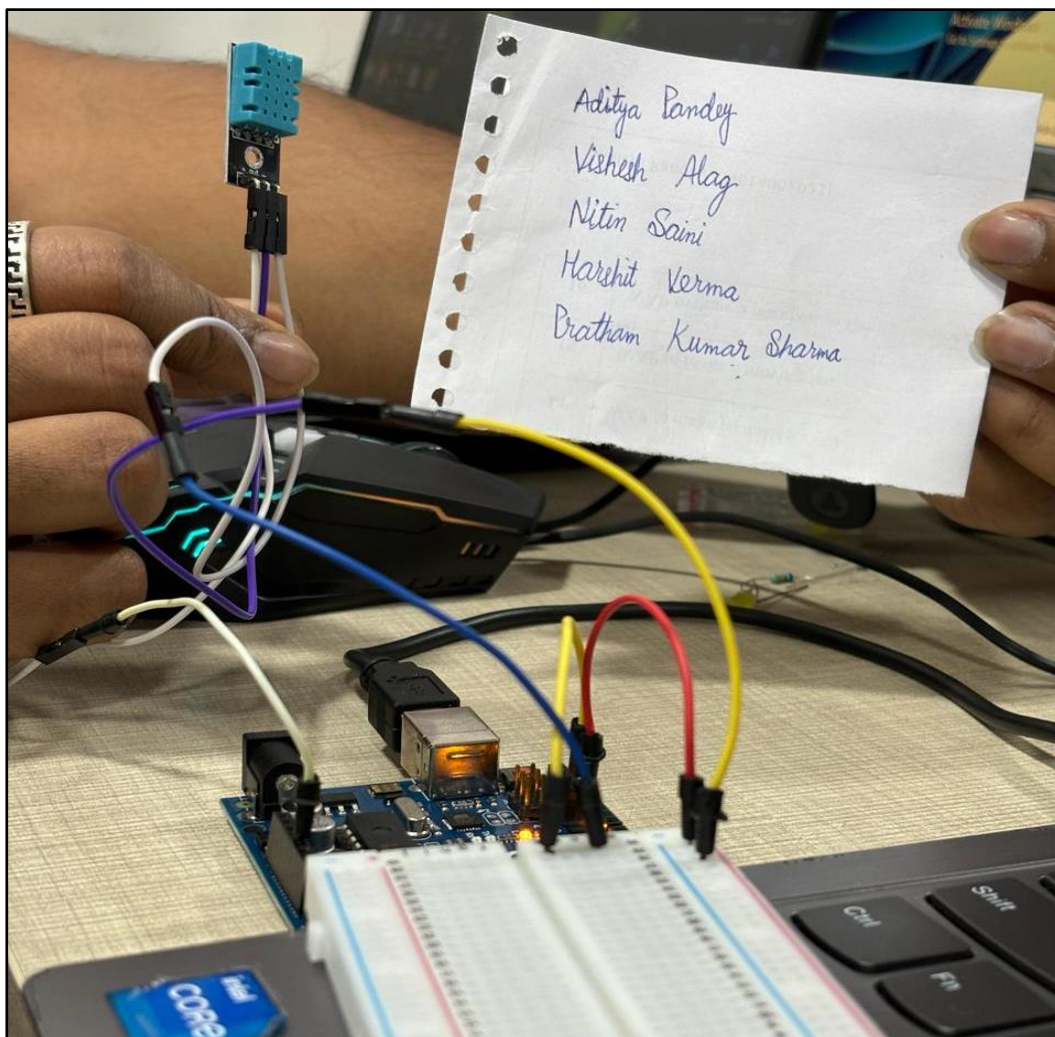
This experiment aims at demonstrating all the major Arduino functionalities, but by using actual hardware components, instead of simulating them on TinkerCAD.

**Experiment 1:** To demonstrate the working of LED light using Arduino.

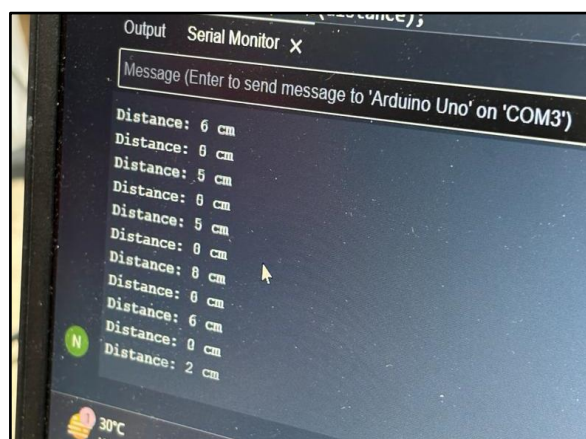
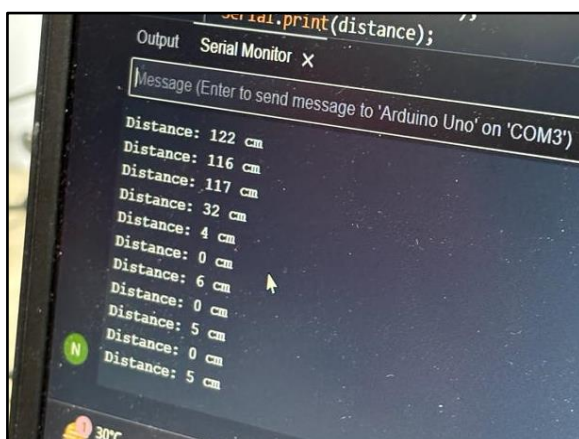
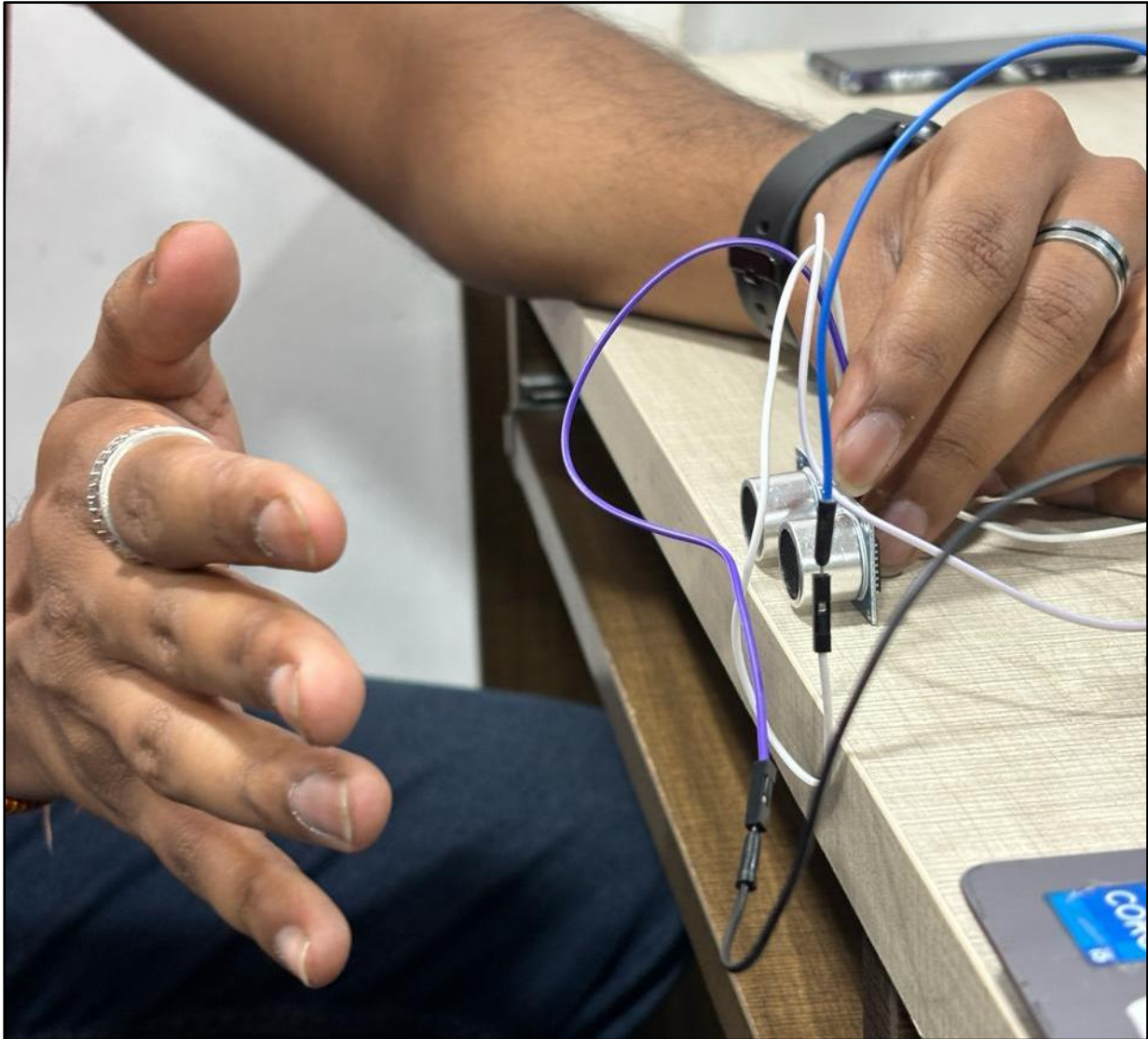




**Experiment 2:** To demonstrate the working of temperature sensor using Arduino.



**Experiment 3:** To measure distance using HC-SR04 sensor.



**Steps of Working:**

9. Steps of working for Experiment 1 are the same as Practical 1 (Blinking LED).
10. Steps of working for Experiment 2 are the same as Practical 6 (Temperature Sensor).
11. Steps of working for Experiment 3 are the same as Practical 8 (HC-SR04 sensor).

**Precautions:**

- ***Experiment 1:*** To avoid excess current from entering into the LED, which may damage the LED if it is left running for a long time, we need to connect the Anode of the LED to one of the terminals of a resistor ( $220\Omega$ ), and then connect the other terminal of the resistor to the D13 point on Arduino.

**Note:**

- All these experiments are already done in the Practical File. This practical consists of the same experiments, but with their respective hardware implementations.



## Practical - 12

**Aim:** To install libraries in the IDE of Arduino.

### Introduction:

Libraries in the Arduino Integrated Development Environment (IDE) are collections of pre-written code that provide additional functionality to Arduino projects. These libraries contain functions and classes that allow users to easily interface with external components, sensors, communication protocols, and other hardware peripherals without having to write complex code from scratch.

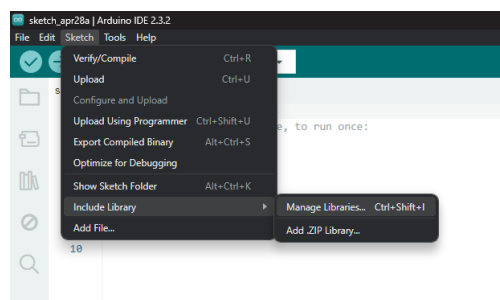
Libraries greatly simplify the process of programming Arduino projects by providing ready-to-use code for common tasks, such as controlling LEDs, reading sensor data, communicating with displays, and more. They help save time and effort by abstracting low-level details and providing high-level APIs (Application Programming Interfaces) that can be easily integrated into Arduino sketches.

Arduino IDE comes with a built-in library manager that allows users to search, install, and manage libraries directly from within the IDE. Additionally, users can create their own libraries or download libraries from third-party sources to extend the capabilities of their Arduino projects.

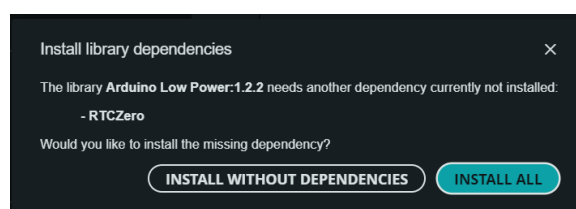
Overall, libraries play a crucial role in expanding the functionality and versatility of Arduino projects, making it easier for both beginners and advanced users to create innovative electronic devices and prototypes.

### Steps to install libraries in Arduino IDE:

1. Launch the Arduino IDE on your computer.
2. Go to the "Sketch" menu, then click on "Include Library" > "Manage Libraries...". This will open the Library Manager.

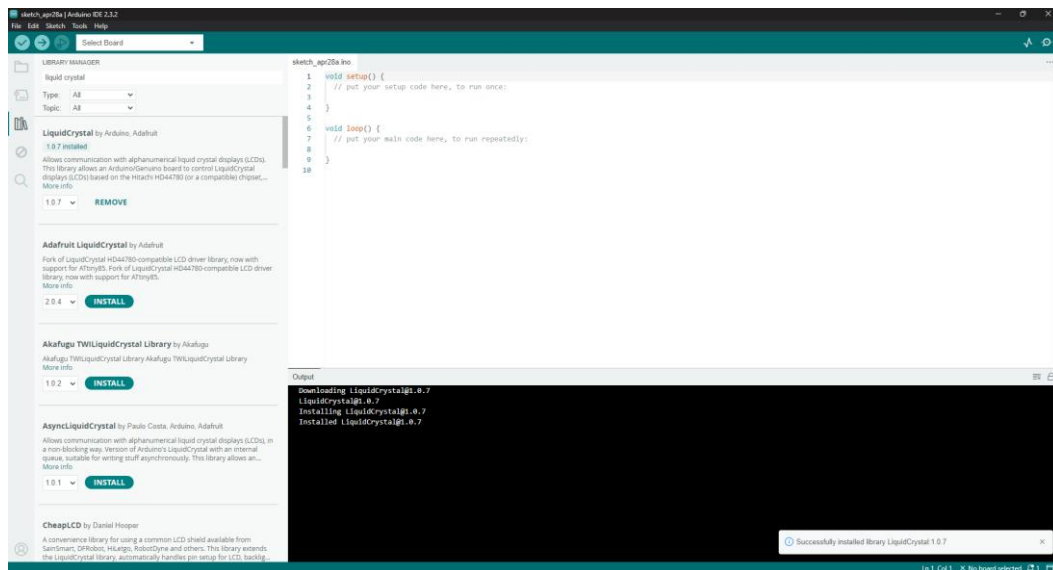


3. In the Library Manager window, you can search for the library you want to install. You can type the name of the library in the search bar to find it quickly.
4. Once you find the library you want to install, click on it. You'll see an "Install" button next to it. Click on the "Install" button to install the library.





5. The Arduino IDE will download and install the library. Depending on the size of the library and your internet connection speed, this might take a few seconds to a minute.
6. Once the installation is complete, you'll see a "**Installed**" label next to the library name in the Library Manager window.



7. Click the "**Close**" button to close the Library Manager.
8. You can now use the installed library in your Arduino sketches. You typically include the library at the top of your sketch using the **#include** directive.

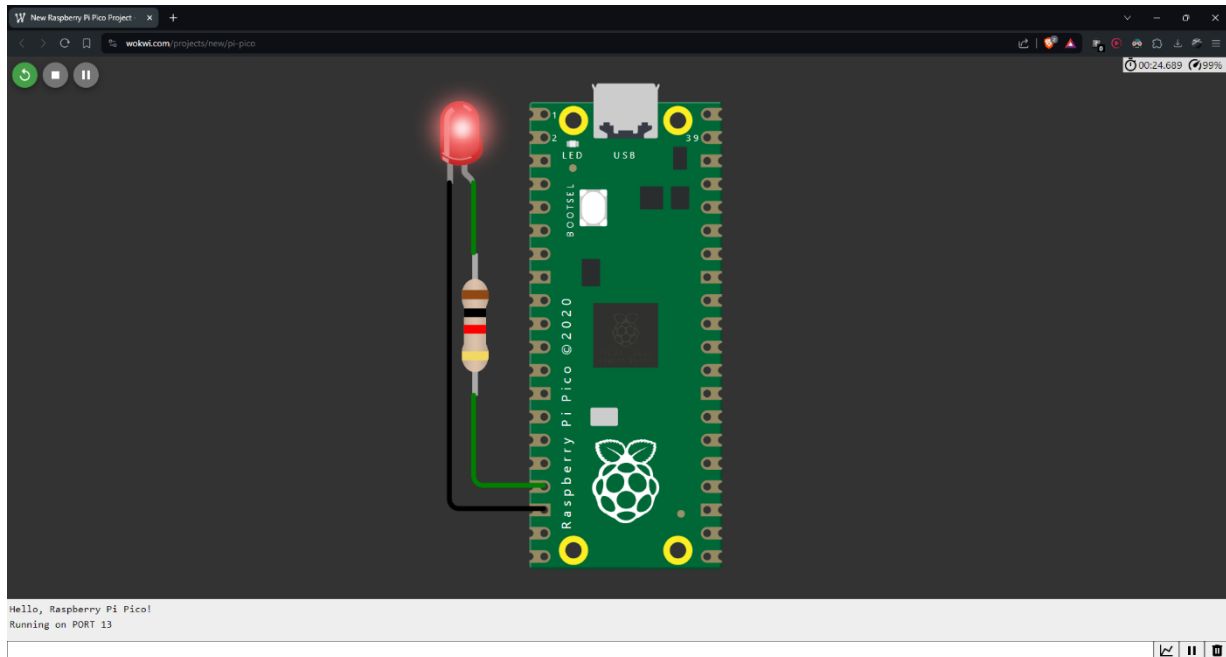
## Practical - 13

**Aim:** To blink an LED using Raspberry Pi.

### **Introduction:**

This experiment aims at demonstrating basic Raspberry Pi functionality by creating a simple circuit to blink an LED, and to create a functional circuit that demonstrates digital output control and serves as a foundational introduction to Raspberry Pi programming and hardware interaction for beginners.

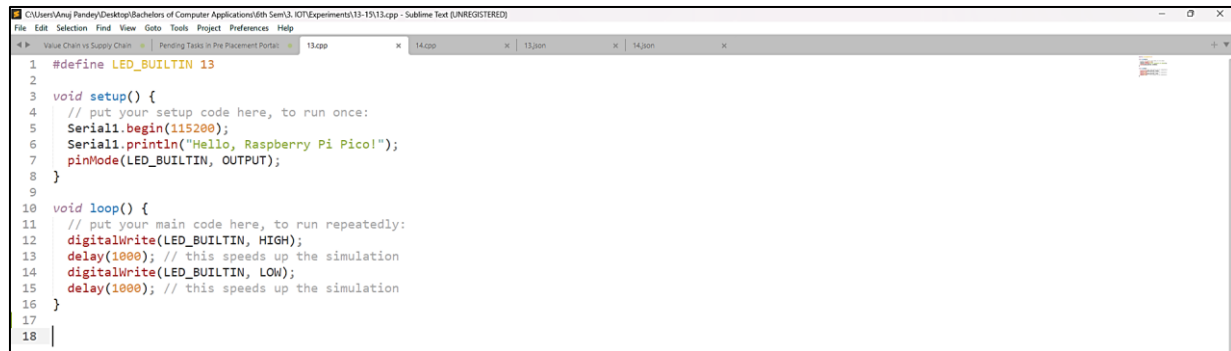
### **Circuit Diagram:**



### **Diagram & Connections/ Key Components:**

```
1 {
2   "version": 1,
3   "author": "Aditya Pandey",
4   "editor": "wokwi",
5   "parts": [
6     { "type": "wokwi-pi-pico", "id": "pico", "top": -3.15, "left": 42, "attrs": {} },
7     { "type": "wokwi-led", "id": "led1", "top": -3.6, "left": -5.8, "attrs": { "color": "red" } },
8     {
9       "type": "wokwi-resistor",
10      "id": "r1",
11      "top": 91.2,
12      "left": -10.15,
13      "rotate": 90,
14      "attrs": { "value": "1000" }
15    }
16  ],
17  "connections": [
18    [ "pico:GP0", "$serialMonitor:RX", "", [] ],
19    [ "pico:GP1", "$serialMonitor:TX", "", [] ],
20    [ "pico:GND.4", "led1:C", "black", [ "h0" ] ],
21    [ "led1:A", "r1:1", "green", [ "h9.6", "v28.8" ] ],
22    [ "r1:2", "pico:GP13", "green", [ "v37.2", "h19.2" ] ]
23  ],
24  "dependencies": {}
25 }
26
27
```

## Code:



```
1 #define LED_BUILTIN 13
2
3 void setup() {
4     // put your setup code here, to run once:
5     Serial1.begin(115200);
6     Serial1.println("Hello, Raspberry Pi Pico!");
7     pinMode(LED_BUILTIN, OUTPUT);
8 }
9
10 void loop() {
11     // put your main code here, to run repeatedly:
12     digitalWrite(LED_BUILTIN, HIGH);
13     delay(1000); // This speeds up the simulation
14     digitalWrite(LED_BUILTIN, LOW);
15     delay(1000); // This speeds up the simulation
16 }
17
18 |
```

## Steps of Working:

12. Connect the negative terminal (*Cathode*) of the LED with the ground (*GND*) on the board.
13. Connect the positive terminal (*Anode*) of the LED with GP13 point on the board.

## Precautions:

- To avoid excess current from entering into the LED, which may damage the LED if it is left running for a long time, we need to connect the Anode of the LED to one of the terminals of a resistor ( $220\Omega$ ), and then connect the other terminal of the resistor to the D13 point on the board.

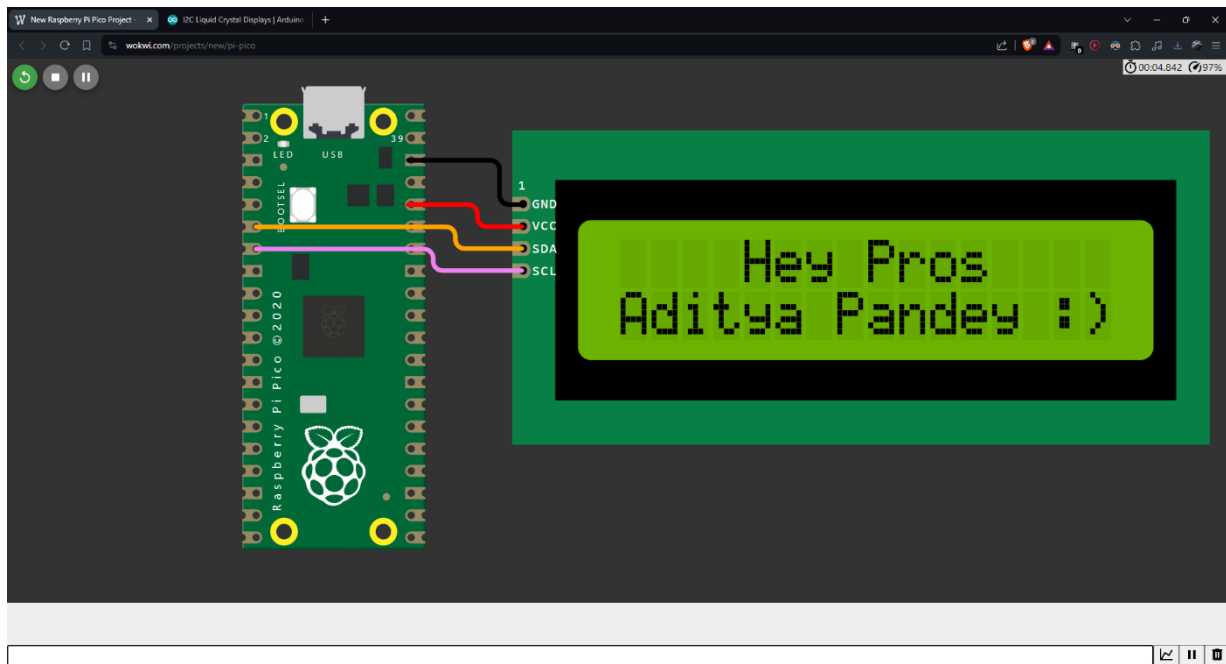
## Practical - 14

**Aim:** To display your name on an LCD screen using Raspberry Pi.

### **Introduction:**

This project aims at demonstrating the working of a Liquid Crystal Display (*LCD*) screen and displaying the student's name on it using Raspberry Pi. The students learn to connect and configure an LCD screen, interpret and execute code for customized text message, and comprehend the basics of character-based interfaces.

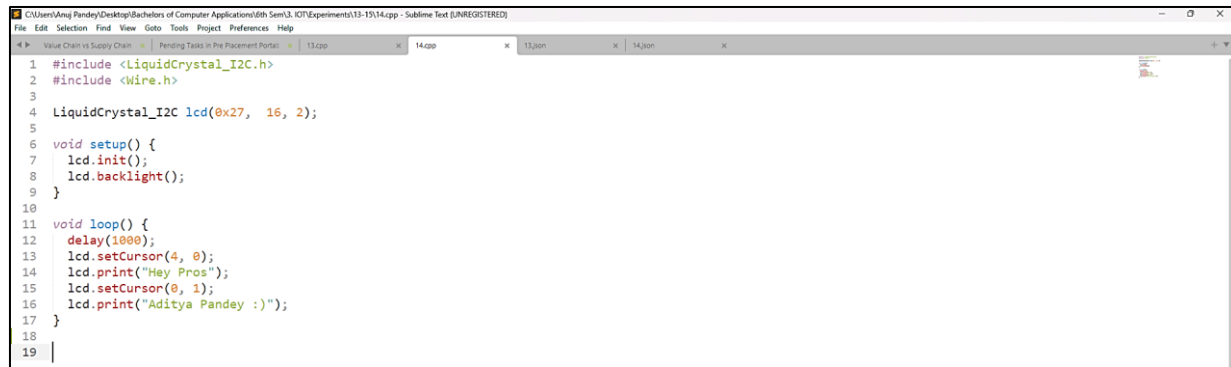
### **Circuit Diagram:**



### **Diagram & Connections/ Key Components:**



## Code:

A screenshot of a Sublime Text editor window. The title bar shows the file path: C:\Users\Anag Pandey\Desktop\Bachelors of Computer Applications\6th Sem\3. IoT\Experiments\13-19\14.cpp - Sublime Text [UNREGISTERED]. The editor contains an Arduino sketch for an LCD. The code includes the LiquidCrystal\_I2C and Wire libraries, initializes an LCD at address 0x27, and prints "Hey Pros" and "Aditya Pandey :)" on the screen. The code is as follows:

```
1 #include <LiquidCrystal_I2C.h>
2 #include <Wire.h>
3
4 LiquidCrystal_I2C lcd(0x27, 16, 2);
5
6 void setup() {
7   lcd.init();
8   lcd.backlight();
9 }
10
11 void loop() {
12   delay(1000);
13   lcd.setCursor(4, 0);
14   lcd.print("Hey Pros");
15   lcd.setCursor(0, 1);
16   lcd.print("Aditya Pandey :)");
17 }
18
19
```

## Steps of Working:

1. Connect GND of the LCD screen to the GND on the board.
2. Connect VCC of the LCD screen to the 3V point of power on the board.
3. Connect SDA of the LCD screen to the GP4 on the board.
4. Connect SCL of the LCD screen to the GP4 on the board.
5. To make this setup functional, we use *LiquidCrystal.h* and *Wire.h* libraries in the code.

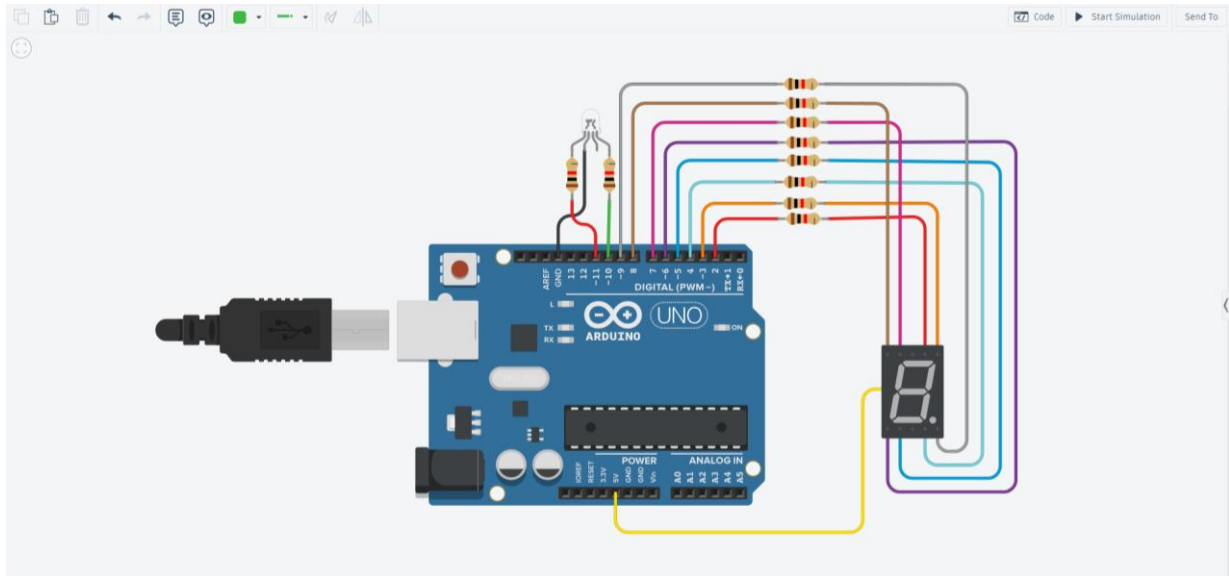
## Practical - 15

**Aim:** To demonstrate the working of the traffic lights using 1-Digit 7-Segment Display and an RGB LED.

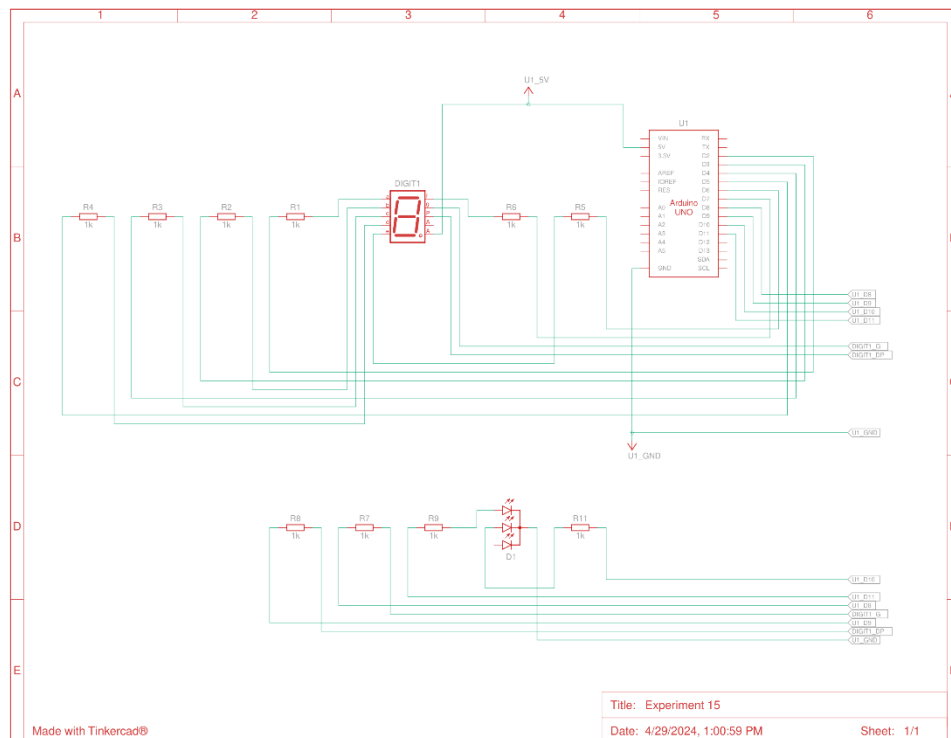
### **Introduction:**

In this project, we delve into the heart of traffic management, employing Arduino to simulate the operation of a traffic light system. Through the integration of a 1-Digit 7-Segment Display and an RGB LED, we will visualize the state transitions of the traffic lights, while assuming that traffic light changes its colour after exactly 10 seconds, and the countdown for each colour is shown in the 7-Segment Display.

### **Circuit Diagram:**



### **Schematic View:**



## Key Components:

Component List			Download CSV
Name	Quantity	Component	
U1	1	Arduino Uno R3	
Digit1	1	Anode 7 Segment Display	
R1 R2 R3 R4 R5 R6 R7 R8 R9 R11	10	1 kΩ Resistor	
D1	1	LED RGB	

## Code:

```
1 String numbers[10] = {
2   "11111101", // 0
3   "01100000", // 1
4   "11011011", // 2
5   "11110010", // 3
6   "01100111", // 4
7   "10110110", // 5
8   "10111111", // 6
9   "11100000", // 7
10  "11111111", // 8
11  "11110110", // 9
12 };
13 int x = 0;
14
15 void setup() {
16   for (int i = 2; i < 10; i++) {
17     pinMode(i, OUTPUT); analogWrite(10, 255);
18   }
19 }
20
21 void loop() {
22   for (int i = 9; i >= 0; i--) {
23     displayDigit(i); delay(1000);
24   }
25   if (x == 0) {
26     analogWrite(10, 100); analogWrite(11, 255);
27   } else if (x == 1) {
28     analogWrite(10, 0); analogWrite(11, 255);
29   } else {
30     x = -1;
31     analogWrite(10, 255); analogWrite(11, 0);
32   }
33   x++;
34 }
35
36 void displayDigit(int digit) {
37   for (int i = 0; i < 8; i++) {
38     if (numbers[digit][i] == '1') { digitalWrite(i+2, LOW); }
39     else { digitalWrite(i+2, HIGH); }
40   }
41 }
42 }
```

## Steps of Working:

1. Connect 1-Digit 7-Segment Display with the Arduino board in the same way as Experiment 10.
2. Connect Red node of the RGB LED to D11 point on the board.
3. Connect Green node of the RGB LED to D10 point on the board.
4. Connect the negative terminal (*Cathode*) of the RGB LED with the ground (*GND*) on the board.
5. Note that we don't connect Blue node of the RBG LED to the Arduino, as we don't need it in the traffic signal.