Group A: 01

Experiment No: 01

Study of Microcontrollers and Single-Board Computers – History and Evolution

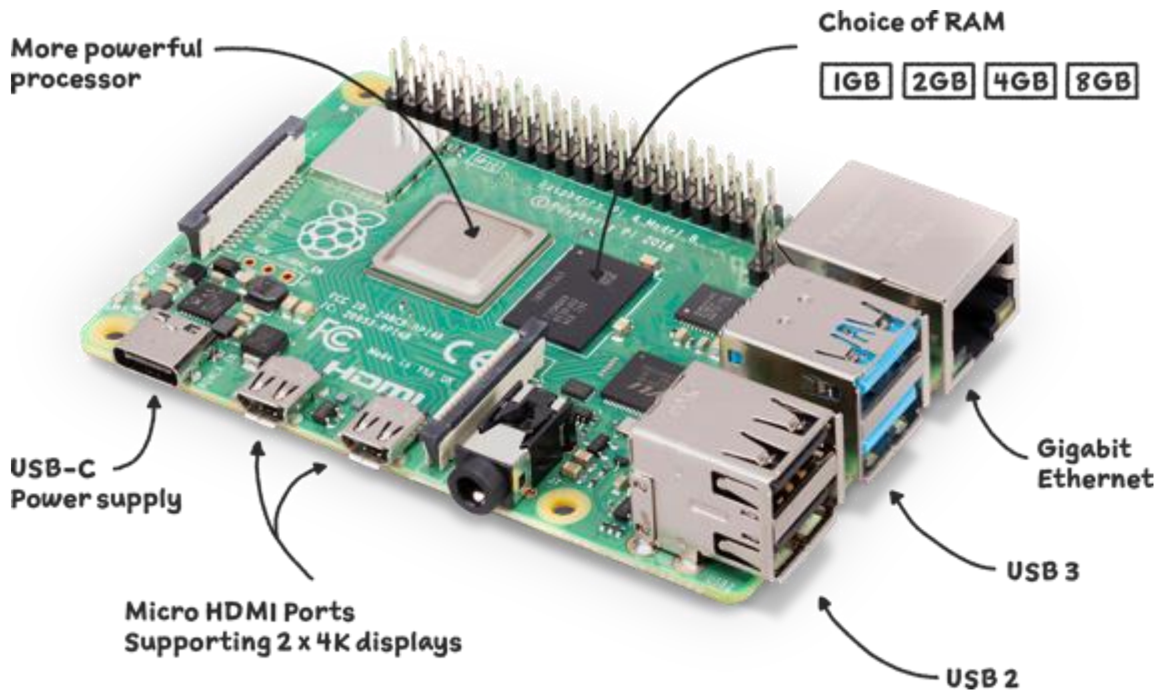
Problem Statement:

Study of Raspberry-Pi/ Beagle board/ Arduino and other microcontroller (History & Elevation)

Theory:

A. Raspberry-Pi:

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries. It does not include peripherals (such as keyboards and mice). The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. The Raspberry Pi is a credit-card-sized computer that costs between $5 and $35. It's available anywhere in the world, and can function as a proper desktop computer or be used to build smart devices. A Raspberry Pi is a general-purpose computer, usually with a Linux operating system, and the ability to run multiple programs. Raspberry Pi is like the brain. Its primary advantage comes in processing higher level processing capability. It's a single board computer.



Evolution of Raspberry Pi

- **Raspberry Pi 1 (2012):** First model, Model B released in Feb 2012; followed by simpler Model A.

- **Improvements (2014):** Model B+ introduced with better design; A+ and B+ followed a year later.

- **Compute Model (2014):** Released for embedded applications.

- **Raspberry Pi 2 (2015):** More RAM, quad-core Cortex-A7 CPU, 4-6 times more powerful than original.

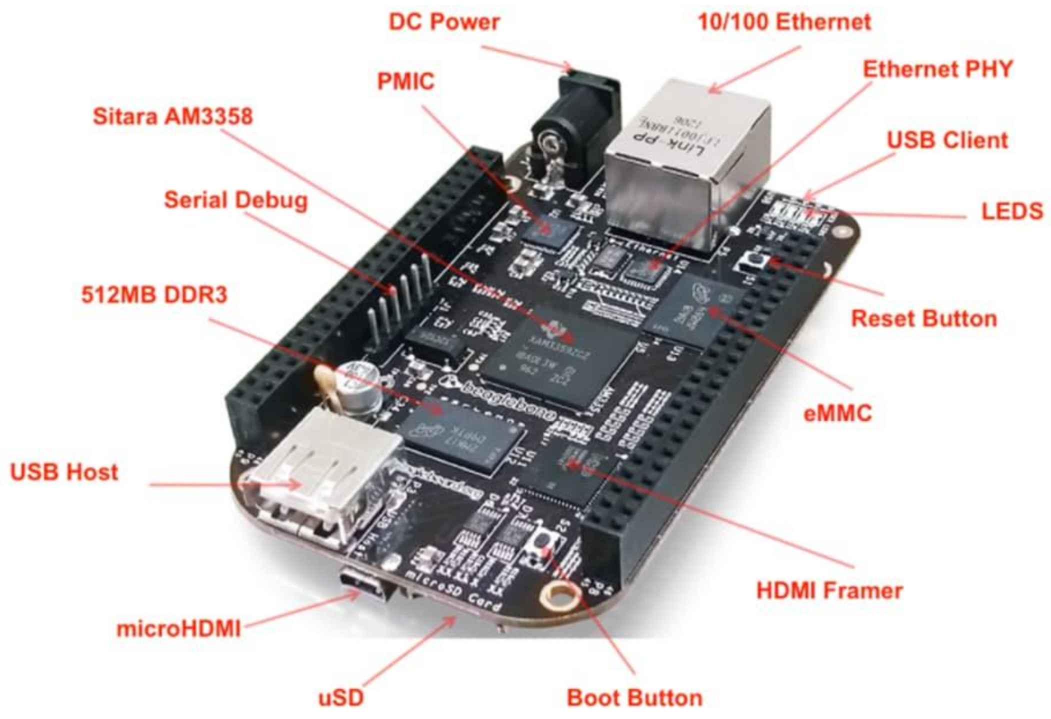
- **Raspberry Pi Zero (2015):** Smaller size, reduced I/O, priced at $5.

- **Raspberry Pi 3 (2016):** Quad-core Cortex-A53, onboard Wi-Fi, Bluetooth, USB boot; 10x performance of Pi 1.

- **Raspberry Pi Zero W (2017):** Zero model with Wi-Fi and Bluetooth, priced at $10.

B. Beagle Board:

Beagle board :- The Beagle Board is a low-power open-source single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The Beagle Board was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip. The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the Creative Commons share-alike license. The board was designed using Cadence OrCAD for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used. Beagle Bone Black is a low-cost, open source, community-supported development platform for ARM® Cortex™-A8 processor developers and hobbyists. Boot Linux in under 10-seconds and get started on Sitara™ AM335x ARM Cortex-A8 processor development in less than 5 minutes with just a single USB cable.

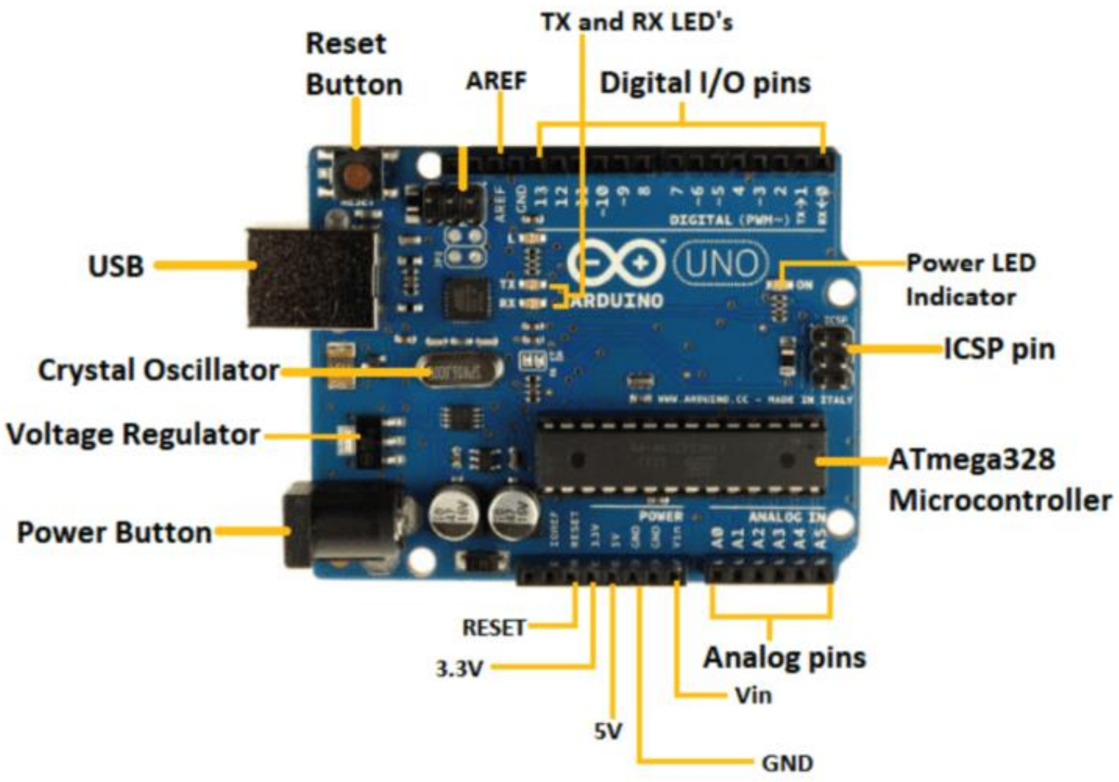


<u>**Evoultuion of BeagleBone/ BeagleBoard**</u>

- **BeagleBone (2011):** ARM Cortex-A8 @ 720 MHz, 256 MB RAM, priced $89.
- **BeagleBone Black (2013):** Upgraded to 1 GHz CPU, 512 MB RAM, HDMI, 2 GB flash, priced $45.
- **BeagleBone Black Revision C (2014):** Flash memory increased to 4 GB, shipped with Debian Linux.

C. Arduino:

Arduino is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices and interactive objects that can sense and control objects in the physical and digital world. Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboards (shields) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers are typically programmed using a dialect of features from the programming languages C and C++. In addition to using traditional compiler tool chains, the Arduino project provides an integrated development environment (IDE) based on the Processing language project. Arduino is open-source hardware. The hardware reference designs are distributed under a Creative Commons Attribution Share-Alike 2.5 license and are available on the Arduino website. Layout and production files for some versions of the hardware are also available.



Evolution of Arduino :

Arduino was created in Ivrea Italy as a Masters thesis project. The goal of Arduino was to allow non-technical individuals to create technical projects of their own.

The Arduino was intended to be affordable. Over 700,000 Arduino boards have been commercially produced since its founding.

1. Serial Arduino

- **Release Year:** 2005
- **Processor:** ATmega8
- **Frequency:** 16 MHz
- **Host Interface:** DE-9 Serial Connection (RS232)
- Designed for simple assembly with minimal components.

2. Arduino Nano

- **Release Date:** May 15, 2008
- **Processor:** ATmega328 (ATmega168 before v3.0)
- **Frequency:** 16 MHz
- **Host Interface:** USB (Mini-B USB)

- Uses surface-mounted processor; no DC power jack.

3. Arduino UNO

- **Release Date:** September 24, 2010
- **Processor:** ATmega328P
- **Frequency:** 16 MHz
- **Host Interface:** USB (with FTDI chip)

Difference between Microprocessor and Microcontroller

Feature	**Microprocessor**	**Microcontroller**
Definition	A CPU on a single chip; requires external components like memory and I/O.	A complete system on a chip with CPU, memory, and I/O ports integrated.
Components	Only the processor (CPU); RAM, ROM, and I/O are connected externally.	Includes CPU, RAM, ROM, timers, and I/O ports all on one chip.
Cost	Generally more expensive due to the need for external components.	Cost-effective and compact due to integration.
Power Consumption	Higher, as multiple chips are used.	Lower, optimized for embedded systems.
Speed	Faster, suitable for high-performance applications.	Slower, suitable for control-oriented applications.
Example	Intel Core i7, AMD Ryzen	8051, ATmega328 (Arduino), PIC16F877A

The detailed comparison between Arduino, Raspberry Pi, and BeagleBone, three popular platforms for electronics and embedded systems

Feature	**Arduino**	**Raspberry Pi**	**BeagleBone**
Type	Microcontroller	Single-board computer (SBC)	Single-board computer (SBC)
Processor	AVR (e.g., ATmega328P) or ARM Cortex	ARM Cortex-A (Quad-core or more)	ARM Cortex-A8
Operating System	None (bare-metal programming)	Linux-based OS (e.g., Raspbian, Ubuntu)	Linux (Debian, Ubuntu)

Programming Language	C/C++ via Arduino IDE	Python, C/C++, Java, etc.	Python, C/C++, Bash, etc.
Connectivity	Limited (no built-in Wi-Fi or Ethernet in basic models)	Built-in Wi-Fi, Ethernet (varies by model)	Built-in Ethernet, USB host, serial
I/O Pins	~14 Digital, ~6 Analog	26–40 GPIO (digital only)	65+ GPIO, multiple ADCs
Processing Power	Low (16–84 MHz)	High (1.2–1.8 GHz)	Medium (1 GHz)
Memory	Few KBs (RAM & Flash)	512MB to 8GB RAM, SD card storage	512MB RAM, 4GB onboard eMMC
Best Use Case	Real-time control, sensors, simple automation	Full OS applications, media, networking	Industrial applications, robotics, PRUs
Real-Time Capability	Yes (bare-metal control)	No (non-real-time OS)	Limited, but includes PRU (Programmable Real-time Unit)
Power Consumption	Very low	Moderate	Low to Moderate
Typical Projects	Blinking LEDs, motor control, temperature monitoring	Web server, camera interface, AI applications	Industrial automation, high-performance GPIO projects
Price	₹500 – ₹1000	₹3000 – ₹7000	₹6000 – ₹8500

Conclusion:

The experiment provided a comparative understanding of the history and evolution of Arduino, Raspberry Pi, BeagleBoard, and other microcontrollers, highlighting their significance in embedded systems and modern electronics.

Group A: 02

Experiment No: 02

Understanding Operating Systems

Problem Statement:

Study of different operating systems for Raspberry-Pi /Beagle board/Arduino. Understanding the process of OS installation.

Theory:

1) **Raspberry-Pi:** - The Pi can run the official Raspbian OS, Ubuntu Mate, Snappy Ubuntu Core, the Kodi- based media centers OSMC and LibreElec, the non-Linux based Risc OS (one for fans of 1990s Acorn computers). It can also run Windows 10 IoT Core, which is very different to the desktop version of Windows, as mentioned below.

- **OS which install on Raspberry-Pi:** Raspbian, Ubuntu MATE, Snappy Ubuntu, Pidora, Linutop, SARPi, Arch Linux ARM, Gentoo Linux, etc.

- **How to install Raspbian on Raspberry-Pi:**

  Step 1: Download Raspbian

  Step 2: Unzip the file. The Raspbian disc image is compressed, so you'll need to unzip it. The file uses the ZIP64 format, so depending on how current your built-in utilities are, you need to use certain programs to unzip it.

  Step 3: Write the disc image to your microSD card. Next, pop your microSD card into your computer and write the disc image to it. The process of actually writing the image will be slightly different across these programs, but it's pretty self-explanatory no matter what you're using. Each of these programs will have you select the destination (make sure you've picked your microSD card!) and the disc image (the unzipped Raspbian file). Choose, double-check, and then hit the button to write.

  Step 4: Put the microSD card in your Pi and boot up. Once the disc image has been written to the microSD card, you're ready to go! Put that sucker into your Raspberry Pi, plug in the peripherals and power source, and enjoy. The current edition to Raspbian will boot directly to the desktop. Your default credentials are username pi and password raspberry.

2) **BeagleBone Black: -** The BeagleBone Black includes a 2GB or 4GB on-board eMMC flash memory chip. It comes with the Debian distribution factory pre-installed. You can flash new operating systems including Angstrom, Ubuntu, Android, and others.

- **Os which install on BeagleBone Black:** Angstrom, Android, Debian, Fedora, Buildroot, Gentoo, Nerves Erlang/OTP, Sabayon, Ubuntu, Yocto, MINIX 3

- **How to install Debian on BeagleBone Black:**

  Step 1: Download Debian img.xz file.

  Step 2: Unzip the file.

  Step 3: Insert your MicroSD (uSD) card into the proper slot. Most uSD cards come with a full-sized SD card that is really just an adapter. If this is what you have then insert the uSD into the adapter, then into your card reader.

  Step 4: Now open Win32 Disk imager, click the blue folder icon, navigate to the debian img location, and double click the file. Now click Write and let the process complete. Depending on your processor and available RAM it should be done in around 5 minutes.

  Step 5: Alright, once that's done, you'll get a notification pop-up. Now we're ready to get going. Remove the SD adapter from the card slot, remove the uSD card from the adapter. With the USB cable disconnected insert the uSD into the BBB.

  Step 6: Now, this next part is pretty straight forward. Plug the USB cable in and wait some more. If everything is going right you will notice that the four (4) leds just above the USB cable are doing the KIT impression. This could take up to 45 minutes, I just did it again in around 5 minutes. Your mileage will vary. Go back and surf reddit some more.

  Step 7: If you are not seeing the leds swing back and forth you will need to unplug the USB cable, press and hold down the user button above the uSD card slot (next to the 2 little 10 pin ICs) then plug in the USB cable. Release the button and wait. You should see the LEDs swinging back and forth after a few seconds. Once this happens it's waiting time. When all 4 LEDs next to the USB slot stay lit at the same time the flash process has been completed.

  Step 8: Remove the uSD card and reboot your BBB. You can reboot the BBB by removing and reconnecting the USB cable, or hitting the reset button above the USB cable near the edge of the board.

  Step 9: Now using putty, or your SSH flavor of choice, connect to the BBB using the IP address 192.168.7.2. You'll be prompted for a username. Type root and press Enter. By default, there is no root password. I recommend changing this ASAP if you plan on putting your BBB on the network. To do this type password, hit enter, then enter your desired password. You will be prompted to enter it again to verify.

3) **Arduino: -** The Arduino itself has no real operating system. You develop code for the Arduino using the Arduino IDE which you can download from Arduino - Home. Versions are available for Windows, Mac and Linux. The Arduino is a constrained microcontroller.

   Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. You are literally writing the "firmware" when you write the code and upload it. It's both good and its bad.

Conclusion:

The experiment enabled a clear understanding of different operating systems compatible with Raspberry Pi and BeagleBoard, along with the process of OS installation, and emphasized the contrast with firmware-based platforms like Arduino.

Group A: 03

Experiment No: 03

Temperature Monitoring with Buzzer Notification

Problem Statement:

Write an application to read temperature from the environment. If temperature crosses threshold value then it notifies with buzzer

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
DHT11 (3-pin module)	1	Already includes internal pull-up resistor
Buzzer	1	To sound an alert when temperature exceeds threshold
Breadboard	1	For easy wiring (optional if direct jumpers used)
Male-to-Male Jumper Wires	~6	For power, ground, and signal connections
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

DHT11 (3-pin version):

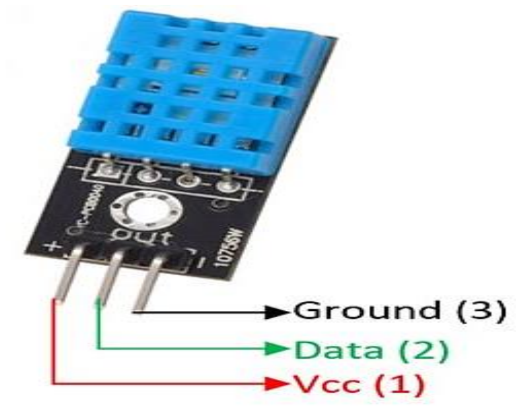
VCC (Pin 1) → Arduino 5 V
DATA (Pin 2) → Arduino Digital Pin 7
GND (Pin 3) → Arduino GND

Buzzer:

- (Positive) → Arduino Digital Pin 3
  – (Negative) → Arduino GND

Theory:

The DHT11 sensor is a digital temperature and humidity sensor widely used in electronics projects involving environmental monitoring. It provides temperature data in degrees Celsius and humidity as a percentage. The sensor module has three pins: VCC, DATA, and GND. Internally, it uses a capacitive humidity sensor and a thermistor to measure surrounding air conditions and sends the data to a microcontroller via a single digital line. The 3-pin module version includes an onboard pull-up resistor, which simplifies wiring and ensures reliable data communication with the Arduino or any other microcontroller.



This sensor is highly popular in Arduino-based projects due to its low cost, ease of use, and decent accuracy for basic applications. It operates on a 3.3V to 5.5V power supply and provides updated readings every second (1Hz sampling rate). Though not suitable for high-precision applications, the DHT11 serves well in educational setups, hobby electronics, and basic automation projects like weather stations, temperature alarms, and indoor climate monitoring systems.

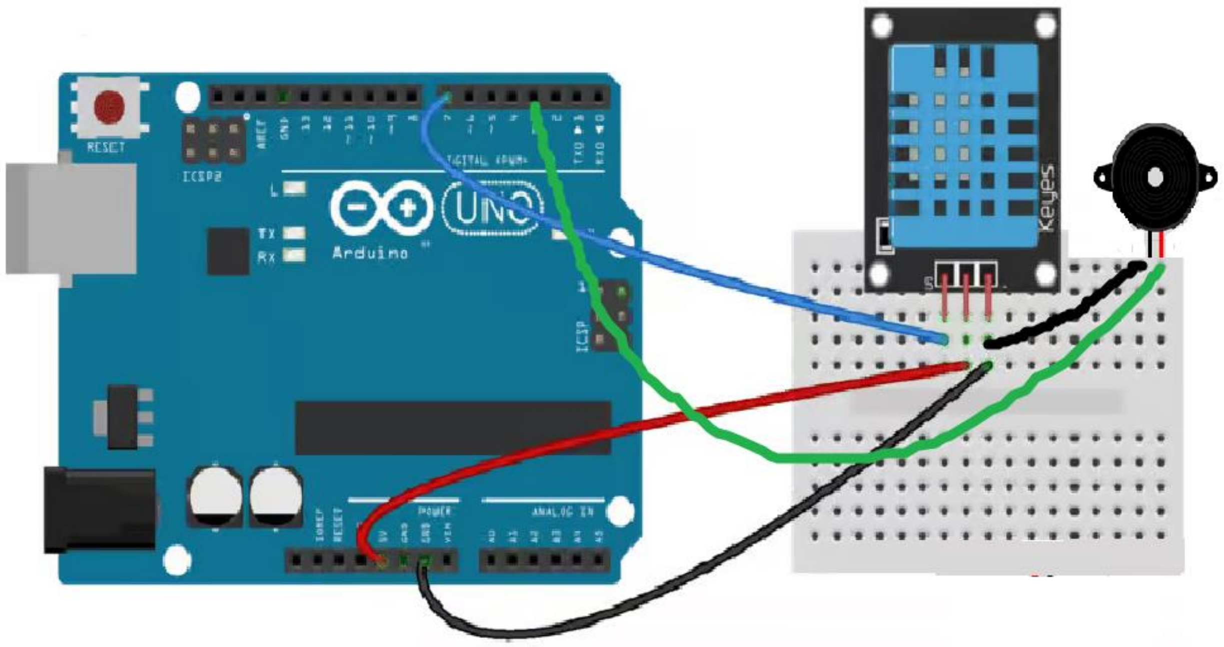
This application reads the ambient temperature using a DHT11 sensor and activates a buzzer if the temperature exceeds a predefined threshold value (30°C). The DHT11 sensor provides digital temperature readings, which are read periodically. The buzzer is used as an audio alert mechanism to notify that the environmental condition is beyond the acceptable range.

In terms of logic, the program initializes serial communication and sets up the DHT11 and buzzer pin. In the main loop, it waits for 2 seconds between readings to match the sensor's response time. The dht.readTemperature() function retrieves the temperature in Celsius. If the reading is valid and greater than the threshold, the buzzer is activated using digitalWrite() with HIGH signal; otherwise, it is turned OFF.

This application makes use of:

- dht.readTemperature() to get real-time temperature data.
- digitalWrite() to control the buzzer state.
- isnan() to check for invalid readings.
- Constants to define threshold and pin assignments for modularity.

Diagram:



Procedure:

1. Connect the DHT11 sensor to Arduino same as previous experiments.
2. Connect the buzzer with the positive terminal to digital pin 8 and the negative terminal to GND.
3. Write an Arduino program to read temperature periodically from DHT11.
4. Set a temperature threshold value (e.g., 30°C) in the code.
5. If the temperature exceeds the threshold, trigger the buzzer using digital HIGH.
6. Upload the code to the Arduino board.
7. Open the Serial Monitor to verify temperature readings in real time.
8. Optionally, adjust the threshold or buzzer duration for different alert behaviors.
9. Monitor the environment and check buzzer alerts when the temperature exceeds the limit.

Program Code:

```
#include <DHT.h>


// Define pins
#define DHTPIN 7         // DHT11 data pin
#define DHTTYPE DHT11    // DHT sensor type
```

```
#define BUZZER_PIN 3     // Buzzer pin
#define TEMP_THRESHOLD 30.0  // Threshold temperature in °C

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);        // Start Serial Monitor
  dht.begin();               // Initialize DHT sensor
  pinMode(BUZZER_PIN, OUTPUT);  // Buzzer as output
  digitalWrite(BUZZER_PIN, LOW);
}

void loop() {
  delay(2000); // Wait a bit between readings

  float temperature = dht.readTemperature();  // Read temperature in °C

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

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

  if (temperature > TEMP_THRESHOLD) {
    Serial.println("Temperature above threshold! Activating buzzer.");
    digitalWrite(BUZZER_PIN, HIGH);  // Turn on buzzer
  } else {
    digitalWrite(BUZZER_PIN, LOW);   // Turn off buzzer
  }
}
```

Output:

(Attach your experiment image)

Conclusion:

The experiment successfully implemented a temperature monitoring system using the DHT11 sensor and alerted the user with a buzzer when the temperature crossed a predefined threshold, demonstrating real-time environmental response.

Group B: 01

Experiment No: 04

LED Control ON/OFF or Blinking

Problem Statement:

Write a program using Arduino to control LED (One or more ON/OFF). Or Blinking.

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
LED	1	For controlling only one LEDs (for ON/OFF or blinking pattern)
Resistor	1	For each LED, limits current
Breadboard	1	For easy wiring (optional if direct jumpers used)
Male-to-Male Jumper Wires	~6	For power, ground, and signal connections
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

LED (Red):

- Anode (Long Pin) → Arduino Digital Pin 13 (via 220Ω resistor)
- Cathode (Short Pin) → Arduino GND

Theory:

The LED has two legs, the longer of which is the anode (positive) and the shorter of which is the cathode (negative).

LEDs (Light Emitting Diodes) are becoming increasingly popular among a wide range of people. When a voltage is given to a PN Junction Diode, electrons, and holes recombine in the PN Junction and release energy in the form of light (Photons). An LED's electrical sign is comparable to that of a PN Junction Diode. When free electrons in the conduction band recombine with holes in the valence band in forward bias, energy is released in the form of light.

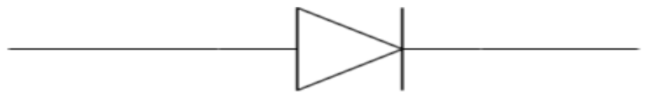


Figure - Diode

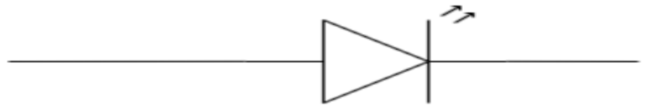
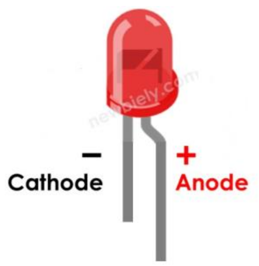


Figure - Light Emitting Diode (LED)



An LED (Light Emitting Diode) is a simple electronic component that emits light when an electric current flows through it. Using an Arduino, one or more LEDs can be controlled to switch ON or OFF, or to blink at specified intervals. The Arduino board communicates with the LEDs via its digital output pins, where a HIGH signal turns the LED ON, and a LOW signal turns it OFF.

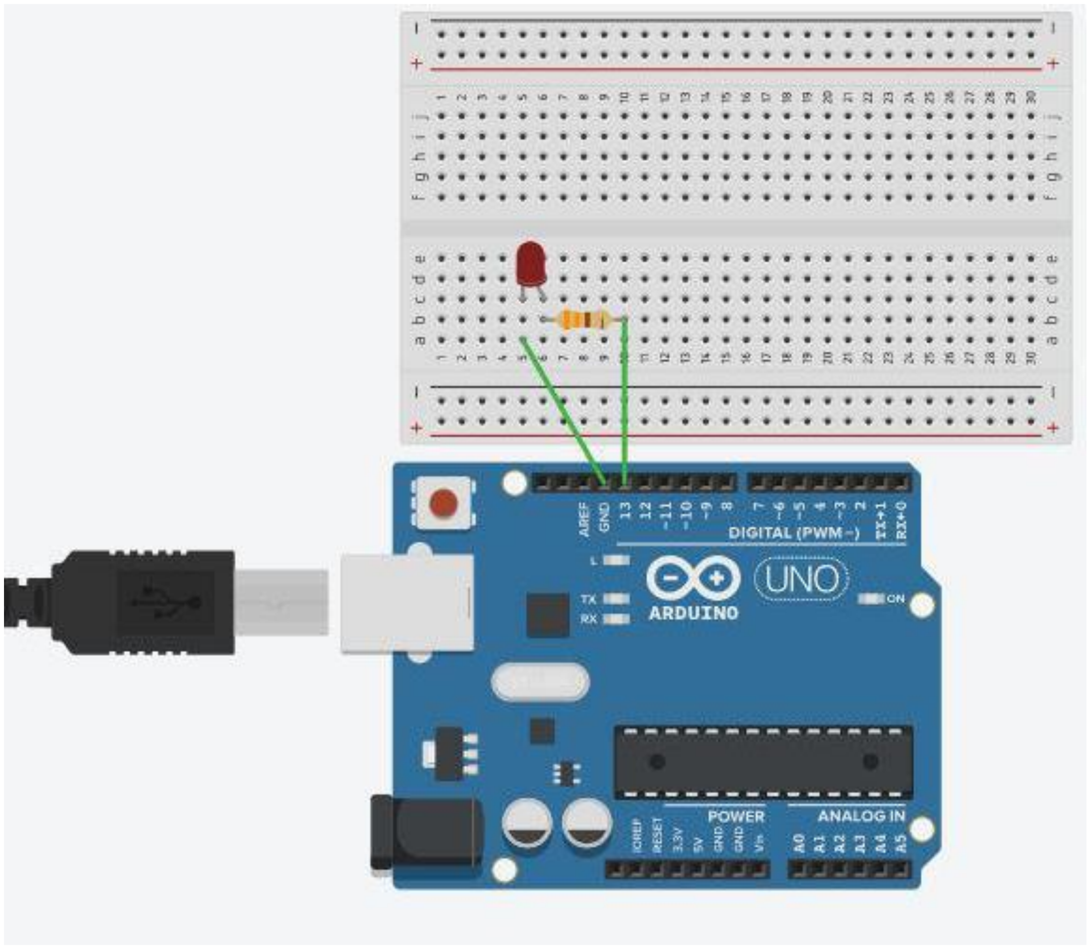
The core concept involves configuring the Arduino pins connected to the LEDs as output pins during initialization. In the continuous execution loop, these pins can be toggled between HIGH and LOW states either based on timing delays to create blinking effects or fixed states to turn LEDs ON or OFF. Blinking is usually achieved by turning the LED ON, waiting for a short delay (such as 500 milliseconds), turning it OFF, and then waiting again, creating a visible flashing effect.

This program also illustrates the fundamental timing control in microcontrollers using delay functions, which pause execution for specific durations. Managing multiple LEDs simultaneously involves controlling each pin individually, allowing for complex patterns or simple indicator signals. The program demonstrates digital output control, which is the basis for many embedded system applications.

- **pinMode(pin, OUTPUT)**: Sets the LED pin as an output to control it.
- **digitalWrite(pin, HIGH/LOW)**: Turns the LED ON or OFF by setting the pin voltage.
- **delay(milliseconds)**: Creates a pause between turning the LED ON and OFF to produce a blinking effect.

Overall, this approach introduces basic digital I/O operations with microcontrollers, essential for learning how to interface with simple output devices and control them through software.

Diagram:



Procedure:

1. Connect the LED’s longer leg (anode) to Arduino digital pin 13.
2. Connect the shorter leg (cathode) to a 220Ω resistor.
3. Connect the other end of the resistor to the Arduino GND pin.
4. Write the Arduino program to control the LED: turn it ON, OFF, or make it blink with delays.
5. Upload the code to the Arduino board using the Arduino IDE.
6. Power the Arduino and observe the LED behavior.
7. Verify the LED turns ON steadily, remains OFF, or blinks according to the program.

Program Code:

```
int led1 = 13;
void setup() {
  pinMode(led1, OUTPUT);
}
void loop() {
  digitalWrite(led1, HIGH);
  delay(500);
  digitalWrite(led1, LOW);
  delay(500);
}
```

Output:

(Attach your experiment image)

Conclusion:

The LED control program successfully demonstrated how to use digital pins on an Arduino to turn LEDs ON/OFF and blink them, introducing basic output control.

Group B: 02

Experiment No: 05

LED Control via Serial Input

Problem Statement:

Create a program so that when the user enters ‘b’ the green light blinks, ‘g’ the green light is illuminated ‘y’ the yellow light is illuminated and ‘r’ the red light is illuminated

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
LED(Red, Yellow, Green)	3	For controlling LEDs (for ON/OFF or blinking pattern)
Resistor	3	For each LED, limits current
Breadboard	1	For easy wiring (optional if direct jumpers used)
Male-to-Male Jumper Wires	~6	For power, ground, and signal connections
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

□ Red LED:

- Anode → Arduino Digital Pin 3 (via 220Ω resistor)
- Cathode → Arduino GND

□ Yellow LED:

- Anode → Arduino Digital Pin 4 (via 220Ω resistor)
- Cathode → Arduino GND

□ Green LED:

- Anode → Arduino Digital Pin 5 (via 220Ω resistor)
- Cathode → Arduino GND

Theory:

This approach expands on controlling LEDs by adding interactivity, where an Arduino listens for input commands sent through a serial communication channel, typically from a computer’s serial monitor. The program reads characters entered by the user and performs actions based on the received commands to control different colored LEDs.

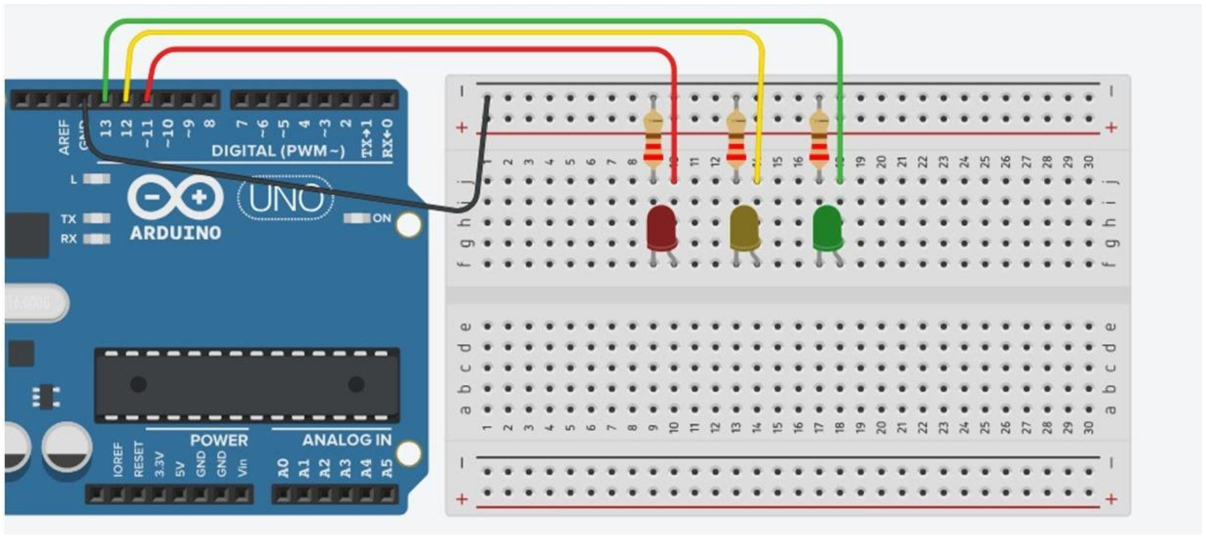
The hardware setup involves connecting multiple LEDs—each representing a color such as green, yellow, and red—to different Arduino pins configured as outputs. The program initializes serial communication to receive data at a defined baud rate. In the main loop, it continuously checks if any serial data has arrived. Upon receiving a character (like 'b', 'g', 'y', or 'r'), it uses conditional logic to decide which LED to illuminate or blink.

For instance, receiving 'b' might make the green LED blink, while 'g', 'y', and 'r' commands turn on the green, yellow, and red LEDs respectively. This involves turning off other LEDs to avoid multiple lights being on simultaneously unless intended. The blinking LED is controlled through toggling the pin state with delays, while steady illumination involves setting the corresponding pin HIGH.

- **Serial.begin(9600)**: Initializes serial communication to receive commands from the user.
- **pinMode(pin, OUTPUT)**: Configures pins connected to different LEDs as outputs.
- **Serial.available()**: Checks if user input is available on the serial port.
- **Serial.read()**: Reads a single character input from the user.
- **digitalWrite(pin, HIGH/LOW)**: Controls LEDs based on input.
- **delay(milliseconds)**: Used to create blinking by toggling LED state with timed pauses.
- **Conditional Statements (if/else if)**: Decide which LED to activate or blink based on the input character.

This method demonstrates serial communication basics, character handling, and conditional control flow, showing how microcontrollers can interact with users dynamically. It highlights the use of serial input as a control interface beyond simple button presses.

Diagram:



Procedure:

1. Connect three LEDs (Red, Yellow, Green) each with a 220Ω resistor in series.
2. Connect the LEDs’ anodes to Arduino digital pins 11 (Red), 12 (Yellow), and 13 (Green).
3. Connect all the cathodes to the Arduino GND pin.
4. Write an Arduino program that listens to serial input characters.
5. Map characters to LED actions:
    - ‘b’ makes Green LED blink
    - ‘g’ turns Green LED ON
    - ‘y’ turns Yellow LED ON
    - ‘r’ turns Red LED ON
6. Upload the code and open the Serial Monitor.
7. Send characters (‘b’, ‘g’, ‘y’, or ‘r’) through Serial Monitor.
8. Observe the corresponding LEDs light up or blink as per input.
9. Test all commands to ensure proper LED response.

Program Code:

```
int redLED = 11;
int yellowLED = 12;
int greenLED = 13;

void setup() {
  Serial.begin(9600);
  pinMode(redLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  Serial.println("Enter r, y, g, or b:");
}

void loop() {
  if (Serial.available()) {
    char input = Serial.read();
    digitalWrite(redLED, LOW);
    digitalWrite(yellowLED, LOW);
    digitalWrite(greenLED, LOW);

    if (input == 'r') {
      digitalWrite(redLED, HIGH);
    } else if (input == 'y') {
      digitalWrite(yellowLED, HIGH);
    } else if (input == 'g') {
      digitalWrite(greenLED, HIGH);
    } else if (input == 'b') {
      for (int i = 0; i < 3; i++) {
        digitalWrite(greenLED, HIGH);
        delay(300);
        digitalWrite(greenLED, LOW);
        delay(300);
      }
    }
  }
}
```

Output:

(Attach your experiment image)

Conclusion:

This experiment allowed user interaction via serial input to control multiple colored LEDs, enhancing understanding of conditional logic and serial communication.

Group B: 03

Experiment No: 06

Square Number Calculator

Problem Statement:

Write a program that asks the user for a number and outputs the number squared that is entered.

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

- **No external wiring needed.**
- Only connect the Arduino to your PC via USB cable.

Theory:

This program introduces the concept of receiving numeric input from the user through serial communication and performing arithmetic processing on it. The Arduino waits for the user to input a number, reads this data as a string, converts it into an integer, calculates its square, and then sends the result back via the serial monitor.

The main components are the serial interface and the Arduino's internal computational capability. Upon setup, serial communication is initialized, and the program prompts the user to enter a number. In the continuous loop, it listens for serial input, accumulates characters until a newline or termination character is detected, then processes the accumulated string.

Conversion from string to integer uses built-in functions. Once the number is parsed, the program multiplies it by itself to find the square. The result is formatted and sent back to the serial monitor, providing immediate feedback to the user.

- **Serial.begin(9600)**: Sets up serial communication.
- **Serial.available()**: Detects if the user has sent input.
- **Serial.readStringUntil('\n')**: Reads the full line of input as a string.
- **String.toInt()**: Converts the input string to an integer.
- **Arithmetic Operation (num * num)**: Calculates the square of the input number.
- **Serial.print() / Serial.println()**: Sends the output back to the user over serial.

This example emphasizes basic data input/output handling, string manipulation, and arithmetic operations on microcontrollers, reinforcing how simple computations can be combined with user interaction.

Procedure:

1. Write an Arduino program to read integer input from the Serial Monitor.
2. Prompt the user to enter a number.
3. When a number is received, calculate its square by multiplying the number by itself.
4. Output the squared result back to the Serial Monitor.
5. Upload the code to Arduino.
6. Open Serial Monitor and set baud rate as specified in the code.
7. Enter any integer number in the input box and press Enter.
8. Observe the squared value displayed on the Serial Monitor.
9. Repeat with various numbers to validate the program’s correctness.

Program Code:

```
void setup() {
  Serial.begin(9600);
  Serial.println("Enter a number to square:");
}

void loop() {
  if (Serial.available() > 0) {
    String inputStr = Serial.readStringUntil('\n');
    int num = inputStr.toInt();
    int square = num * num;
    Serial.print("Square of ");
    Serial.print(num);
    Serial.print(" is: ");
    Serial.println(square);
    Serial.println("Enter another number:");
  }
}
```

Output:

(Attach your experiment image)

Conclusion:

The program correctly received numeric input through the serial monitor and displayed the squared result, showcasing how to handle user input and perform calculations using Arduino.

Group B: 04

Experiment No: 07

Temperature Reading (Celsius)

Problem Statement:

Write a program read the temperature sensor and send the values to the serial monitor on the computer.

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
DHT11 (3-pin module)	1	Already includes internal pull-up resistor
Breadboard	1	For easy wiring (optional if direct jumpers used)
Male-to-Male Jumper Wires	~6	For power, ground, and signal connections
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

DHT11 (3-pin version):

VCC (Pin 1) → Arduino 5 V
DATA (Pin 2) → Arduino Digital Pin 2
GND (Pin 3) → Arduino GND

Theory:

A DHT lindlutes a Didiennand digital sensor connecting an Ardu-no to read and senil ermperature atars to the Seria-Montior, A to peritment arca inm DHTII sare of to read and sere's epnillitasin date to the Serial Monttor. The digit tal sensor a functior. of a full digitad seneer con-line digital signal processed by us Ardumo and than tranez-mined to the Serial-Monttor. The experiment conners inter-facing with secsers, reading data, and ertusilotleing a bridge bet ween phys-cark physical measurements and.

The DHT11 is a popular digital temperature and humidity sensor widely used in embedded systems due to its simplicity and affordability. Unlike analog sensors, the DHT11 communicates temperature data digitally using a single-wire communication protocol, which provides temperature readings in degrees Celsius along with humidity values.

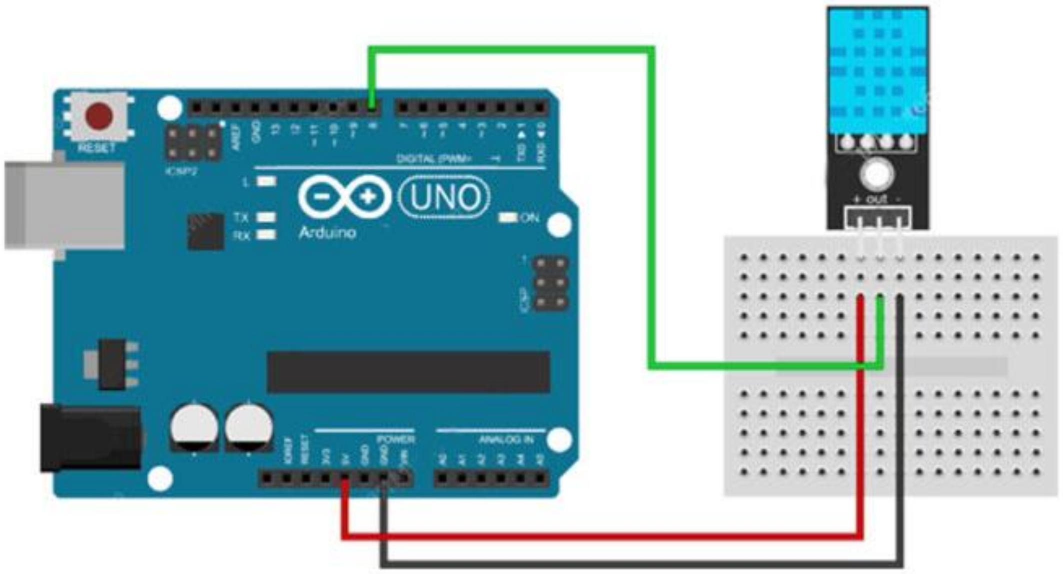
The Arduino interfaces with the DHT11 sensor by sending a start signal and then reading the data pulses transmitted by the sensor. The sensor sends a fixed-format data stream representing temperature and humidity, which the Arduino decodes using specific timing and protocols, often simplified by using dedicated libraries.

Once the temperature data is acquired, the program displays the value through the serial monitor, enabling real-time monitoring on a connected computer. The sensor readings are typically taken at intervals (like every second or two) to provide updated environmental information.

- **DHT dht(pin, type)**: Creates a sensor object linked to a pin and sensor type.
- **dht.begin()**: Initializes communication with the DHT11 sensor.
- **dht.readTemperature()**: Reads temperature data from the sensor in Celsius.
- **isnan(value)**: Checks if the sensor reading is valid.
- **Serial.begin(9600)** and **Serial.print()/println()**: Used to communicate sensor readings.
- **delay(2000)**: Waits 2 seconds between readings (sensor's minimum interval).

This approach illustrates digital sensor communication, data acquisition, and serial data transmission, which are essential skills for integrating smart sensors into microcontroller projects.

Diagram:



Procedure:

1. Connect the DHT11 sensor's VCC pin to Arduino 5V.
2. Connect the DATA pin of the DHT11 to Arduino digital pin 8.
3. Connect the GND pin of the DHT11 to Arduino GND.
4. Include the DHT library in your Arduino IDE.
5. Write the Arduino program to initialize the sensor and read temperature in Celsius.
6. Setup serial communication at 9600 baud.

7. Upload the code to Arduino.
8. Open Serial Monitor to view temperature readings.
9. Observe and verify temperature values printed every 2 seconds.
10. Test the sensor by exposing it to different temperatures to confirm accuracy.

Program Code:

```
#include <DHT.h>

#define DHTPIN 8       // Digital pin connected to the DHT11 sensor data pin
#define DHTTYPE DHT11  // DHT 11 sensor type

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate
  dht.begin();         // Initialize the DHT11 sensor
}

void loop() {
  float temperature = dht.readTemperature(); // Read temperature in Celsius

  if (isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
  } else {
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println(" °C");
  }

  delay(2000); // Wait 2 seconds before next reading
}
```

Output:

(Attach your experiment image)

Conclusion:

The DHT11 temperature sensor successfully read temperature data and displayed it on the Serial Monitor, enabling basic sensor interfacing and data output.

Group B: 05

Experiment No: 08

Temperature with Min/Max & °F Display

Problem Statement:

Write a program so it displays the temperature in Fahrenheit as well as the maximum and minimum temperatures it has seen.

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
DHT11 (3-pin module)	1	Already includes internal pull-up resistor
Breadboard	1	For easy wiring (optional if direct jumpers used)
Male-to-Male Jumper Wires	~6	For power, ground, and signal connections
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

DHT11 (3-pin version):

- VCC (Pin 1) → Arduino 5 V
- DATA (Pin 2) → Arduino Digital Pin 8
- GND (Pin 3) → Arduino GND

Theory:

Building upon the DHT11 sensor reading process, this program not only acquires the current temperature but also keeps track of the maximum and minimum temperatures recorded during operation. The Arduino stores these values in variables and updates them when new readings surpass previously stored extremes.

Since the DHT11 provides temperature data in degrees Celsius, the program converts this to Fahrenheit using the standard formula for users who prefer imperial units. The current, maximum, and minimum temperature values are transmitted continuously via serial communication for display on the serial monitor.

This functionality enables effective monitoring of temperature trends over time, making the system useful for applications requiring environmental awareness, such as climate control or weather stations.

 All functions from previous Experiment are used here, plus:

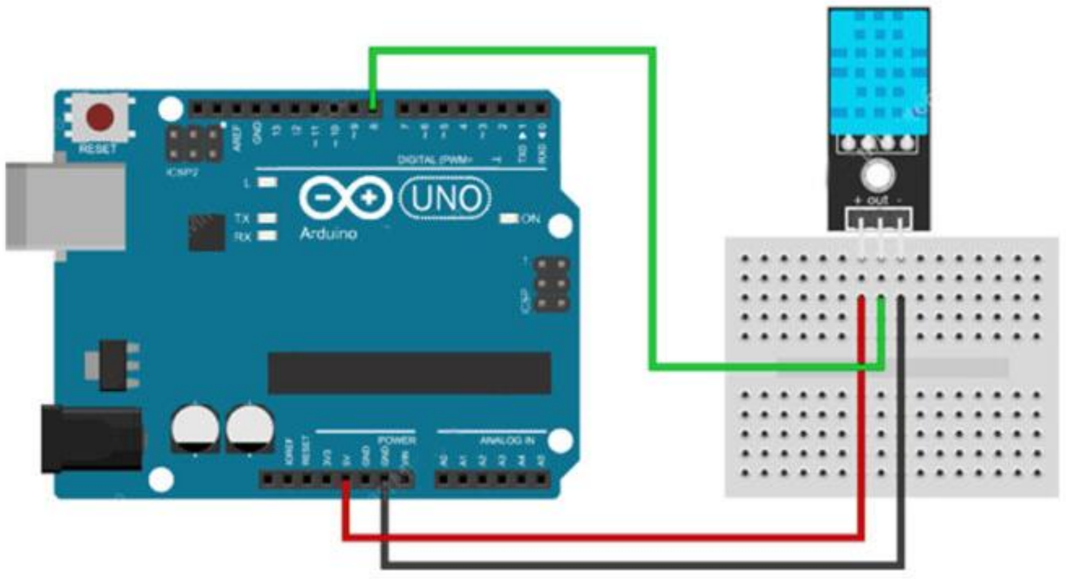
 Variable comparison and assignment (if (tempC > maxTempC)): To update maximum and minimum temperature records.

 Arithmetic conversion (tempF = tempC * 9/5 + 32): Converts Celsius to Fahrenheit.

 Multiple **Serial.print()** calls to display current, max, and min temperatures clearly.

The use of the DHT11 sensor highlights digital communication protocols, data handling, and real-time environmental monitoring.

Diagram:



Procedure:

1. Connect the DHT11 sensor to Arduino as in Experiment 4.
2. Write or modify the Arduino program to:
   - Read temperature in Celsius.
   - Convert Celsius to Fahrenheit.
   - Track and update minimum and maximum temperature readings.
3. Setup serial communication.
4. Upload the program.
5. Open Serial Monitor and observe:
   - Current temperature in both Celsius and Fahrenheit.
   - Minimum and maximum temperature recorded since reset.
6. Test the system over time by changing the temperature environment.

7. Confirm that min and max values update correctly with changing temperatures.

Program Code:

```
#include <DHT.h>

#define DHTPIN 8
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

float maxTempC = -1000.0; // initialize to a very low temp
float minTempC = 1000.0;  // initialize to a very high temp

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

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

  if (isnan(tempC)) {
    Serial.println("Failed to read from DHT sensor!");
  } else {
    // Update max and min temperatures
    if (tempC > maxTempC) maxTempC = tempC;
    if (tempC < minTempC) minTempC = tempC;

    float tempF = tempC * 9.0 / 5.0 + 32.0;

    Serial.print("Current Temp: ");
    Serial.print(tempC);
    Serial.print(" °C / ");
    Serial.print(tempF);
    Serial.println(" °F");

    Serial.print("Max Temp: ");
    Serial.print(maxTempC);
    Serial.print(" °C");

    Serial.print(" | Min Temp: ");
    Serial.print(minTempC);
    Serial.println(" °C");
  }

  delay(2000);
}
```

Output:

(Attach your experiment image)

Conclusion:

The program accurately tracked and displayed the current, maximum, and minimum temperatures recorded, introducing memory of values and conditional logic.

Group B: 06

Experiment No: 09

Temperature Graphing on Serial Monitor

Problem Statement:

Write a program to show the temperature and shows a graph of the recent measurements.

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
DHT11 (3-pin module)	1	Already includes internal pull-up resistor
Breadboard	1	For easy wiring (optional if direct jumpers used)
Male-to-Male Jumper Wires	~6	For power, ground, and signal connections
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

DHT11 (3-pin version):

VCC (Pin 1) → Arduino 5 V

DATA (Pin 2) → Arduino Digital Pin 8

GND (Pin 3) → Arduino GND

Theory:

This setup leverages the DHT11 sensor's periodic temperature readings and stores a series of recent measurements in a buffer or array inside the Arduino. This collection of data points allows for observing temperature trends over a period rather than just individual readings.

The Arduino transmits this series of stored temperature values over the serial port. While the Arduino cannot graphically display the data by itself, external applications or serial plotters can use this information to render a temperature graph, visually representing changes over time.

Managing a moving window of data requires efficient memory handling and timing to update the buffer with new sensor readings and discard the oldest ones. This continuous data streaming and buffering technique exemplify how embedded systems can support more advanced monitoring and visualization tasks.

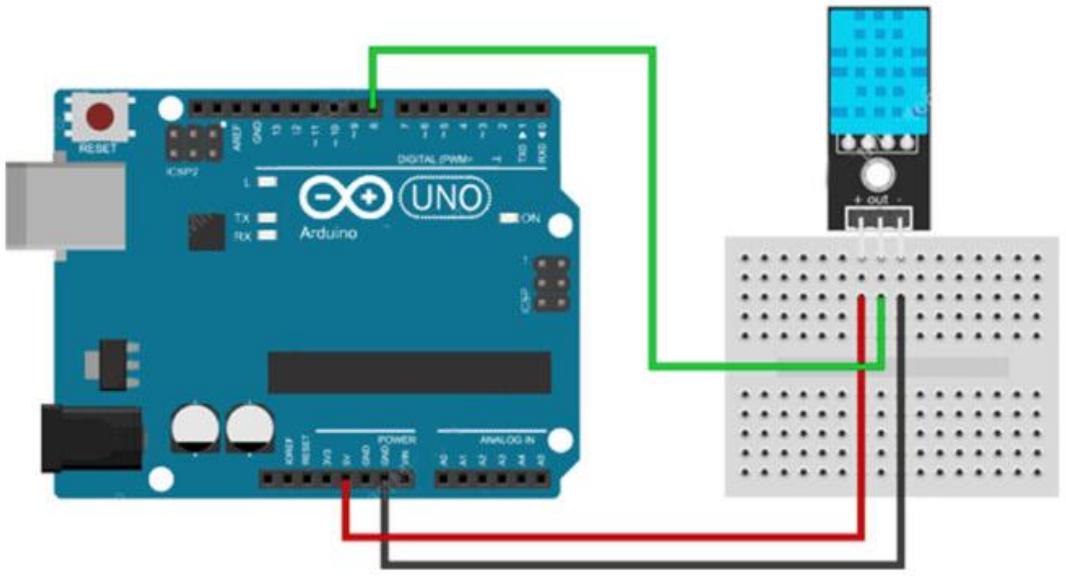
- Uses DHT library methods to read temperature as in Experiment 4.
- **Serial.println()** repeatedly outputs temperature values formatted for plotting.

 delay(2000) manages reading intervals.

 The program supports external tools to visualize temperature trends by continuously sending data.

The use of the DHT11 sensor in this context demonstrates how digital sensor data can be integrated with time-series data management for effective environmental monitoring.

Diagram:



Procedure:

1. Connect the DHT11 sensor to Arduino same as previous experiments.
2. Write an Arduino program to read temperature periodically.
3. Setup serial communication to send temperature data continuously.
4. Upload the code.
    a. Open the Arduino IDE Serial Plotter (not just Serial Monitor).
    b. In Arduino IDE, go to **Tools** → **Serial Plotter**.
    c. Set the baud rate to **9600**.
5. Observe live graph plotting temperature over time.
6. Optionally, tweak the code to adjust sample intervals or format data for better graphing.
7. Monitor the graph for trends and fluctuations.
8. Use this method to visualize temperature changes intuitively in real time.

Program Code:

```
#include "DHT.h"
```

```
#define DHTPIN   8      // digital pin connected to DATA
#define DHTTYPE  DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
  Serial.println("Temperature"); // plotter label
}

void loop() {
  delay(2000); // sensor needs 1–2 seconds

  float t = dht.readTemperature(); // °C
  if (isnan(t)) return;

  Serial.println(t); // send to Serial Plotter
}
```

Output:

(Attach your experiment image)

Conclusion:

The experiment displayed real-time temperature values as a graph using the Serial Plotter, allowing a visual understanding of sensor data trends over time.

Group C: 01

Experiment No: 10

Problem Statement:

Write an application to control the operation of hardware simulated traffic signals.

Components Required:

Component	**Quantity**	**Purpose**
Arduino Uno	1	Or any compatible Arduino board (Nano, Mega, etc.)
LED(Red, Yellow, Green)	3	For controlling LEDs (for ON/OFF or blinking pattern)
Resistor	3	For each LED, limits current
Breadboard	1	For easy wiring (optional if direct jumpers used)
Male-to-Male Jumper Wires	~6	For power, ground, and signal connections
USB Cable (Type B)	1	To connect Arduino Uno to your computer
Computer	1	To run the Arduino IDE and view Serial Plotter

Wiring:

- Red LED:
    - Anode → Arduino Digital Pin 8 (via 220Ω resistor)
    - Cathode → Arduino GND
- Yellow LED:
    - Anode → Arduino Digital Pin 9 (via 220Ω resistor)
    - Cathode → Arduino GND
- Green LED:
    - Anode → Arduino Digital Pin 10 (via 220Ω resistor)
    - Cathode → Arduino GND

Theory:

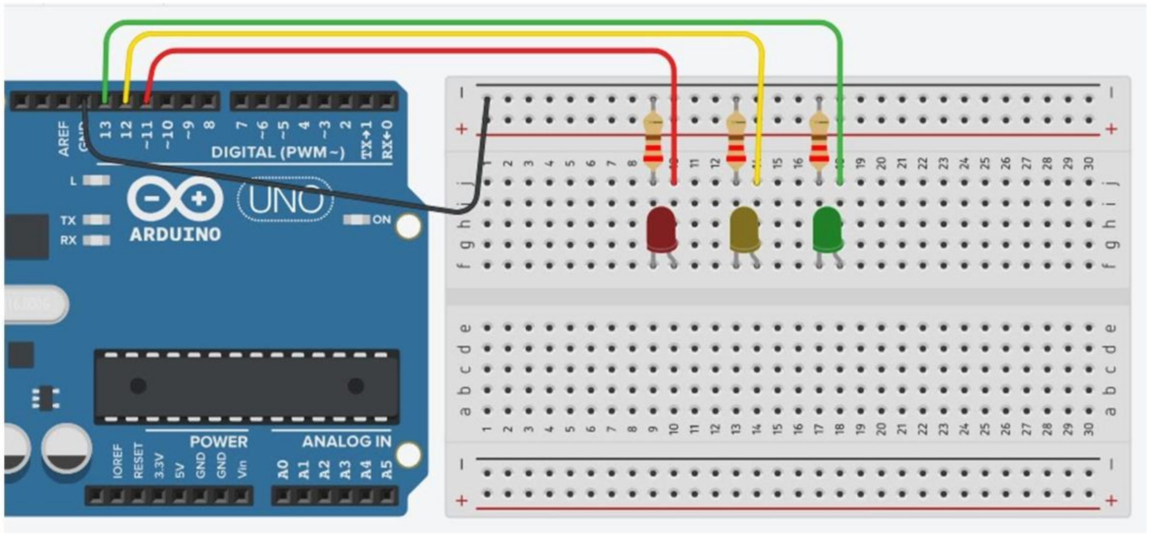
This program simulates a basic traffic light system using three LEDs representing red, yellow, and green lights, controlled in a cyclic pattern. The goal is to replicate the operation of traffic signals using delays to simulate the duration each light stays ON.

The setup() function configures the LED pins as outputs. Inside the loop(), the green LED is turned ON for 5 seconds to simulate the “go” signal. It’s followed by the yellow LED for 2 seconds to signal a transition. Then the red LED is activated for 5 seconds to simulate the "stop" signal, followed again by a 2-second yellow transition before returning to green.

Key methods and functions used include:

- pinMode() to configure LED pins as outputs.
- digitalWrite() to turn individual LEDs ON or OFF.
- delay() to manage timing and control signal duration.

Diagram:



Procedure:

1. Connect the red, yellow, and green LEDs to Arduino using digital pins 11, 12, and 13 respectively.
2. Add 220Ω resistors in series with each LED to limit current.
3. Write an Arduino program to simulate traffic light sequencing: green → yellow → red.
4. Add delays (e.g., 3s for green, 1s for yellow, 3s for red) to simulate timing logic.
5. Upload the code to Arduino board.
6. Observe the LEDs lighting up in proper traffic signal sequence.
7. Optionally, enhance the simulation by adding a pedestrian button or sensor input.
8. Run the setup for continuous cycling and observe real-time transitions.

Program Code:

```
// Define pin numbers for the LEDs

const int redLED = 11;
const int yellowLED = 12;
const int greenLED = 13;

void setup() {
  // Set LED pins as OUTPUT
  pinMode(redLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
}

void loop() {
  // Green light ON for 5 seconds
  digitalWrite(greenLED, HIGH);
  digitalWrite(redLED, LOW);
  digitalWrite(yellowLED, LOW);
  delay(5000);

  // Yellow light ON for 2 seconds (before red)
  digitalWrite(greenLED, LOW);
  digitalWrite(yellowLED, HIGH);
  delay(2000);

  // Red light ON for 5 seconds
  digitalWrite(yellowLED, LOW);
  digitalWrite(redLED, HIGH);
  delay(5000);

  // Yellow light ON for 2 seconds (before green)
  digitalWrite(redLED, LOW);
  digitalWrite(yellowLED, HIGH);
  delay(2000);
  // Repeat
}
```

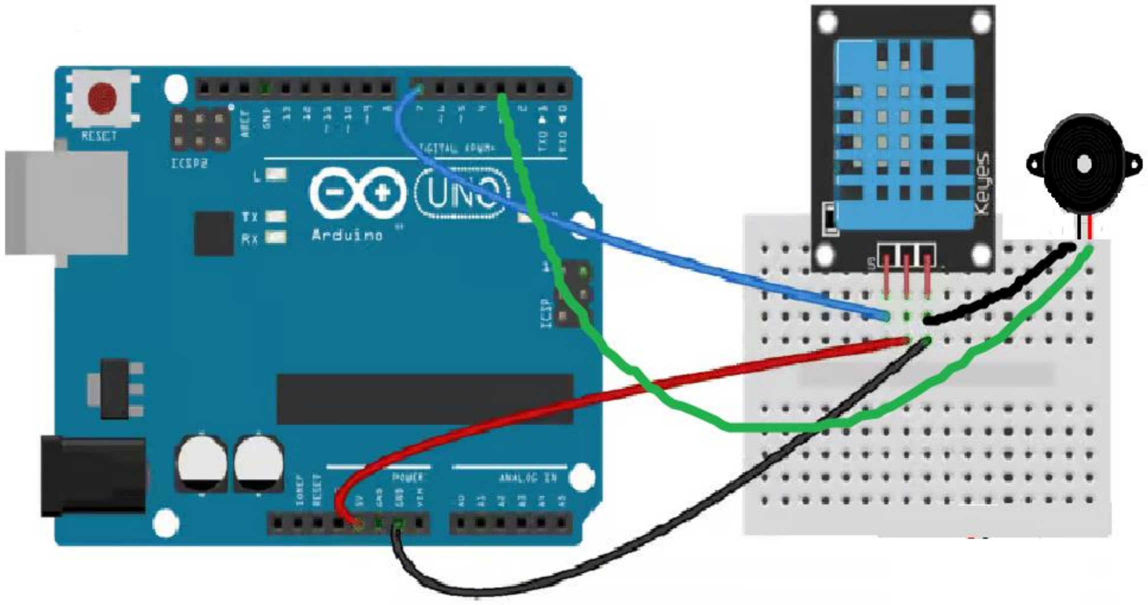
Output:

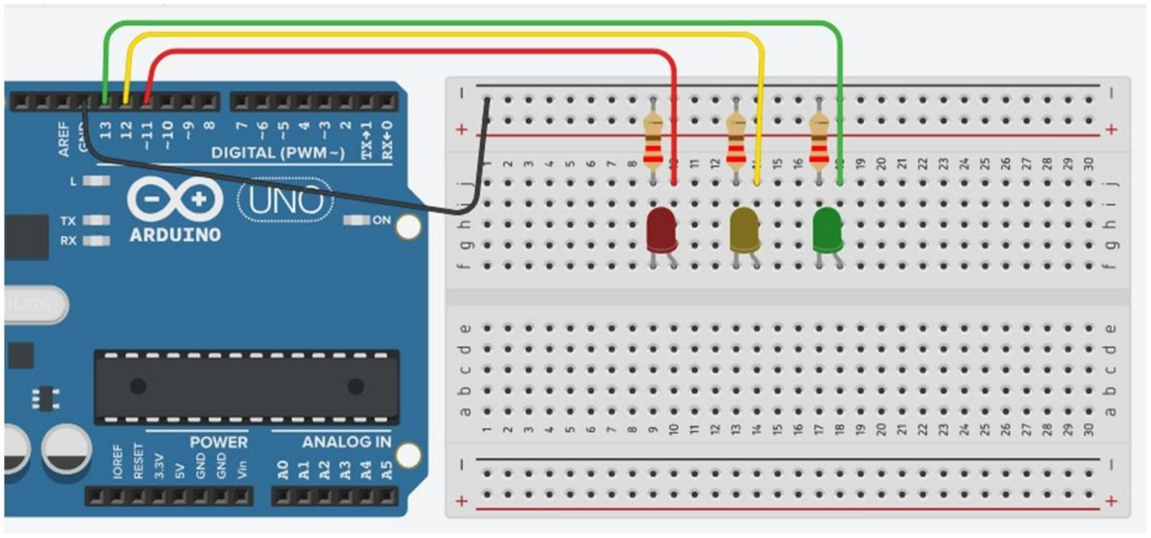
(Attach your experiment image)

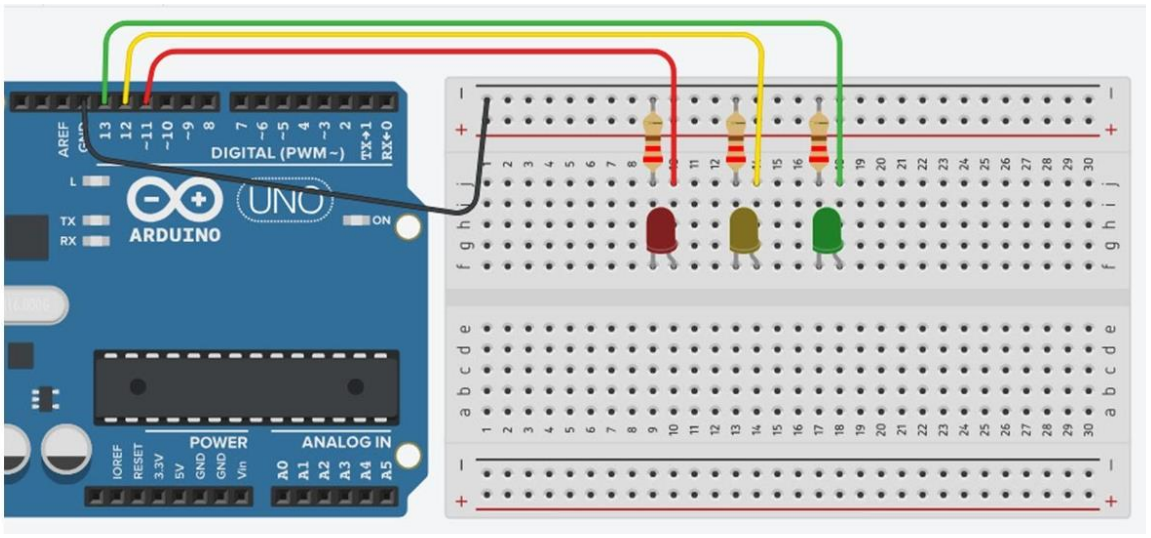
Conclusion:

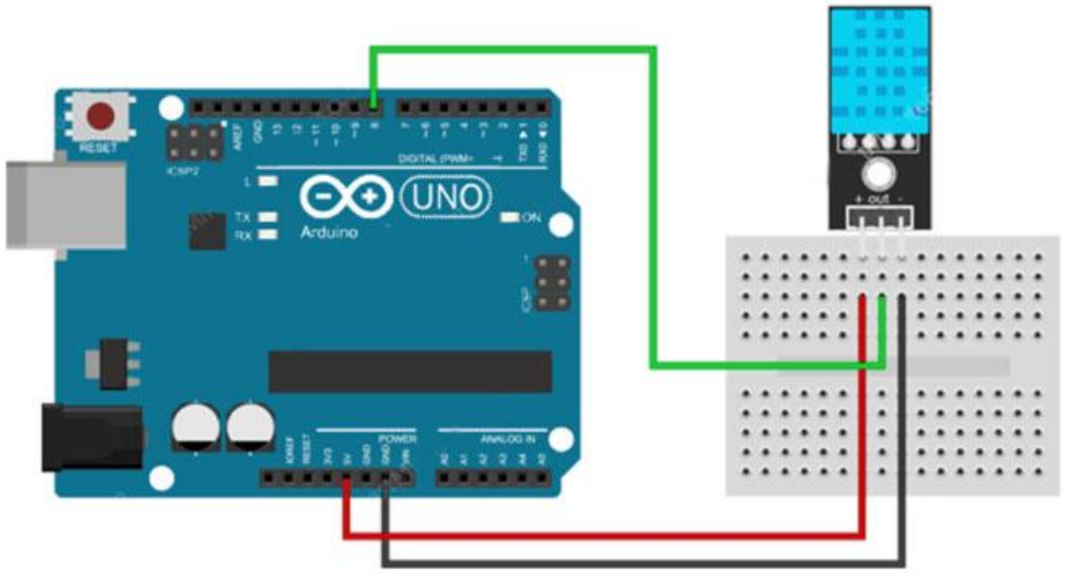
The experiment successfully simulated a working traffic signal system using an Arduino and LEDs, demonstrating control over timing sequences and output logic suitable for real-world automation.

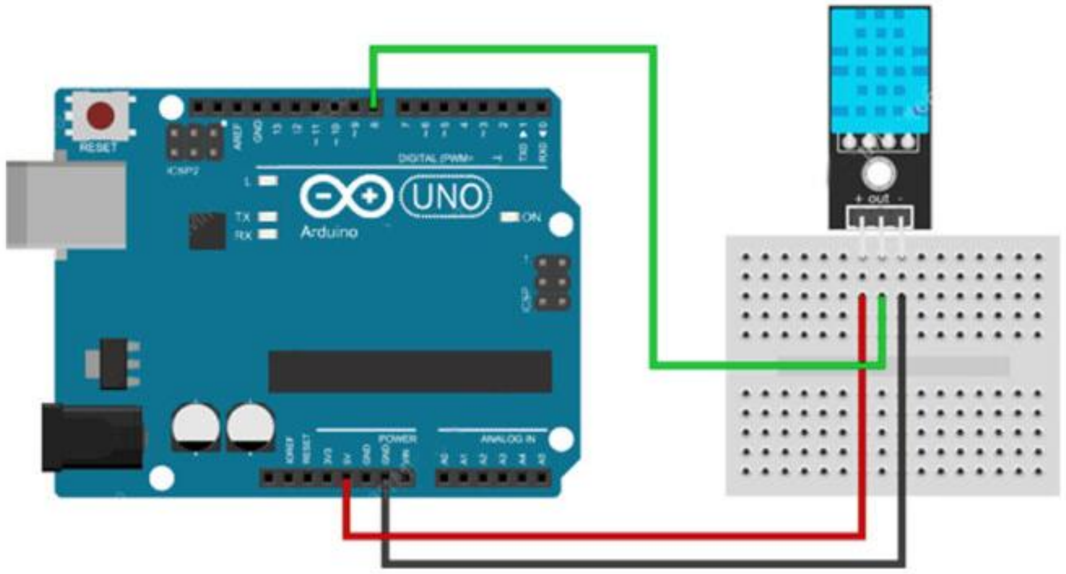
Print the image in color, then cut and paste it as directed by the manual.

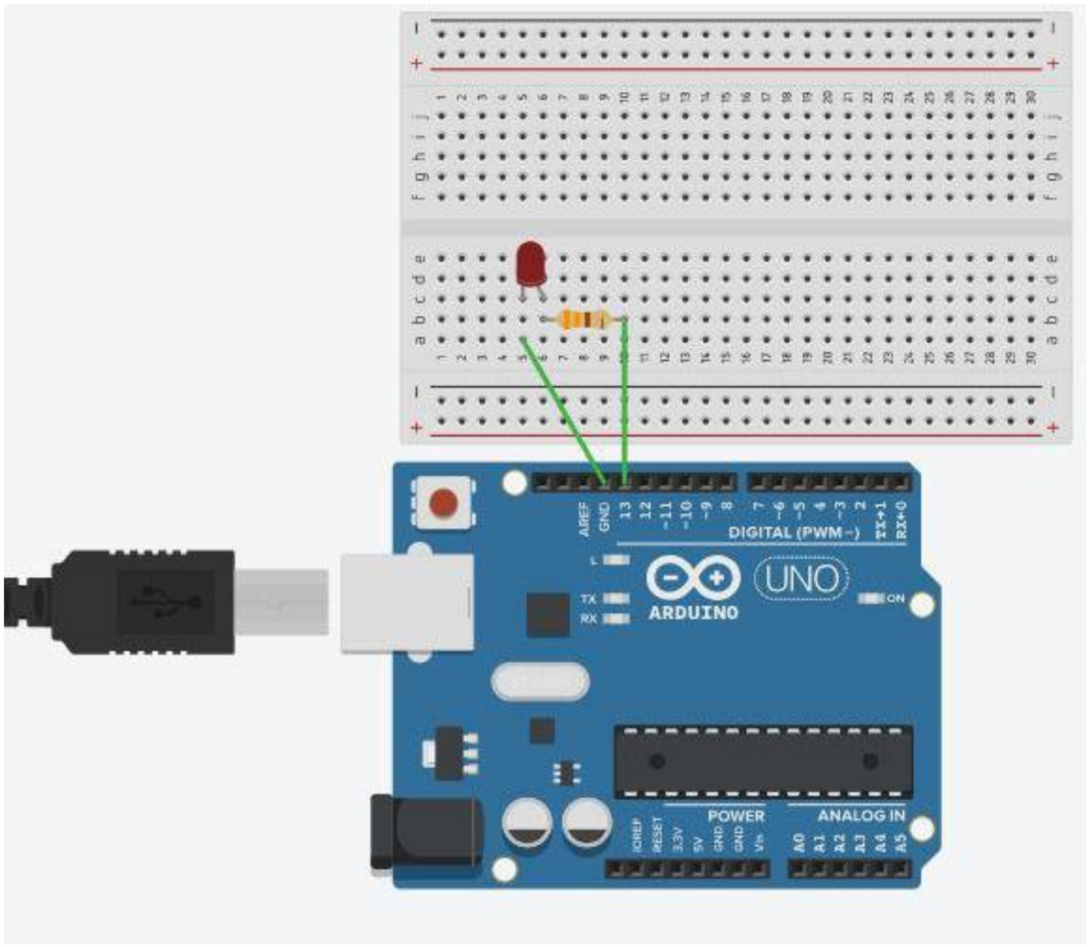


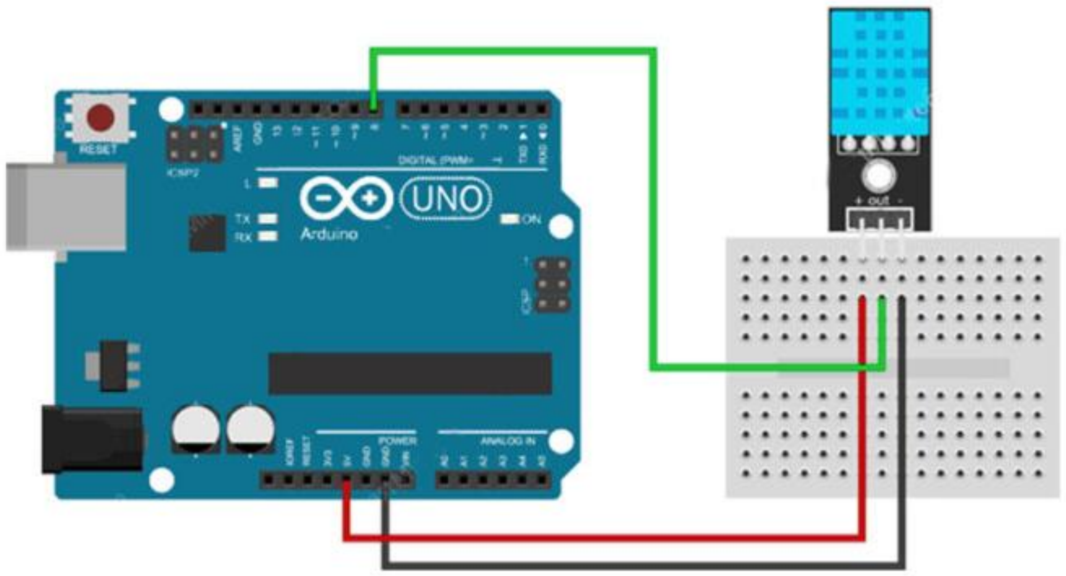


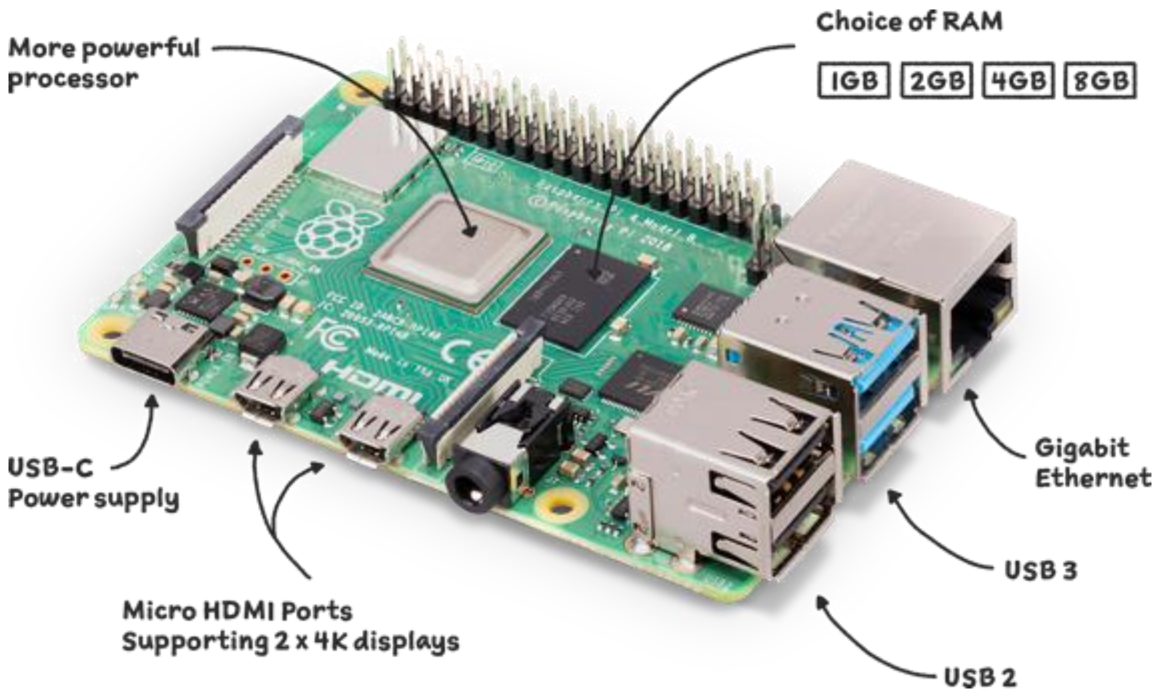

AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX▸1
RX◂0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN

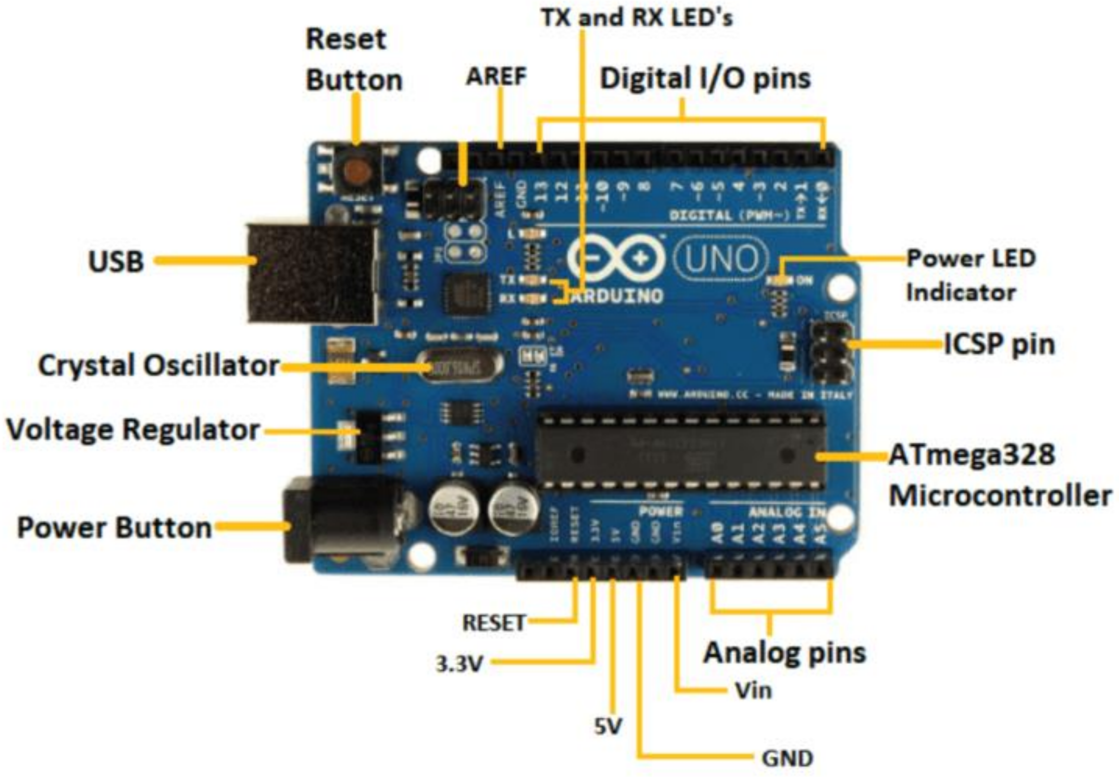

RESET
ICSP2
AREF
GND
DIGITAL (PWM= ~)
Arduino
UNO
ON
TX
RX
L
ICSP
POWER
ANALOG IN
+ out -

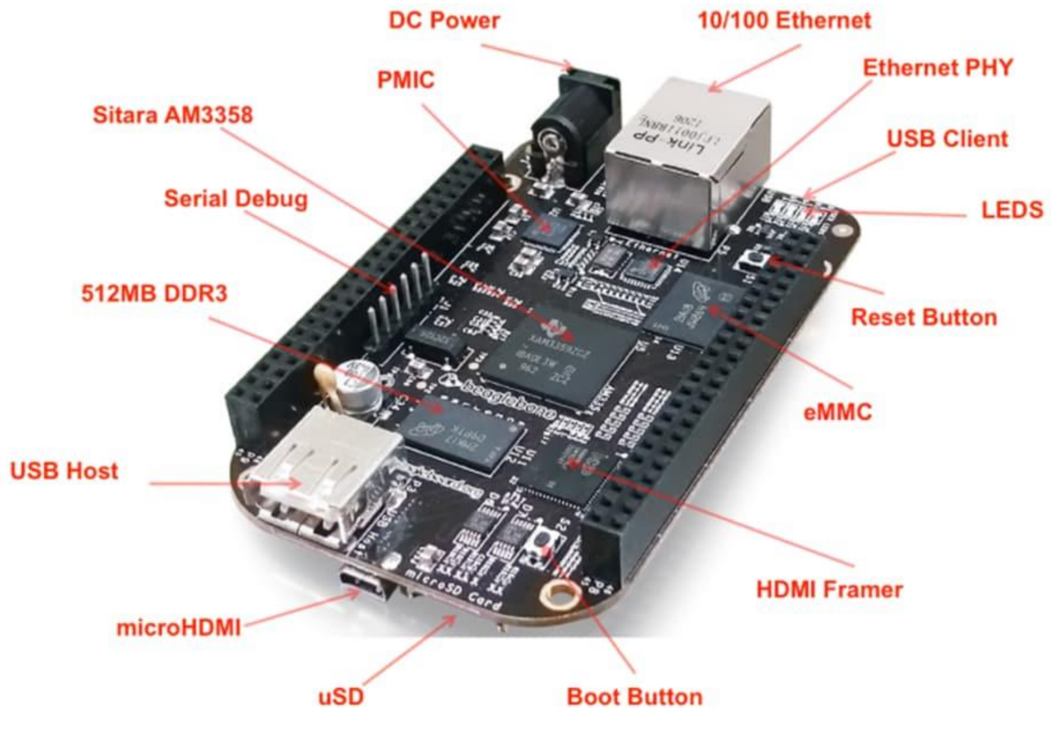





RESET
ICSP2
DIGITAL (PWM~)
UNO
Arduino
TX
RX
ON
ICSP
POWER
ANALOG IN
+ out -


More powerful processor
Choice of RAM
1GB
2GB
4GB
8GB
USB-C Power supply
Micro HDMI Ports Supporting 2 x 4K displays
Gigabit Ethernet
USB 3
USB 2


Reset Button
TX and RX LED's
AREF
Digital I/O pins
USB
Power LED Indicator
ICSP pin
Crystal Oscillator
Voltage Regulator
ATmega328 Microcontroller
Power Button
RESET
3.3V
Analog pins
Vin
5V
GND


DC Power
10/100 Ethernet
PMIC
Ethernet PHY
Sitara AM3358
USB Client
Serial Debug
LEDS
512MB DDR3
Reset Button
eMMC
USB Host
HDMI Framer
microHDMI
uSD
Boot Button